A Software Tool for Assessing the Performance of Water Distribution System Solution Methods based on Graph Theory

Mengning Qiu

Thesis submitted in total fulfilment for the degree of
Doctor of Philosophy

School of Civil, Environmental and Mining Engineering
The University of Adelaide

May 2018

Abstract

Hydraulic simulation models have been used to simulate the steady-state of a water distribution system (WDS) for serval decades. These models have been used in WDS simulation toolkits and have played a critical role in the design, operation, and management of WDSs in industry and research. In recent years, a number of graph theory based WDS solution methods have been developed. These methods have explored the structural properties (both matrix and graph) of the problem to improve the speed and reliability of WDS simulations. One question that naturally arises is which method or combination of methods should be applied?

In this thesis, a WDS simulation testbed, called WDSLib, has been developed as a tool that can be used to answer the above question. WDSLib is an extensible simulation toolkit for the steady-state analysis of a WDS. It has been created using modularised object-oriented design and implemented in C++ programming language. WDSLib can be used (1) to implement, test, and compare different solution methods, (2) to focus the research on the most time-consuming parts of a solution method, (3) to guide the choice of solution method when multiple simulation runs are used (such as occurs in a genetic algorithm run).

WDSLib has been used to investigate the performance of four solution methods, namely the global gradient algorithm (GGA), the reformulated co-tree flows method, the GGA with the forest-core partitioning algorithm (FCPA), and the RCTM with the FCPA, on eight case study benchmark networks with between 934 and 19647 pipes and between 848 and 17971 nodes. The results can be used to inform the choice of the solution method for a given combination of the network features under different design settings. This work also demonstrates how to (1) use the WDSLib to implement, test, and benchmark the existing solution methods and (2) use the results to determine which method or combination of methods to used under a setting of interest.

A new graph theory algorithm, called the bridge-block partitioning algorithm (BBPA), has been proposed which further partitions the WDS network in a number of bridge components and a number of block components. The BBPA is also implemented in the WDSLib in order to ensure a fair comparison with the existing methods. The BBPA is a pre-processing and post-processing method, the use of which provides significant advantages over the current methods in terms of both the computational speed and the reliability of the solution. This work also demonstrates how to (1) use the WDSLib to implement, test, and benchmark the new solution method and (2) use the WDSLib to demonstrate the efficiency of new method without having to reengineer the content of shared WDSLib functions and data representations.

This page is intentionally left blank

Contents

This page is intentionally left blank

Statement of Originality

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.

I acknowledge that copyright of published works contained within this thesis resides with the copyright holder(s) of those works.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship

This page is intentionally left blank

List of Figures

List of Tables

Publications

1. Qiu, M, Alexander, B, Simpson, AR & Elhay, S 2018, 'WDSLib: A Water Distribution System Simulation Test Bed', Manuscript submitted for publication to *Environmental Modelling & Software* (submitted Apr 2018).
2. Qiu, M, Simpson, AR, Elhay, S & Alexander, B 2017, 'A Benchmarking Study of Water Distribution System Solution Methods', Manuscript submitted for publication to *Journal of Water Resources Planning and Management* (submitted Nov 2017).
3. Qiu, M, Elhay, S, Simpson, AR & Alexander, B 2018, 'A Bridge-Block Partitioning Algorithm for Speeding up Analysis of Water Distribution Systems', Manuscript submitted for publication to *Journal of Water Resources Planning and Management* (submitted May 2018).

This page is intentionally left blank

Acknowledgments

I would take this opportunity to express my deepest gratitude to my excellent and patient supervisors Prof. Angus R. Simpson, Dr. Sylvan Elhay, and Dr. Bradley Alexander. The guidance and wisdom that they have given to me will always be a most valuable asset for the rest of my life. They were always available to provide constructive suggestions and discuss any issues.

I would also like to acknowledge the support of my family. To my wife, Lan Hu, for her unconditional love, encouragement, and support throughout my PhD studies. To my parents who always ask me "When are you going to finish your PhD?" To my one-year-old baby, Aimee, for her presence in my life when I needed her the most. My family has given me the push that I needed to cross the finish line.

This page is intentionally left blank

Chapter 1

Introduction and Publications Overview

1.1 Introduction

Water distribution systems (WDSs) are essential components of every city and town to satisfy the water consumption requirements of the population, of agriculture and for industry. The expansion of a WDS system will be necessary when the size of a city expands. As a result, the cost, especially the capital cost, may be high.

A number of different techniques in terms of minimising the cost of the designed water distribution systems have been studied previously, including traditional methods, such as linear programming and non-linear programming (Alperovits and Shamir 1977; Quindry et al. 1981), and evolutionary algorithms (EAs), such as genetic algorithms (Murphy et al. 1993; Simpson et al. 1994; Dandy et al. 1996; Savic and Walters 1997), simulated annealing (Loganathan et al. 1995; Cunha and Sousa 1999), tabu search (Lippai et al. 1999). harmony search (Geem et al. 2002; Geem 2006), the shuffled frog leaping algorithm (Eusuff and Lansey 2003), particle swarm optimisation (Suribabu and Neelakantan 2006; Montalvo et al. 2008), ant colony optimisation (Maier et al. 2003; Zecchin et al. 2006, 2007; Ostfeld and Tubaltzev 2008), memetic algorithm (Baños et al. 2007) and differential evolution (Suribabu 2010; Vasan and Simonovic 2010; Zheng et al. 2012). The main advantage of using EAs over the traditional methods is that EAs are able to deal with the nonlinear, higher dimensional and discrete nature of the WDS design problems. It has been demonstrated by the existing research that it is promising to use of EAs to find an optimal or near optimal WDS design. One drawback of using EAs is that they can be time-consuming. This is because designing a WDS using an EA always requires a large number of evaluations (usually hundreds of thousands). A component that consumes a substantial amount of time in an EA run for the optimisation of a WDS is the hydraulic simulation .

The steady-state demand-driven hydraulic simulation of a water distribution system has been a research topic since 1936 with the first manual method, the loop flow corrections method (Cross 1936), approached the problem by using successive approximations. Since then, many attempts have been made to improve the accuracy and the efficiency of the hydraulic simulation component. These attempts to improve the WDS hydraulic simulation model have became increasingly important when a large number of network simulations is required especially when considering the use of (1) evolutionary algorithms to perform single-objective and multi-objective optimisation of WDS network designs, and (2) real-time network monitoring and calibration under the supervisory control and data acquisition (SCADA) operational setting.

In a hydraulic simulation, there are two sets of primary equations that govern the underlying relationships of a WDS under steady-state conditions: a set of mass conservation or continuity equations and a set of energy conservation equations. Some assumptions are made to simplify the governing equations of a hydraulic simulation including: (1) that the velocity heads are negligible when compared to the friction head losses, (2) that the minor head losses at the pipe junctions and fittings are much smaller than the friction head losses, (3) water is incompressible, (4) the demands are considered to occur at a particular time instance and are concentrated at the nodes of a network, and (5) the demands are independent of nodal pressure. With the above assumptions, the two governing equations mentioned above can be described as: (1) the mass conservation equations: the total inflow must equal to the total outflow at any node; (2) the energy conservation equations: the head difference must be equal to the friction head loss for any pipe.

These two sets of governing equations can be formulated as a large and sparse non-linear saddle point problem (Benzi et al. 2005). There is a number of well-known iteration methods for solving this non-linear saddle point problem. These include: range space methods (Todini and Pilati 1988), null space methods (Rahal 1995; Elhay et al. 2014), and loop-based methods (Epp and Fowler 1970; Nielsen 1989). Moreover, the use of graph theory has became increasingly popular in developing solution methods to improve both the efficiency and the reliability of WDS solution process. The main reason underpinning the philosophy of using graph theory with hydraulic simulation is the invariant nature of the network topology. This fixed topology can often be exploited as a pre-and-post-processing step to speed-up the computations.

Range Space Methods: The global gradient algorithm (GGA) (Todini and Pilati 1988), a range space method, employed block elimination to reduce the size of the key matrix. Although graph theory is not used when deriving GGA solution method, the node-arc incidence matrix, which was first used in Todini and Pilati (1988) to describe the network topology, provides a portal into using graph theory to simplify the solution process of a WDS network. Simpson et al. (2012) developed the concept of separating the forest and core components while Deuerlein (2008) introduced the forest-core partitioning algorithm (FCPA). The forest component is separated out from the core by sweeping the node-arc incidence matrix. After the forest component is separated out, a standard GGA is then applied to the core component of the network. The main advantage of the FCPA is to separate the forest, which is linear component of the system of equations, from the core, which is the nonlinear component of the system of equations. This process speeds up the demand-dependent model (DDM) solution process when a network has a significant forest portion. Later, the graph matrix partitioning algorithm (GMPA) (Deuerlein et al. 2015) was proposed. The GMPA exploited the linear relationships between flows of the internal trees within the core and the flows of the corresponding super-links after the forest of the network had been removed.

Loop-Based Methods: The Hardy Cross method (Cross 1936), a loop based method, is the oldest method. In the Hardy Cross method, the system of equations is solved by successive approximation, in which a set of flows that satisfies continuity is successively corrected loop by loop until the predefined stopping test has been met. In another paper, Epp and Fowler (1970) developed a programmable version of the Hardy Cross method. However, the loop-based method is not widely used because (1) it required the identification of the loops, (2) it required the use of a pseudo-source if the network has more than one source, and (3) it required the determination a set of initial flows that satisfies continuity. Deuerlein (2008) proposed a decomposition model for WDS graph, in which the network

is first partitioned into forest component and core component. After separating out the forest component, the core component can be further partitioned into blocks and bridges. The remaining nonlinear component is then solved by a loop-based method. Later, Creaco and Franchini (2013) used the concept of the minimum cycle basis to achieve the sparsest key matrix for loop formulation. The main disadvantage of their method is that the process to identify the minimum cycle basis is time consuming. More recently, Alvarruiz et al. (2015) presented two methods to identify the minimum cycle basis that are significantly less time-consuming.

Null Space Methods: The null space method uses a spanning tree and a co-tree combination to reduce the effort in identifying loops. The null space method, in the context of hydraulic simulation, is a special case of the loop-based method. For example, in most cases, the minimum cycle basis achieved in Creaco and Franchini (2013) and Alvarruiz et al. (2015) cannot be achieved in the null space method. The co-tree flows method (CTM) (Rahal 1995) is the first null space method, which partitions the network into a spanning tree and a co-tree. The CTM has the following disadvantages: (1) it uses a pseudo-source if the network has more than one source, (2) it requires that a set of initial flows be found that satisfies continuity. Later, the reformulated co-tree flows method (RCTM) was introduced by Elhay et al. (2014) to address the start-up requirements of the loop-based method and the CTM by incorporating the Schilders' Factorisation (Schilders 2009). In another paper, Abraham and Stoianov (2015) proposed a partial update method for null space method. Savings in computation time, compared with the RCTM, are achieved by reducing the calculations of the head loss components and matrix multiplications by only calculating them when the stopping test for the corresponding pipe flows has not been met.

Despite the intensive research that has been undertaken in the field, these methods are not widely adopted in the industry because: (1) the relative performance of different graph theory based algorithm depends on the topology of the target network and it is difficult to evaluate the impact of these topology factors by only examining the incidence matrix that describes the pipe network connectivity and (2) a simulation platform is not available to efficiently implement these algorithms so that a user is not able to easily benchmark the performance of different solution methods on the network of interest.

1.2 Research Aims

This research has been carried out in order to address the aforementioned limitations and to achieve a broader acceptance of the application of graph theory in the field of hydraulics, the aims of which can be summarised as follows:

Aim #1: To develop an extensible, robust, and efficient testbed for WDS solution methods EPANET2 (Rossman 2000) is one of the most widely used WDS simulation packages. EPANET2 implemented the global gradient algorithm (Todini and Pilati 1988) to provide a steady-state demand-driven solution of a WDS. However, it has been pointed out by Guidolin et al. (2010) that the EPANET2 implementation is not explicitly designed to be easily understood or to easily accommodate new solution methods. As a result, the researchers who have focused on the hydraulic solution methods have used different platforms (for example Matlab and C++) to compare the performance of methods. Cross-platform comparisons favour compiled languages, for example C++, over interpreted languages, for example Matlab. This thesis advances the field by developing a simulation platform, called WDSLib, for the testing and the evaluation of existing and new WDS

solution methods. Moreover, a number of graph theory based WDS solution methods have been efficiently implemented to provide a fast simulation platform for both once-off and multi-run simulation settings.

Aim #2: To provide insight in the choice of solution methods for given combinations of network features and given design settings It is often difficult, if not impossible, to determine a priori what method or combination of methods to use for a given network topology. The simulation platform developed in **Aim #1** is used to benchmark the hydraulic solution of a number of case study water distribution networks with a variety of topology features. The correlations between these topology features and the relative performance of the methods of interest are studied.

Aim #3: To develop a new graph theory based algorithm to further partition the WDS A new algorithm that can be used to further partition the network is proposed. This algorithm is implemented in the simulation platform developed in **Aim #1** and a detailed case study is carried out exploring the algorithm's efficiency and its reliability.

1.3 Publications

This thesis is comprised of three publications. Their contribution to the body of knowledge is aligned with the research aims in Section 1.2. This section gives a brief description for each publication and its contribution.

Chapter 3 presents the development of an extensible simulation platform, WDSLib, for the demand-driven steady-state analysis of a WDS. WDSLib has been created using a modularised object-oriented design and implemented in the C++ programming language, and has been validated against a reference MATLAB implementation. Two solution methods, namely the global gradient algorithm (GGA) and the reformulated co-tree flows method (RCTM), and a pre-processing and post-processing method, the forest-core partitioning algorithm (FCPA), are currently implemented in WDSLib.

Chapter 4 presents a thorough benchmark study to compare the performance of GGA, GGA with FCPA, RCTM, and RCTM with FCPA using WDSLib developed in the first publication. The results of this study will help inform the choice of solution methods for given combinations of network features and given design settings.

Chapter 5 proposes a bridge-block partitioning algorithm (BBPA) that further partitions the network into bridges, blocks and cut-vertices. It has been shown that the use of the BBPA is not only able to significantly reduce the computation time of the once-off simulation and the multi-run simulation, but also able to improve the reliability of the solution.

1.3.1 Contributions to the development of a WDS Simulation Platform for WDS Simulation and Optimisation

A number of contributions have been made in developing a framework for efficiently incorporating graph theory in a WDS simulation model that can be used for simulation, optimisation, and management of a WDS network. These contributions are presented while describing the workflows involved in different graph theory based WDS solution methods.

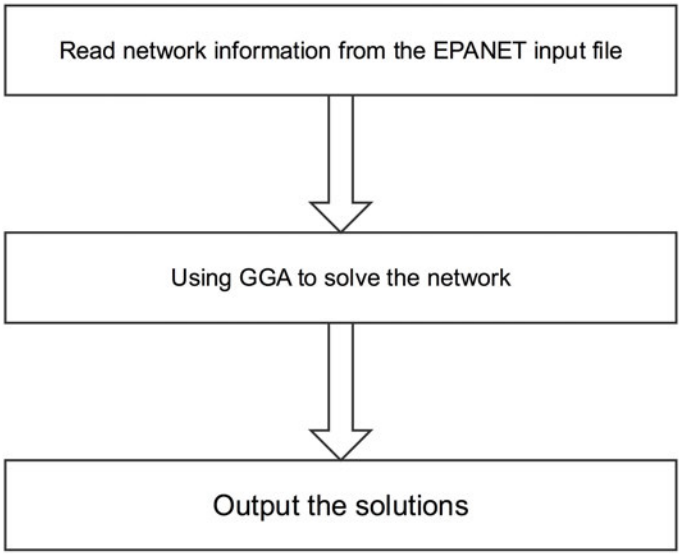


Fig. 1.1. Traditional water distribution system simulation toolkit structures

Fig. 1.1 shows the high level workflow that is used in existing WDS simulation platforms. It is obvious that this workflow in traditional WDS simulation toolkits is straightforward: first, the network information is parsed from an EPANET input file (for example, `ENopen`), second, the network is solved by using the GGA (for example, `ENsolveH`), and finally the solution of the network is outputted.

In contrast, WDSLib, that was developed as part of this research, has incorporated two WDS solution methods, namely the global gradient algorithm and the reformulated co-tree flows methods, and two pre-processing and post-processing methods, namely forest-core partitioning algorithm and bridge-block partitioning algorithm. Each of the two solution methods can be used with either, both, or neither of the two pre-processing and post-precessing methods. The high level flowchart for each combination is shown in Fig. 1.2. The functions used for different combinations of solution methods are categorised into five different level of repetitions: once before every multiple simulation (L_{1a}), once before every iterative phase (L_{2a}), once every iterative phase (L_3), once after every iterative phase (L_{2b}), and once after every multiple simulation (L_{1b}). The level of each function is determined by the number of times it should be run. This categorisation of the WDS solution methods is discussed in more detail in Chapter 3. Publication 1 describes the WDSLib, a WDS simulation test bed. WDSLib allows users to (i) choose from, or modify, different approaches and implementations of different WDS model analyses, and (ii) extend the toolkit to include new developments. WDSLib uses a pluggable architecture where solution methods, and their accompanying pre-processing and post-processing code are easily substituted, studied, and benchmarked. WDSLib is later used as a benchmarking tool in Publications 2 and 3.

1.3.2 Contributions to WDS Solution Methods

Publication 2 presents a benchmarking study on two existing WDS solution methods, the GGA and the RCTM, with or without the FCPA (a pre- and post-processing method). In this research, the aim is to address the problem of which solution method or methods to apply. Previous publications have performed case studies comparing the performance of their respective methods against that of the GGA. However, these comparisons have often been

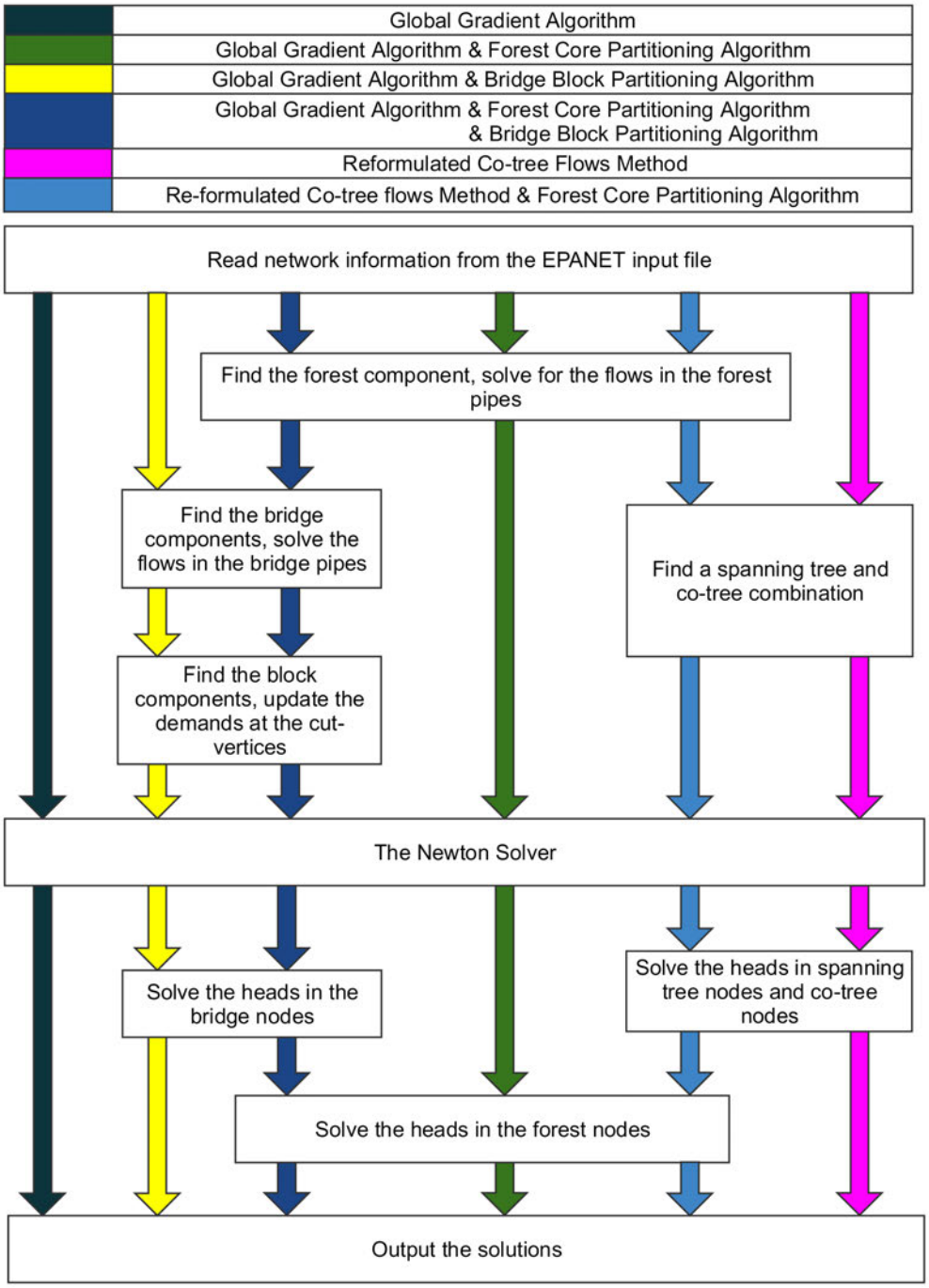


Fig. 1.2. New WDSLib water distribution system simulation toolkit structure

done using different implementation languages, and different levels of code optimisation – which makes a fair cross-comparison of methods difficult. This research here presents a thorough benchmark study to compare the performance of GGA, GGA-with-FCPA, RCTM, and RCTM-with-FCPA for a range of case study networks using a fast C++ implementation.

Publication 3 proposes a new graph theory algorithm, the bridge-block partitioning algorithm (BBPA), to further partition a WDS network into bridges and blocks. A bridge element can be solved using a linear solver, similar to a forest element in the FCPA and each block can be solved separately as a smaller non-linear system. The BBPA is a pre-and-post-processing method that is able to (1) reduce the computation time for a

once-off simulation setting, a multi-run simulation setting and an operational simulation setting; (2) improve the numerical reliability of the solution; and (3) allow the solution of each block to be found in parallel.

This thesis is structured as follows. Chapter 2 gives a detailed review of existing solution methods that can be used to solve the steady-state demand-driven flows and heads in a WDS. Chapter 3 presents the WDSLib software package. A benchmark study on four WDS solution methods is carried out in Chapter 4. Chapter 5 describes the bridge-block partitioning algorithm. Finally, Chapter 6 offers some conclusions and recommendations for future study.

This page is intentionally left blank

Chapter 2

Review of the Existing Water Distribution System Solution Methods

This chapter reviews the fundamental aspects of the hydraulic analysis of a steady-state demand-driven water distribution system. The system of equations for a WDS is first described in Section 2.1. Section 2.2 describes some of the recent applications of graph theory concepts in the solution of the steady-state problem for a water distribution system. Then, in Section 2.3, the solution methods that are used to simulate the steady-state of a WDS are reviewed.

2.1 WDS Model equations

This thesis considers a demand-driven water distribution system with n_p pipes, n_j unknown-head nodes and n_f fixed-head nodes. The j-th pipe of the network can be characterised by its diameter d_j, length l_j, resistance factor r_j. The i-th node of the network can be characterised by its nodal demand d_i, and the elevation head z_i.

Let $\boldsymbol{q} = (q_1, q_2, ...q_{n_p})^T$ denote the vector of unknown flows, $\boldsymbol{h} = (h_1, h_2, ...h_{n_j})^T$ denote the vector of unknown heads, $\boldsymbol{r} = (r_1, r_2, ...r_{n_p})^T$ denote the vector of pipe resistance factors, $\boldsymbol{d} = (d_1, d_2, ...d_{n_j})^T$ denote the vector of nodal demands, $\boldsymbol{e_l} = (e_{l_1}, e_{l_2}....e_{l_{nr}})^T$ denote the vector of fixed head elevations.

The head loss exponent n is assumed to be dependent only on the head loss model: $n = 2$ for the Darcy-Weisbach head loss model and $n = 1.852$ for Hazen-Williams head loss model. The head loss within the pipe j, which connects the node i and the node k, is modelled by $h_i - h_k = r_j q_j |q_j|^{n-1}$. Denote by $\boldsymbol{G}(\boldsymbol{q}) \in \mathbb{R}^{n_p \times n_p}$, a diagonal square matrix with elements $[\boldsymbol{G}]_{jj} = r_j |q_j|^{n-1}$ for $j = 1, 2, ...n_p$. Denote by $\boldsymbol{F}(\boldsymbol{q}) \in \mathbb{R}^{n_p \times n_p}$, a diagonal square matrix where the j-th element on its diagonal $[\boldsymbol{F}]_{jj} = \frac{\partial}{\partial q_j}[\boldsymbol{G}]_{jj} q_j$. The unknown-head node-arc incidence matrix $\boldsymbol{A_1}$ is full rank, where $[\boldsymbol{A_1}]_{ij}$ is used to represent the relationship between pipe i and node j: $[\boldsymbol{A_1}]_{ij} = -1$ if pipe i enters node j, $[\boldsymbol{A_1}]_{ij} = 1$ if pipe i leaves node j, and $[\boldsymbol{A_1}]_{ij} = 0$ if pipe i is not connected to node j. The matrix $\boldsymbol{A_2}$ is the fixed-head node-arc incidence matrix, where $[\boldsymbol{A_2}]_{ij}$ is used to represent the relationship between pipe i and fixed head node j: $[\boldsymbol{A_2}]_{ij} = -1$ if pipe i enters fixed head node j, $[\boldsymbol{A_2}]_{ij} = 1$ if pipe i leaves fixed head node j, and $[\boldsymbol{A_2}]_{ij} = 0$ if pipe i is not connected to fixed head node j. The steady-state flows and heads in the WDS system

are modeled by the demand-driven model (DDM) continuity equations Eq. (2.1) and the energy conservation equations Eq. (2.2):

$$-\boldsymbol{A_1}^T \boldsymbol{q} - \boldsymbol{d} = \boldsymbol{O} \tag{2.1}$$

$$\boldsymbol{G}(\boldsymbol{q})\boldsymbol{q} - \boldsymbol{A_1}\boldsymbol{h} - \boldsymbol{A_2}\boldsymbol{e}_l = \boldsymbol{O}, \tag{2.2}$$

which can be expressed as

$$\begin{pmatrix} \boldsymbol{G}(\boldsymbol{q}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{h} \end{pmatrix} - \begin{pmatrix} \boldsymbol{A_2}\boldsymbol{e}_l \\ \boldsymbol{d} \end{pmatrix} = \boldsymbol{O}, \tag{2.3}$$

where its Jacobian matrix is

$$\boldsymbol{J} = \begin{pmatrix} \boldsymbol{F}(\boldsymbol{q}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix}. \tag{2.4}$$

and it is sometimes referred to as a nonlinear saddle point problem (Benzi et al. 2005).

This non-linear system is normally solved by the Newton method, in which $\boldsymbol{q}^{(m+1)}$ and $\boldsymbol{h}^{(m+1)}$ are repeatedly computed from $\boldsymbol{q}^{(m)}$ and $\boldsymbol{h}^{(m)}$ by

$$\begin{pmatrix} \boldsymbol{F}^{(m)}(\boldsymbol{q}^m) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}^{(m+1)} - \boldsymbol{q}^{(m)} \\ \boldsymbol{h}^{(m+1)} - \boldsymbol{h}^{(m)} \end{pmatrix} = - \begin{pmatrix} \boldsymbol{G}^{(m)}\boldsymbol{q}^{(m)} - \boldsymbol{A_1}\boldsymbol{h}^{(m)} - \boldsymbol{A_2}\boldsymbol{e}_l \\ -\boldsymbol{A_1}^T\boldsymbol{q}^{(m)} - \boldsymbol{d}, \end{pmatrix} \tag{2.5}$$

until the relative differences $\frac{||\boldsymbol{q}^{(m+1)}-\boldsymbol{q}^{(m)}||_\infty}{||\boldsymbol{q}^{(m+1)}||_\infty}$ and $\frac{||\boldsymbol{h}^{(m+1)}-\boldsymbol{h}^{(m)}||_\infty}{||\boldsymbol{h}^{(m+1)}||_\infty}$ are sufficiently small.

The most widely used WDS simulation method in current use is the Global Gradient Algorithm (GGA) (Todini and Pilati 1988), which solves the non-linear system of equations, Eq. (2.5), representing the WDS. The GGA and its implementations exhibit excellent convergence characteristics for a wide range of starting values and a wide variety of WDS problems. However, some networks have structural properties which can be exploited to further improve the efficiency of the solution process. In the next section, some of the graph theory concepts that are used in WDS analysis are described.

2.2 Graph Theory Concepts

Associated with a WDS is a graph G=(V, E), where the elements of V are the nodes (vertices) of the graph G and elements of E are the pipes (links) of the graph G. The first description of the WDS graph was introduced by Todini and Pilati (1988), in which the network graph is described by a directed node-arc incidence matrix. This directed node-arc incidence matrix is divided into an unknown-head node-arc incidence matrix, $\boldsymbol{A_1}$, and a fixed-head node-arc incidence matrix, $\boldsymbol{A_2}$.

Forest A tree is a graph in which any two vertices are connected by exactly one edge. Most WDSs have trees, the collections of which are called forests. By dividing a WDS graph into its linear forest component, $G_f = \left(V_f, E_f\right)$, and non-linear core component, $G_c = (V_c, E_c)$, the flows in the forest pipes can be computed a priori and the heads in the forest nodes can be computed a posteriori by a linear process. Hence, the dimension of the key matrices in the solution process can be significantly reduced when the forest is a large part of the network.

Spanning Tree A spanning tree is an acyclic subgraph which traverses every node in a graph, such that the addition of any co-tree element creates a loop. An acyclic graph is a graph having no graph cycles. A WDS, with or without a forest, can be partitioned into two subgraphs: a spanning tree component, $G_{st} = (V_{st}, E_{st})$, and a set of co-tree edges, E_{ct}, so that $E_{st} \cup E_{ct} = E_c$, $E_{st} \cap E_{ct} = \emptyset$. This relationship can sometimes be used to further exploit the block structure of the Jacobian matrix to produce, in realistic WDSs, an even smaller key matrix. This is achieved by dealing separately with the spanning tree and the co-tree in the Newton method linearisation.

Loop A loop, know as a *simple cycle* in graph theory, is a path of edges and vertices wherein a vertex is reachable from itself with no repetitions of vertices and edges. Two loops, C_1 and C_2, can be used to form another loop by using the *symmetric difference* of two sets $((C_1 \cup C_2) - (C_1 \cap C_2))$. The set of all loops is called the *cycle space*. Consider a connected graph G=(V,E) with a spanning tree $G_{st} \in G$ and the complementary co-tree edges E_{ct} . For every co-tree edge $e \in E_{ct}$ there is a unique cycle C_e in G_{st} + e; these cycles C_e are the *fundamental cycles* of G with respect to the spanning tree G_{st}.

If T is a spanning tree or spanning forest of a given graph G, and e is an edge that does not belong to T, then the fundamental cycle C_e defined by e is the simple cycle consisting of e together with the path in T connecting the endpoints of e. There are exactly $n_p - n_j + c$ fundamental cycles, one for each edge that does not belong to T . Each of them is linearly independent from the remaining cycles, because they include an edge e that is not present in any other fundamental cycle. Therefore, the fundamental cycles form a *cycle basis* for the cycle space. A cycle basis of a graph is a minimal set of simple cycles that allows every cycle in the cycle space to be expressed as a symmetric difference of basis cycles.

Minimum cycle basis The cycles that can be made by a spanning tree and the corresponding co-tree is a subset of the cycle space. In cycle-based methods, it is often preferable to use a shortest cycle basis. The Shortest Maximal Cycle Basis (SMCB) is a cycle basis B of a given graph G with the property that the length of the longest cycle included in B is the smallest among all bases of G. It is possible to minimise the number of non-zeros in the key matrix of loop-based methods by using a shortest cycle basis.

2.3 Solution Methods

We consider three types of hydraulic solution methods: (1) range space methods, (2) loop-based methods and (3) null space methods. These three types of solution methods and the applications of graph theory in each of the three categories are discussed in the following sections.

2.3.1 Range Space Methods

The global gradient algorithm (GGA), a range space method, was first proposed by Todini and Pilati (1988). They applied block elimination to Eq. (2.5) to yield a two-step Newton solver for the cases when the head loss is modelled by the Hazen-William formula:

$$\boldsymbol{h}^{(m+1)} = \boldsymbol{U}^{-1}\left\{-n\boldsymbol{d} + \boldsymbol{A_1}^T[(1-n)\boldsymbol{q}^{(k)} - \boldsymbol{G}^{-1}\boldsymbol{A_2}\boldsymbol{e_l}]\right\} \tag{2.6}$$

where the Schur complement is defined as $\boldsymbol{U} = \boldsymbol{A_1}^T \boldsymbol{G}^{-1} \boldsymbol{A_1}$

$$\boldsymbol{q}^{(m+1)} = \frac{1}{n}\left\{(n-1)\boldsymbol{q}^{(k)} + \boldsymbol{G}^{-1}(\boldsymbol{A_2 e_l} + \boldsymbol{A_1 h})\right\} \tag{2.7}$$

The GGA has become the most widely used network formulation method in hydraulic software packages, such as EPANET2 (Rossman 2000). This is mainly because of the outstanding convergence characteristics that have been exhibited by the GGA. Even so, a number of issues have been reported in the literature.

Simpson and Elhay (2010) pointed out that the original GGA was designed only for the use of Hazen-William head loss model, in which the Hazen-William coefficient is assumed to be independent of flow. The Darcy-Weisbach friction factor, unlike the Hazen-William coefficient C, is dependent on the pipe flow. However, Rossman (2000) incorrectly treated the Darcy-Weisbach resistance factors in computing the Jacobian matrix. Simpson and Elhay (2010) gave the correct formulae to compute the Jacobian for the Darcy-Weisbach head-loss model. They derived the terms of the matrix $\boldsymbol{F}$ for three flow regimes based on the ranges of Reynolds number. A special case is the transitional flow regime, where Dunlop's interpolating cubic spline (Dunlop 1991) is used to ensure a smooth transition of the friction factors from the laminar to the turbulent flow (Table (2.2)). Note that Dunlop's interpolating cubic spline used in their paper is identical to that in EPANET2. The resulting two-step Newton solver when the head loss is modelled by either the Darcy-Weisbach or Hazen-William formula is:

$$\boldsymbol{V h}^{(m+1)} = -\boldsymbol{d} + \boldsymbol{A_1}^T \boldsymbol{F}^{-1}\left[(\boldsymbol{G}-\boldsymbol{F})\,\boldsymbol{q}^{(m)} - \boldsymbol{A_2 e_l}\right] \tag{2.8}$$

where the Schur complement is defined as $\boldsymbol{V} = \boldsymbol{A_1}^T \boldsymbol{F}^{-1} \boldsymbol{A_1}$ and $\boldsymbol{F}$ is the correct computed Jacobian matrix when the head loss is modelled by either the Darcy-Weisbach or Hazen-William formula,

$$\boldsymbol{q}^{(m+1)} = \boldsymbol{q}^{(m)} + \boldsymbol{F}^{-1}\boldsymbol{A_1 h}^{(m+1)} - \boldsymbol{F}^{-1}\left[\boldsymbol{G q}^{(m)} - \boldsymbol{A_2 e_l}\right]. \tag{2.9}$$

The use of the correct Jacobian matrix restores the quadratic convergence associated with the Newton method. Furthermore, when the correct corresponding Jacobian is used with the Darcy Weisbach head-loss model, the Jacobian matrix is no longer singular as a result of the presence of zero flows due to the use of the correct expression for friction factor as shown in the first row of Table 2.1.

In the following, denote

$$\mathcal{R} = \frac{4|q|}{\pi\nu D} = \frac{vD}{\nu}, \quad a = \left(\frac{2\ln^2 10}{\pi^2 g}\right), \quad b = 1/3.7, \quad c = 5.74\,(\pi\nu/4)^{9/10}, \quad \eta = \mathcal{R}/2000$$

$$\rho = \left|\frac{D}{q}\right|^{9/10}, \theta = \frac{\epsilon}{3.7D} + \frac{5.74}{\mathcal{R}^{9/10}} = \frac{b\epsilon}{D} + c\rho, \quad \widehat{\theta} = \frac{b\epsilon}{D} + \frac{5.74}{4000^{9/10}}, \quad \sigma = \frac{|q|}{q}, q \neq 0$$

$$\omega = 2 + \frac{9c\rho}{5\theta\ln(\theta)}.$$

Elhay and Simpson (2011) proposed a regularisation method to deal with the zero flows when the pipe head losses are modelled by the Hazen-William head-loss formula. This method sets an upper bound on the condition number of the Schur Complement. This regularisation starts by adding a modification vector,

$$\begin{pmatrix} \boldsymbol{T} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{O} \end{pmatrix}\begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{h} \end{pmatrix},$$

Table 2.1. The resistance factor formulae

Range of $\mathcal{R}$	Resistance factor r
$\mathcal{R} \leq 2000$ Laminar flow*	$\frac{128\nu}{\pi g}\frac{L}{D^4}$
$2000 < \mathcal{R} < 4000$ Transitional flow	$\left(\frac{8}{\pi^2 g}\right)\frac{L}{D^5}\sum_{k=0}^{3}\left(\alpha_k + \beta_k/\theta\right)\eta^k$
$\mathcal{R} \geq 4000$ Turbulent flow	$\frac{L}{D^5}\frac{a}{\ln^2\theta}$

*Note that this resistance factor term for laminar flow is independent of q

Table 2.2. Coefficients of the cubic interpolating spline defining the Darcy-Weisbach friction factor for $2000 < \mathcal{R} < 4000$. The constants are $\tau = 0.00514215$ and $\xi = -0.86859$.

k	α_k	β_k
0	$5/(\xi^2 ln^2\widehat{\tau})$	$\tau/(\xi^3 ln^3\widehat{\tau})$
1	$0.128 - 12/(\xi^2 ln^2\widehat{\tau})$	$-5\tau/(2\xi^3 ln^3\widehat{\tau})$
2	$-0.128 + 9/(\xi^2 ln^2\widehat{\tau})$	$2\tau/(\xi^3 ln^3\widehat{\tau})$
3	$0.032 - 2/(\xi^2 ln^2\widehat{\tau})$	$-\tau/(2\xi^3 ln^3\widehat{\tau})$

Table 2.3. The elements of the Darcy-Weisbach head loss model vector $\boldsymbol{\phi}(\boldsymbol{r}(\boldsymbol{q}), \boldsymbol{q})$.

Range of $\mathcal{R}$	The diagonal terms in $\boldsymbol{G}$
$\mathcal{R} \leq 2000$	$q\left(\frac{128\nu}{\pi g}\right)\frac{L}{D^4}$
$2000 < \mathcal{R} < 4000$	$q\|q\|\left(\frac{8}{\pi^2 g}\right)\frac{L}{D^5}\sum_{k=0}^{3}\left(\alpha_k + \beta_k/\theta\right)\eta^k$
$\mathcal{R} \geq 4000$	$q\|q\|\frac{L}{D^5}\frac{a}{\ln^2\theta}$

Table 2.4. The diagonal terms of the matrix $\boldsymbol{F}$, the Jacobian of Darcy-Weisbach head loss model $\boldsymbol{\phi}(\boldsymbol{r}(\boldsymbol{q}), \boldsymbol{q})$.

Range of $\mathcal{R}$	The diagonal terms in $\boldsymbol{F}$
$\mathcal{R} \leq 2000$	$\left(\frac{128\nu}{\pi g}\right)\frac{L}{D^4}$
$2000 < \mathcal{R} < 4000$	$\left(\frac{8}{\pi^2 g}\right)\frac{L}{D^5}\|q\|\sum_{k=0}^{3}\left\{\frac{9c}{10}\frac{\beta_k}{\mathrm{p}^2}\left\|\frac{D}{q}\right\|^{9/10} + (2+k)\left(\alpha_k + \beta_k/\theta\right)\right\}\eta^k$
$\mathcal{R} \geq 4000$	$\frac{L}{D^5}\frac{a\|q\|}{\ln^2\theta}\left(2 + \frac{9c\rho}{5\theta\ln\theta}\right)$

to both sides of Eq. (2.3) to ensure that the Schur complement is invertible. The modified two-stage GGA equations are:

$$\boldsymbol{W}\boldsymbol{h}^{(m+1)} = -\boldsymbol{d} + \boldsymbol{A_1}^T(\boldsymbol{F}+\boldsymbol{T})^{-1}\left[(\boldsymbol{G}-\boldsymbol{F}-\boldsymbol{T})\,\boldsymbol{q}^{(m)} - \boldsymbol{A_2}\boldsymbol{e_l}\right] \tag{2.10}$$

where $\boldsymbol{W} = \boldsymbol{A_1}^T(\boldsymbol{F}+\boldsymbol{T})^{-1}\boldsymbol{A_1}$

$$\boldsymbol{q}^{(m+1)} = \boldsymbol{q}^{(m)} + (\boldsymbol{F}+\boldsymbol{T})^{-1}\boldsymbol{A_1}\boldsymbol{h}^{(m+1)} - (\boldsymbol{F}+\boldsymbol{T})^{-1}\left[\boldsymbol{G}\boldsymbol{q}^{(m)} - \boldsymbol{A_2}\boldsymbol{e_l}\right] \tag{2.11}$$

Forest and Core

Simpson et al. (2012) proposed the forest-core partitioning algorithm (FCPA) to reduce the WDS simulation runtime of the GGA. The main contribution of this paper was the proposed detailed computer algorithm to implement the decomposition model as earlier suggested by Deuerlein (2008).

The FCPA can be described by the following steps:

1. Identify the forest component as distinct from the core component, and at the same time find the flows of the forest pipes and adjust the demands.
   a) Create four lists, p, a list of pipes, v, a list of nodes, s, a list of pipes, and t, a list of nodes. Initialise p with all pipe indices within the network, v with all node indices within the network, and each of s and t with an empty list. When the forest identification has been completed, p will contain the indices of the pipes in the core, v will contain the indices of the nodes in the core, s will contain the indices of the pipes in the forest, and t will contain the indices of the nodes in the forest.
   b) Identify the columns of the submatrix $\boldsymbol{A_1}(p,v)$ (also denoted by $\boldsymbol{A_{1_{p,v}}}$) – which can be interpreted as a sub-matrix of $\boldsymbol{A_1}$ that is composed of the rows of $\boldsymbol{A_1}$ indicated by list p and the columns of $\boldsymbol{A_1}$ indicated by list v – that has only one non-zero element (this represents a leaf node in a tree). Record the column number and the corresponding row number of each non-zero element. For instance, the i-th column has only one non-zero element and it sits at the j-th row. This means $\boldsymbol{A_1}(p(j),v(i))$ is the only nonzero element in column i.
   c) Find, if it exists, the column m of row j of the submatrix $\boldsymbol{A_1}(p,v)$, which contains the other non-zero element, and its value to $\alpha = \pm 1$.
   d) If m is defined, replace the m-th element within the demand vector with $d_m = d_m + d_i$ and set the flow in pipe j to $q_j = -\alpha d_i$ and insert the value into the $\boldsymbol{q}$ vector.
   e) Move the index of pipe j from list p to list s and node i from list v to list t.
   f) Repeat the steps until all columns in $\boldsymbol{A_1}(p,v)$ have more than one non-zero element.
2. Solve for heads and flows of the core component of the network using the standard GGA (as in the above paper):

$$\begin{pmatrix} \boldsymbol{G_{p,p}} & -\boldsymbol{A_{1_{p,v}}} \\ -\boldsymbol{A_{1_{p,v}}}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q_p} \\ \boldsymbol{h_v} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A_{2_p}}\boldsymbol{e_l} \\ \boldsymbol{d_v} \end{pmatrix}. \tag{2.12}$$

3. Once the iterative solution process for the core has stopped, the forest heads can be found by solving the linear system:

$$\boldsymbol{A_{1_{s,t}}} \boldsymbol{h_t} = \boldsymbol{G_{s,s}} \boldsymbol{q_s} - \boldsymbol{A_{2_s}} \boldsymbol{e_l} - \boldsymbol{A_{1_{s,v}}} \boldsymbol{h_v}. \tag{2.13}$$

The FCPA simplifies the problem by identifying the linear forest part of the problem and solving it separately from the nonlinear core part to avoid unnecessary computation in the iterative process.

Internal Trees and Topological Minors

Deuerlein et al. (2015) presented another graph partitioning algorithm, called the graph matrix partitioning algorithm (GMPA), that has the potential to reduce the dimension of the non-linear part of the system of equations to be solved even further and hence reduce the simulation runtime. The detailed steps of the GMPA are:

1. The GMPA starts by partitioning the network into core and forest using the FCPA.
2. The GMPA partitions (i) the nodes in the core component of the graph into two lists: a list of supernodes s, nodes with degree greater than two (in other words three or more pipes are connected to the node), and a list of interior tree nodes i, nodes with degree two and (2) the pipes in the core component of the graph into two lists: a list of internal co-tree chords c, the chords of the internal tree and a list of internal tree branches t, pipes that connected to the interior path nodes.
3. The system of equations in Eq. (2.12) can be permuted into:

$$\begin{pmatrix} \boldsymbol{G_{t,t}} & & -\boldsymbol{A_{1_{s,t}}} & -\boldsymbol{A_{1_{i,t}}} \\ & \boldsymbol{G_{c,c}} & -\boldsymbol{A_{1_{s,c}}} & -\boldsymbol{A_{1_{i,c}}} \\ -\boldsymbol{A_{1_{s,t}}}^T & -\boldsymbol{A_{1_{s,c}}}^T & & \\ -\boldsymbol{A_{1_{i,t}}}^T & -\boldsymbol{A_{1_{i,c}}}^T & & \end{pmatrix} \begin{pmatrix} \boldsymbol{q_t} \\ \boldsymbol{q_c} \\ \boldsymbol{h_s} \\ \boldsymbol{h_i} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A_{2_t}} \boldsymbol{e_l} \\ \boldsymbol{A_{2_c}} \boldsymbol{e_l} \\ \boldsymbol{d_s} \\ \boldsymbol{d_i} \end{pmatrix} \tag{2.14}$$

4. The GGA with the GMPA, which is used to in Deuerlein et al. (2015), can now be divided into two parts: a global step,

$$\boldsymbol{A_p^T} \boldsymbol{G_s^{(k)}}^{-1} \boldsymbol{A_p} \boldsymbol{h_s^{(k+1)}} = \widehat{\boldsymbol{d}}_s + \boldsymbol{A_p^T} \boldsymbol{q_c^{(k)}} - \boldsymbol{A_p^T} \boldsymbol{G_s^{(k)}}^{-1} \left(\boldsymbol{h_s^{(k)}} + \boldsymbol{A_R} \boldsymbol{e_l} \right) \tag{2.15}$$

$$\boldsymbol{q_c^{(k+1)}} = \boldsymbol{q_c^{(k)}} - \boldsymbol{G_s^{(k)}}^{-1} \left(\boldsymbol{h_s^{(k)}} + \boldsymbol{A_p} \boldsymbol{h_s^{(k+1)}} + \boldsymbol{A_R} \boldsymbol{e_l} \right), \tag{2.16}$$

and a local step,

$$\boldsymbol{q_t^{(k+1)}} = \boldsymbol{P^T} \boldsymbol{q_c^{(k)}} - \boldsymbol{A_{1_{i,t}}} \boldsymbol{d_i} \tag{2.17}$$

$$\boldsymbol{h_i^{(k+1)}} = -\boldsymbol{A}_{1_{i,t}}^{-1} \left(\boldsymbol{A_{1_{s,t}}} \boldsymbol{h_s^{(k+1)}} + \boldsymbol{A_{2_t}} \boldsymbol{e_l} + \boldsymbol{G_t^{(k)}} \boldsymbol{q_t^{(k+1)}} \right), \tag{2.18}$$

where $\boldsymbol{P} = \boldsymbol{A_{1_{i,c}}} \boldsymbol{A_{1_{i,t}}}^{-1}$, $\boldsymbol{A_p} = \boldsymbol{A_{1_{s,c}}} + \boldsymbol{P} \boldsymbol{A_{1_{s,t}}}$, $\boldsymbol{A_R} = \boldsymbol{A_{2_c}} + \boldsymbol{P} \boldsymbol{A_{2_t}}$, $\widehat{\boldsymbol{d}}_s = \boldsymbol{A_{1_{s,t}}} \boldsymbol{A_{1_{i,t}}}^{-1} \boldsymbol{d_i} + \boldsymbol{d_s}$, and $\boldsymbol{G_s}^{(k)} = \boldsymbol{G_{c,c}^{(k)}} + \boldsymbol{P} \boldsymbol{G_{t,t}^{(k)}} \boldsymbol{P}^T$.

The GMPA exploits the linear relationships between flows of the internal trees and the flows of the corresponding super-links after the forest of the network has been removed. The GMPA can save significant computational time by reducing the dimension of the problem for the most time consuming part of the calculations.

2.3.2 Loop Based Methods

The Hardy Cross method (Cross 1936), also known as the loop flow corrections method (LFC), is the oldest solution method and uses successive approximations, solving for each loop flow correction independently. In this method, a set of initial flows satisfying continuity is successfully improved until the energy balance around the loop is achieved. In the Hardy Cross method, there are two sets of equations –(i) mass conservation equations and (ii) loop energy conservation equations– which are used to model the underlying relationship between the flows and heads of a WDS. The Hardy Cross method is a manual iterative method that was popular for its simplicity before computers.

Minimum cycle basis

Creaco and Franchini (2013) presented a new automatic procedure for the identification of a minimum cycle basis for a planar graph. It can be described as follows:

1. Find a spanning tree and co-tree permutation
2. For i=1, 2, ... number of loops, do

    a) Remove the i-th co-tree pipe
    b) Find the shortest path between the two end nodes of the i-th co-tree pipe using Dijkstra's algorithm (Dijkstra 1959)
    c) The minimum cycle is i-th co-tree pipe and the shortest path between the two end nodes of the i-th co-tree pipe

The advantage of this algorithm in the solution of WDS models is that using it minimises the number of of non-zeros in the Schur complement which allows for a faster iterative solution. However, one major drawback is its expensive overhead. Therefore, this algorithm is of limited use in practice.

Recently, Alvarruiz et al. (2015) proposed two algorithms to select a set of network loops in order to achieve a highly sparse matrix. Although, a smaller number of non-zeros in the Schur complement was reported in Creaco and Franchini (2013), the substantial improvement in terms of the efficiency reported by Alvarruiz et al. (2015) suggests the latter algorithm is the better practical choice.

2.3.3 Null Space Methods

The null space methods are special cases of loop-based method: all null space formulations can be rewritten as loop-based formulations, but not all loop-based formulations can be rewritten as null space formulations.

Co-Tree flows method

In 1995, Rahal (1995) proposed a co-tree flows formulation (CTM). The CTM algorithm can be described as:

1. Identify the main supply source, which is the one with the highest elevation head.
2. Transform the WDS network into its associated circulating graph by connecting all unknown-head nodes with the main source by pseudo-links.
3. Connect the main source with other sources by pseudo-links.

4. Identify the spanning tree and co-tree of the WDS.
5. Sum the demands that are to be carried by the tree branches, S_{TD}, assuming the co-tree will not carry any demand.
6. Determine the associated chain of branches, C_T, closing a circuit for each co-tree chord.
7. Specify a set of initial water flows in the co-tree chords, $Q_T^{(0)}$.
8. Compute the head difference between the main water source and the all other water sources Z.
9. For i=1,2,... n until the stopping criteria has been met:
    a) Assemble the Jacobian matrix of
    $$\boldsymbol{H(Q_T)} + \boldsymbol{C_T H(C_T^T Q_T} + \boldsymbol{S_{TD}) = O} \tag{2.19}$$
    b) Use the Newton method to compute the co-tree flow correction and hence the new co-tree chord flow rate.
10. Calculate the corresponding flow rates for the spanning tree
11. Determine the nodal heads using the tree structure

The CTM differs from the methods that have been discussed so far. Although it bears some resemblance to the loop flows correction formulation, it does not require initial pipe flows to satisfy continuity. The CTM is able to reduce the dimension of the equations of a water distribution system to the number of unknown co-tree flows (equal to the number of pipes minus the number of nodes). In order to perform CTM, it is necessary to: (1) identify the associated circulating graph; (2) determine the demands that are to be carried by tree branches; (3) find the associated chain of branches closing a circuit for each co-tree chord; and (4) compute pseudo link head losses Z.

Reformulated Co-tree flows method In a paper by Elhay et al. (2014), a reformulated co-tree flows method (RCTM) was proposed. It exploits the relationship between the co-tree flows and spanning tree flows. This is achieved by applying the Schilders' factorization to permute the $\boldsymbol{A_1}$ matrix into a lower triangular square block at the top, representing a spanning tree, and a rectangular block below, representing the corresponding co-tree. As a result, the number of unknowns that needs to be solved for in the nonlinear system is the number of the co-tree flow pipes, as in the CTM.

The RCTM can be summarised as follows:

1. The RCTM starts by partitioning the network into a spanning tree and a co-tree.
2. Permute:
    a) the unknown head index matrix ($\boldsymbol{A_1}$) into two blocks with an lower triangular block, $\boldsymbol{T_1}$, representing the spanning tree, above a rectangular block, $\boldsymbol{T_2}$, representing the co-tree;
    b) the vector of unknown flows $\boldsymbol{q}$ into $\boldsymbol{q_1}$, the vector of flows in the spanning tree pipes, and $\boldsymbol{q_2}$, the vector of flows in the co-tree pipes;

c) the product of the fixed-head node-arc incidence matrix and the vector of elevation heads of the fixed-head nodes, $\boldsymbol{A_2 e_l}$, into $\boldsymbol{a_1}$ for the spanning tree, and $\boldsymbol{a_2}$, for the co-tree;

d) the matrix $\boldsymbol{G}$ into matrices $\boldsymbol{G_1}$ for the spanning tree and $\boldsymbol{G_2}$ for the co-tree;

e) the matrix $\boldsymbol{F}$ into matrices $\boldsymbol{F_1}$ for the spanning tree and $\boldsymbol{F_2}$ for the co-tree.

3. Compute the matrix $\boldsymbol{L_{21}}$ by

$$\boldsymbol{L_{21}} = -\boldsymbol{T}_2 \boldsymbol{T}_1^{-T}$$

.

4. Provide an initial estimate of the co-tree flows $\boldsymbol{q_2^{(0)}}$

5. For i=1,2,. . . n until the stopping criteria has been met

a) get the corresponding spanning tree flows using

$$\boldsymbol{q}_1^{(i)} = \boldsymbol{L_{21}^T q_2^{(i)}} - \boldsymbol{T_1^{-T} d}. \tag{2.20}$$

b) Solve for the co-tree flows using

$$\boldsymbol{W}^{(i)} \boldsymbol{q}_2^{(i+1)} = \boldsymbol{L_{21}} \left(\boldsymbol{F}_1^{(i)} - \boldsymbol{G}_1^{(i)} \right) \boldsymbol{q}_1^{(i)} + \left(\boldsymbol{F}_2^{(i)} - \boldsymbol{G}_2^{(i)} \right) \boldsymbol{q}_2^{(i)} + \boldsymbol{L_{21} a_1} + \boldsymbol{a_2}. \tag{2.21}$$

6. The heads are found after the iterative process of the RCTM by using a linear solution process:

$$\boldsymbol{R}_1 \boldsymbol{h} = \boldsymbol{F}_1 \boldsymbol{q}_1^{(i+1)} - (\boldsymbol{F}_1 - \boldsymbol{G}_1) \boldsymbol{q}_1^{(i)} - \boldsymbol{a_1}. \tag{2.22}$$

This partitioning of the network equations reduces the size of the non-linear component of the solver to $n_p - n_j$ (the number of co-tree elements in the network). It has been shown in Elhay et al. (2014) that the RCTM and the GGA have identical iterative results and solutions if the same starting values are used. However, for RCTM, the user only needs to set the initial flow estimates for the co-tree pipes, $q_2^{(0)}$, in contrast to GGA where initial flow estimates are required for all pipes. The flows in the complementary spanning tree pipes are generated by Eq.(2.20).

Chapter 3

Publication 1: WDSLib: A Water Distribution System Simulation Test Bed

3.1 Synopsis

Water distribution system solution methods have been frequently used in WDS design, management and operation. In each of the above WDS simulation settings, often a WDS network with a fixed network topology needs to be solved many times. Thus, choosing the most suitable solution method can significantly improve the efficiency in a given setting.

Previous work on WDS simulation has focused on two research areas: (1) hydraulic solution methods and (2) solver software design. There is a disconnect between the two research areas. The researchers in the area of software design have focused on developing EPANET-based toolkits that are thread-safe and object-oriented. At the same time, the researchers in the area of improving the hydraulic solution methods have not developed a universal and reliable toolkit to implement, test and compare different WDS solution methods. In this chapter, WDSLib, a numerically robust, efficient and accurate C++ library, is described. WDSLib is written using a modular object-oriented design which allows users to easily mix and interchange solution components.

In this newly developed WDS software package, four WDS simulation methods are currently implemented, namely the global gradient algorithm (GGA), the GGA with the forest-core partitioning algorithm (FCPA), the reformulated co-tree flows method (RCTM), and the RCTM with the FCPA. WDSLib offers users the ability to: (i) choose from, or modify, different approaches and implementations of different WDS model analyses, and (ii) extend the toolkit to include new developments.

3.1.1 Citation

Qiu, M, Alexander, B, Simpson, AR & Elhay, S 2018, 'WDSLib: A Water Distribution System Simulation Test Bed', Manuscript submitted for publication to *Environmental Modelling & Software*.

Statement of Authorship

Title of Paper	A Benchmarking Study of Water Distribution System Solution Methods
Publication Status	☐ Published ☐ Accepted for Publication ☑ Submitted for Publication ☐ Unpublished and Unsubmitted work written in manuscript style
Publication Details	Qiu, M, Simpson, AR, Elhay, S & Alexander, B 2018a, 'A Benchmarking Study of Water Distribution System Solution Methods', Manuscript submitted for publication to Journal of Water Resources Planning and Management.

Principal Author

Name of Principal Author (Candidate)	Mengning Qiu
Contribution to the Paper	Developed the software package, conducted numerical analysis and prepared manuscript.
Overall percentage (%)	75%
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.
Signature	Date 30/04/2018

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Sylvan Elhay.
Contribution to the Paper	Helped in data interpretation, model development, manuscript evaluation and manuscript editing. (15%)
Signature	Date 30-Apr-2018

Name of Co-Author	Angus R. Simpson
Contribution to the Paper	Provided manuscript evaluation and manuscript editing. (5%)
Signature	Date 30 April 2018

Name of Co-Author	Bradley Alexander		
Contribution to the Paper	Provided manuscript evaluation and manuscript editing. (5%)		
Signature		Date	1/05/18.

3.2 Highlights

- A library for the steady-state analysis of a water distribution system (WDS)
- An open-source C++ software implementation of a number of WDS solution methods.
- A fast simulation platform for both once-off and multi-run simulations
- A timing model to parameterize multiple simulation times is introduced.
- Several improvements to the existing solution methods have been made.

3.3 Abstract

WDSLib is an extensible simulation toolkit for the steady-state analysis of a water distribution system. It includes a range of solution methods: the forest-core partitioning algorithm, the global gradient algorithm, the reformulated co-tree flow method, and also combinations of these methods. WDSLib has been created using a modularized object-oriented design and implemented in the C++ programming. WDSLib has been designed to: avoid unnecessary computations by hoisting each of the modules to its appropriate level of repetition, perform the computations independently of measurement units using scaled variables, accurately report the execution time of all the modules to parameterize multiple simulation times from a series of sampling simulation runs, and guard against numerical failures. WDSLib can be used to: implement, test and compare different solution methods; focus the research on the most time-consuming parts of a solution method; and guide the choice of solution method when multiple simulation runs are required.

3.3.1 Software availability

Name of the Software: WDSLib
Version: 1.0
Available from: https://github.com/a1184182/WDSLib
Language: C++
Year first available: 2018

3.3.2 Keywords

Water Distribution System; C++ toolkit; Object-Oriented design; Forest-Core Partitioning Algorithm; Reformulated Co-tree Flows Method; Global Gradient Algorithm; open source software

3.4 Introduction

Hydraulic simulation has been used to model water distribution systems (WDSs) for several decades and is an essential tool for the design, operation, and management of WDSs in industry and research. Hydraulic simulation allows users (1) to optimize WDS network parameters, such as pipe diameters, in a design setting, (2) to calibrate network parameters, such as demand patterns, in a conventional operational setting, (3) to conduct real-time monitoring and calibration of the network elements in a supervisory control and

data acquisition (SCADA) operational setting, and (4) to adjust control devices, such as valves, in a management setting. In the design setting and both the above operational settings, repeated hydraulic assessment is required on a network with fixed topology. In the management setting, repeated hydraulic assessment is required on a network with flexible network parameter settings. With ever-increasing network sizes and the need for real-time management using a SCADA system, it is important to have a robust simulation package which can be configured to be maximally efficient whatever the setting.

In the field of hydraulic simulation, the system of equations can be formulated as a large and sparse non-linear saddle point problem. There are several well-known iteration methods for solving the non-linear saddle point problem. These include: range space methods (Global Gradient Algorithm (Todini and Pilati 1988)), Null space methods (Co-Tree flow formulation variations (Rahal 1995; Elhay et al. 2014)), and loop-based methods (Loop flow correction (Cross 1936)). Their relative performance in terms of speed, rate-of-convergence, and accuracy depends among other things on the topology of the target network: size of the forest component, the number of network loops, and the density of these network loops. It is difficult to evaluate the impact of these topology factors by only examining the incidence matrix that describes the pipe network connectivity. As a result, the best method to use for a particular network cannot be easily determined a priori. Moreover, extra complexity is introduced when a multi-run hydraulic assessment is required. During a multi-run hydraulic simulation, the elapsed computation time of each method can be broken down into two parts: the components that are only required to be performed once at the very beginning for the same network, called the overhead, and the components that are required to be carried out repeatedly for each separate run until the required number of iterations has been met, called the hydraulic-phase. It is desirable to have a simulation platform, given the different levels of repetition, to implement these alternative algorithms efficiently. Equipped with such a platform a user would be able to easily benchmark the performance of alternative methods on a small number of evaluations for a given network and use that performance to inform the choice of algorithm to use for either a once-off simulation setting or for a multiple simulation setting (such as for an evolutionary algorithm (EA)).

This work describes an extensible WDS simulation platform called WDSLib. WDSLib is a numerically robust, efficient and accurate C++ library that implements many WDS simulation methods. WDSLib is written using a modular object-oriented design which allows users to easily mix and interchange solution components, thereby enabling users to avoid redundant computations. It has been optimized to use sparse data structures which are oriented to the pattern of access required for each solution method. WDSLib has been validated for accuracy on a range of realistic benchmark water distribution networks against reference implementations and tested for speed. The program accepts the input file formats of the industry standard EPANET2 (Rossman 2000) toolkit and its performance is faster than EPANET2 in all tested settings and benchmarks.

The remainder of this paper is structured as follows. The next section describes related methodologies and implementations. A general description of the WDS demand-driven steady-state problem is given in the next section. Section 3.6 presents a mathematical formulation of the network and the solution methods that are used in WDSLib. The tool-kit structure is then given in section 3.7. This is followed, in section 3.8, by the toolkit implementation details. Section 3.9 provides some examples of how the toolkit can be utilized in a simulation work flow. The results are discussed in Section 3.10. Finally, section 3.11 summarizes the results of this paper and describes future extensions to the

toolkit.

3.5 Background

This section describes related water distribution system network solution methods and implementations. The first sub-section describes solution methods, including those used by WDSLib. This is followed by a description of currently available implementations and compares these with WDSLib.

3.5.1 Related Methods

This research considers a water distribution model made up of energy conservation equations and the demand driven model continuity equations. The Hardy Cross method (Cross 1936), also known as the loop flow corrections method, is one of the oldest methods and uses successive approximations, solving for each loop flow correction independently. It is a method that was widely used for its simplicity at the time when it was introduced. More than three decades later, Epp and Fowler (1970) developed a computer version of Cross's method and replaced the numerical solver with the Newton method, which solves for all loop flow corrections simultaneously. However, this method has not been widely used because of the need (i) to identify the network loops, (ii) to find initial flows that satisfy continuity and (iii) to use pseudo-loops.

The GGA is a range space method that solves for both flows and heads. It was the first algorithm, in the field of hydraulics, to exploit the block structure of the Jacobian matrix to reduce the size of the key matrix in the linearization of the Newton method. The GGA has gained popularity through its rapid convergence rate for a wide range of starting values. This is the result of using the Newton method on an optimizations problem that has a quadratic surface. However, it was reported by Elhay and Simpson (2011) that the GGA fails catastrophically in the presence of zero flows in a WDS when the head loss is modeled by the Hazen-Williams formula. Regularization methods have been proposed by both Elhay and Simpson (2011) and Gorev et al. (2012) to deal with zero flows when the head loss is modeled by the Hazen-Williams formula.

The GGA as it was first proposed, applied only for the WDSs in which the head loss is modeled by the Hazen-Williams formula, where the resistance factor was independent of flow. Rossman (1994) extended the GGA to allow the use of the Darcy-Weisbach formula. It has been pointed out in Simpson and Elhay (2010), however, that Rossman incorrectly treated the Darcy-Weisbach resistance factor as independent of the flow. They introduced the correct Jacobian matrix to deal with this. It has been demonstrated that once the correct Jacobian matrix is used, the quadratic convergence rate of the Newton method is restored. Furthermore, Elhay and Simpson (2011) reported that the GGA does not fail in the presence of zero flows when the derivatives of the Darcy-Weisbach Jacobian matrix are correctly computed for laminar flows.

The co-trees flow method (CTM) (Rahal 1995) is a null space method that solves for the co-tree flows and spanning tree flows separately. The CTM, unlike the loop flow corrections method, does not require the initial flows to satisfy continuity. However, it does require: (i) the identification of the associated circulating graph; (ii) the determination of the demands that are to be carried by tree branches; (iii) finding the associated chain of branches closing a circuit for each co-tree chord; (iv) computing pseudo-link head losses. The reformulated co-trees flow method (RCTM) (Elhay et al. 2014) is also a null space

method that solves for co-tree flows and spanning trees flows separately. It represents a significant improvement on the CTM by removing requirements (i) to (iv) above. It uses the Schilders' factorization (Schilders 2009) to permute the node-arc incidence matrix into an invertible spanning tree block and a co-tree block. This permutation reduces the size of the Jacobian matrix from the number of junctions (as in the GGA) to approximately the number of loops in the network.

Abraham and Stoianov (2015) proposed a novel idea to speed-up the solution process when using a null space method to solve a WDS network. Their idea exploits the fact that a significant proportion of run-time is spent computing the head losses. At the same time, flows within some pipes exhibit negligible changes after a few iterations. As a result, there is no point in wasting computer resources to re-compute the pipe head losses for the pipes that have little or no change in flows. This partial update can be used to economize the computational complexity of the GGA, the RCTM and their variations.

The forest-core partitioning algorithm (FCPA) (Simpson et al. 2012) speeds up the solution process in the case where the network has a significant forest component. This algorithm permutes the system equations to partition the linear component of the problem, which is the forest of the WDS, from the non-linear component, which is the core of the WDS. It can be viewed as a method that simplifies the problem by solving for the flows and the heads in the forest just once instead of at every iteration. The FCPA reduces the number of pipes, number of junctions, and the dimension of the Jacobian matrix in the core by the number of forest pipes (or nodes).

The graph matrix partitioning algorithm(GMPA) (Deuerlein et al. 2015) exploited the linear relationships between flows of the internal trees within the core and the flows of the corresponding super-links after the forest of the network has been removed. This was a major breakthrough. The GMPA permutes the node-arc incidence matrix in such a way that all of the nodes with degree two in the core can be treated as a group. By partitioning the network this way, the network can be solved by a global step, which solves for the nodes with degree greater than two (super nodes) and the pipes which connect to them (path chords), and a local step, which solves for the nodes with degree two (interior path nodes) and pipes connected to them (path-tree links).

3.5.2 Related Implementations

EPANET 2 (Rossman 2000) is a widely used WDS simulation package. EPANET 2 implemented the GGA to provide a demand-driven steady-state solution of a WDS. The code for EPANET 2 is in the public domain, allowing many extensions to be developed. Currently available extensions include: the implementation of a pressure-dependent model (Cheung et al. 2005; Morley and Tricarico 2008; Siew and Tanyimboh 2012; Jun and Guoping 2012) and a real-time simulation capability (Vassiljev and Koppel 2015).

The EPANET 2 implementation is not explicitly designed to necessarily be easy to understand or accommodate alternative solution methods (Guidolin et al. 2010). The elements that are used in EPANET 2 are stored by the variables that describe their graph properties. For example, (1) junctions, reservoirs, and tanks are stored as a C struct called *Node* and (2) all valves, pipes, and pumps are stored as a C struct called *Link*. The abundant use of global variables limits the reusability and the possibility of the thread-safe design (Guidolin et al. 2010).

Consequently, it is difficult to cleanly incorporate new solution methods into EPANET 2 in a manner that allows a fair comparison of performance between these methods. Moreover,

because there are no clearly defined interfaces for the incorporation of third-party code components in EPANET 2, there is no guarantee that independently authored extensions will be easy to combine with each other.

In the absence of a popular easy-to-modify WDS simulation platform there is currently no straightforward means for comparing different solution methods. To date, when new solution methods have been developed they have been compared using different research systems, on different platforms with different implementation languages. This leads to difficulty in comparing methods, limits the reusability of code, and creates a barrier for researchers to reproduce and replicate results. To address these issues, an extensible framework is required that allows implementation of new methodologies to be easily incorporated without an adverse impact on the performance of the rest of the system.

To this end, a number of attempts have been made to implement an object-oriented wrapper to encapsulate the EPANET 2 solver (openNet (Morley et al. 2000) and OOTEN(van Zyl et al. 2003)). However, these two systems were focused on providing more flexibility in the processing of input to the core EPANET solver. They did not address any issues relating to the solution process. CWSnet, a C++ implementation in object-oriented style, was produced by Guidolin et al. (2010) as an alternative to EPANET 2.0. In CWSnet, more attention has been given to the hydraulic elements of the WDS network. In addition, CWSNet provides a pressure driven model, and takes advantage of the computing power of the computer's Graphics Processing Unit (GPU). However, in CSWnet the data structures representing the network are specialized to the solution methods that it uses. These data structures are not easily adapted to work efficiently with the different traversal orders, and graph algorithms used by newly developed solution methods. However, CWSnet still uses the same hydraulic solver and the same linear solver techniques implemented in EPANET 2 (Guidolin et al. 2010).

To accommodate the deficiencies referred to above, this paper presents a new hydraulic simulation toolkit WDSlib. WDSlib is coded in C++, and incorporates a number of recently published techniques. This toolkit offers users the ability to: (i) choose from, or modify, different approaches and implementations of different WDS model analyses, and (ii) extend the toolkit to include new developments. These features have been implemented using fast and modularized code. A focus of attention in this research has been program correctness, robustness and code efficiency. The correctness of the toolkit has been validated against a reference MATLAB implementation. The differences between all results (intermediate and final) produced by the C++ toolkit and the MATLAB implementation were shown to be smaller than 10^{-10}. In the interest of toolkit robustness, special attention has been paid to numerical processes to guard against avoidable failures, such as loss of significance through subtractive cancellation, and numerical errors, such as division by zero. The data structures and code libraries in WDSLib are shared and all implementations have been carefully designed to ensure fairness of performance comparisons between algorithms. WDSLib uses a pluggable architecture where solution-methods, and their accompanying pre-processing and post-processing code are easily substituted. In addition, different numerical linear algebra techniques can be incorporated using a well-defined interface. This concludes the discussion of related work. The mathematical formulations of the solution methods used in WDSLib are presented in the next section.

3.6 General WDS Demand-Driven Steady-State Problem

This section describes the general WDS demand-driven steady-state problem. The following starts with the basic definitions and notation, followed by the system equations. Finally, the relevant equations are shown for each of the different solution methods that are implemented in WDSLib. All variables are described in the nomenclature section in Appendix E in Section 3.17.

3.6.1 Definitions and Notation

Consider a water distribution system that contains n_p pipes, n_j junctions, n_r fixed head nodes and n_f forest pipes and nodes. The $j-th$ pipe of the network can be characterized by its diameter D_j, length L_j, resistance factor r_j. The $i-th$ node of the network has two properties: its nodal demand d_i and its elevation z_i.

Let $\boldsymbol{q} = (q_1, q_2, ...q_{n_p})^T$ denote the vector of unknown flows, $\boldsymbol{h} = (h_1, h_2, ...h_{n_j})^T$ denote the vector of unknown heads, $\boldsymbol{r} = (r_1, r_2, ...r_{n_p})^T$ denote the vector of resistance factors, $\boldsymbol{d} = (d_1, d_2, ...d_{n_j})^T$ denote the vector of nodal demands, $\boldsymbol{e_l} = (e_{l_1}, e_{l_2}....e_{l_{n_f}})^T$ denote the vector of fixed head elevations.

The head loss exponent n is assumed to be dependent only on the head loss model: $n = 2$ for the Darcy-Weisbach head loss model and $n = 1.852$ for Hazen-Williams head loss model. The head loss within the pipe j, which connects the node i and the node k, is modelled by $h_i - h_k = r_j q_j |q_j|^{n-1}$. Denote by $\boldsymbol{G}(\boldsymbol{q}) \in \mathbb{R}^{n_p \times n_p}$, a diagonal square matrix with element $[\boldsymbol{G}]_{jj} = r_j |q_j|^{n-1}$ for $j = 1, 2, ...n_p$. Denote by $\boldsymbol{F}(\boldsymbol{q}) \in \mathbb{R}^{n_p \times n_p}$, a diagonal square matrix where the j-th element on its diagonal $[\boldsymbol{F}]_{jj} = \frac{d}{dq_j}[\boldsymbol{G}]_{jj} q_j$. $\boldsymbol{A_1}$ is the full rank, unknown head, node-arc incidence matrix, where $[\boldsymbol{A_1}]_{ji}$ is used to represent the relationship between pipe j and node i; $[\boldsymbol{A_1}]_{ji} = -1$ if pipe j enters node i, $[\boldsymbol{A_1}]_{ji} = 1$ if pipe j leaves node i, and $[\boldsymbol{A_1}]_{ji} = 0$ if pipe j is not connected to node i. $\boldsymbol{A_2}$ is the fixed-head node-arc incidence matrix, where $[\boldsymbol{A_2}]_{ji}$ is used to represent the relationship between pipe j and fixed head node i, $[\boldsymbol{A_2}]_{ji} = -1$ if pipe j enters fixed head node i, $[\boldsymbol{A_2}]_{ji} = 1$ if pipe j leaves fixed head node i, and $[\boldsymbol{A_2}]_{ji} = 0$ if pipe j is not connected to fixed head node i.

3.6.2 System of Equations

The steady-state flows and heads in the WDS system are modeled by the demand-driven model (DDM) continuity equations (1) and the energy conservation equations (2):

$$-\boldsymbol{A_1}^T \boldsymbol{q} - \boldsymbol{d} = \boldsymbol{O} \tag{3.1}$$

$$\boldsymbol{G}(\boldsymbol{q})\boldsymbol{q} - \boldsymbol{A_1}\boldsymbol{h} - \boldsymbol{A_2}\boldsymbol{e_l} = \boldsymbol{O}, \tag{3.2}$$

which can be expressed as

$$\begin{pmatrix} \boldsymbol{G}(\boldsymbol{q}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{h} \end{pmatrix} - \begin{pmatrix} \boldsymbol{A_2}\boldsymbol{e_l} \\ \boldsymbol{d} \end{pmatrix} = 0, \tag{3.3}$$

where its Jacobian matrix is

$$\boldsymbol{J} = \begin{pmatrix} \boldsymbol{F}(\boldsymbol{q}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \tag{3.4}$$

and it is sometimes referred to as a nonlinear saddle point problem (Benzi et al. 2005).

This non-linear system is normally solved by the Newton method, in which $\boldsymbol{q}^{(m+1)}$ and $\boldsymbol{h}^{(m+1)}$ are repeatedly computed from $\boldsymbol{q}^{(m)}$ and $\boldsymbol{h}^{(m)}$ by

$$\begin{pmatrix} \boldsymbol{F}^{(m)}(\boldsymbol{q}^{(m)}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}^{(m+1)} - \boldsymbol{q}^{(m)} \\ \boldsymbol{h}^{(m+1)} - \boldsymbol{h}^{(m)} \end{pmatrix} = - \begin{pmatrix} \boldsymbol{G}^{(m)}\boldsymbol{q}^{(m)} - \boldsymbol{A_1}\boldsymbol{h}^{(m)} - \boldsymbol{A_2}\boldsymbol{e_l} \\ -\boldsymbol{A_1}^T\boldsymbol{q}^{(m)} - \boldsymbol{d}, \end{pmatrix} \quad (3.5)$$

until the relative differences $\frac{||\boldsymbol{q}^{(m+1)}-\boldsymbol{q}^{(m)}||}{||\boldsymbol{q}^{(m+1)}||}$ and $\frac{||\boldsymbol{h}^{(m+1)}-\boldsymbol{h}^{(m)}||}{||\boldsymbol{h}^{(m+1)}||}$ are sufficiently small.

3.6.3 Global Gradient Algorithm (GGA)

Todini and Pilati (1988) applied block elimination to Eq. (3.5) to yield a two-step Hazen-William solver: Eq. (3.6) for the heads and Eq. (3.7) for the flows.

$$\boldsymbol{h}^{(m+1)} = \boldsymbol{U}^{-1}\left\{-n\boldsymbol{d} + \boldsymbol{A_1}^T[(1-n)\boldsymbol{q}^{(m)} - \boldsymbol{G}^{-1}\boldsymbol{A_2}\boldsymbol{e_l}]\right\} \quad (3.6)$$

$$\boldsymbol{q}^{(m+1)} = \frac{1}{n}\left\{(n-1)\boldsymbol{q}^{(k)} + \boldsymbol{G}^{-1}(\boldsymbol{A_2}\boldsymbol{e_l} + \boldsymbol{A_1}\boldsymbol{h})\right\} \quad (3.7)$$

Later, Simpson and Elhay (2010) proposed

$$\boldsymbol{V}\boldsymbol{h}^{(m+1)} = -\boldsymbol{d} + \boldsymbol{A_1}^T\boldsymbol{F}^{-1}\left[(\boldsymbol{G}-\boldsymbol{F})\,\boldsymbol{q}^{(m)} - \boldsymbol{A_2}\boldsymbol{e_l}\right] \quad (3.8)$$

where $\boldsymbol{V} = \boldsymbol{A_1}^T\boldsymbol{F}^{-1}\boldsymbol{A_1}$

$$\boldsymbol{q}^{(m+1)} = \boldsymbol{q}^{(m)} + \boldsymbol{F}^{-1}\boldsymbol{A_1}\boldsymbol{h}^{(m+1)} - \boldsymbol{F}^{-1}\left[\boldsymbol{G}\boldsymbol{q}^{(m)} - \boldsymbol{A_2}\boldsymbol{e_l}\right] \quad (3.9)$$

as the generalized equations that can be applied when the head-loss is modeled by the Hazen-Williams equation or the Darcy-Weisbach equation. The correct Jacobian matrix with the formula for $\boldsymbol{F}$, when head loss is modeled by Darcy-Weisbach equation, can be found in Simpson and Elhay (2010). They showed that the use of the correct Jacobian matrix restores the quadratic rate of convergence.

It is important to note that the GGA, as it was originally proposed, solves the entire network by a non-linear solver, and this can include some unnecessary computations which can be avoided by exploiting the structural properties of the WDS graph composition. The methods described below exploit these structural properties to potentially improve the speed of the solution process.

3.6.4 Forest-Core Partitioning (FCPA)

Associated with a WDS is a graph $G = (V, E)$, where the elements of V are the nodes (vertices) of the graph G and elements of E are the pipes (links) of the graph G. The graph G can be partitioned into smaller subgraphs with special properties. The special properties that are exploited in WDSLib and their formulations are described in this subsection. The concept of partitioning the WDS network was proposed by Deuerlein (2008) in order to simplify the WDS solution process. Simpson et al. (2012) extended the idea of the network partitioning of Deuerlein (2008) and introduced the forest-core partitioning algorithm (FCPA), which partitions the network into a treed component and a looped or core component. The FCPA starts with a searching algorithm which identifies the forest subgraph, $G_f = \left(V_f, E_f\right)$, in which $\boldsymbol{S} \in \mathbb{N}^{n_f \times n_p}$ is the permutation matrix

which identifies the pipes in the forest, E_f, as distinct from the pipes in the core , E_c, and $\boldsymbol{T} \in \mathbb{N}^{n_f \times n_j}$ is the permutation matrix which identifies the nodes in the forest, V_f, as distinct from the nodes in the core, V_c, as distinct from the core subgraph, $G_c = (V_c, E_c)$, in which $\boldsymbol{P} \in \mathbb{N}^{n_{pc} \times n_p}$ is the permutation matrix for E_c and $\boldsymbol{C} \in \mathbb{N}^{n_{jc} \times n_j}$ is the permutation matrix for V_c.

The flows of the pipes in the forest, $\boldsymbol{Sq}$, can be found directly from

$$\boldsymbol{Sq} = -\left(\boldsymbol{T}{\boldsymbol{A_1}}^T\boldsymbol{S}^T\right)^{-1}\boldsymbol{Td}. \tag{3.10}$$

The system for the reduced non-linear problem (for the core heads and flows) can be expressed as

$$\begin{pmatrix} \boldsymbol{PGP}^T & -\boldsymbol{PA_1C}^T \\ -\boldsymbol{C}{\boldsymbol{A_1}}^T\boldsymbol{P} & \boldsymbol{O} \end{pmatrix}\begin{pmatrix} \boldsymbol{Pq} \\ \boldsymbol{Ch} \end{pmatrix} = \begin{pmatrix} \boldsymbol{PA_2e_l} \\ \boldsymbol{Cd} + \boldsymbol{C}{\boldsymbol{A_1}}^T\boldsymbol{S}^T\boldsymbol{Sq} \end{pmatrix}, \tag{3.11}$$

and then the Newton iterative method is applied to Eq. (3.11).

Finally, once the iterative solution process for the core has stopped, the forest heads can be found by solving a linear system:

$$\boldsymbol{Th} = \left(-\boldsymbol{SA_1T}^T\right)^{-1}\left(\boldsymbol{SA_2e_l} - \boldsymbol{SGS}^T\boldsymbol{Sq} + \boldsymbol{SA_1C}^T\boldsymbol{Ch}\right). \tag{3.12}$$

The system for the reduced non-linear problem (for the core heads and flows) in Eq. (3.11) can be expressed as:

$$\begin{pmatrix} \widehat{\boldsymbol{G}} & -\widehat{\boldsymbol{A}}_{\boldsymbol{1}} \\ -{\widehat{\boldsymbol{A}}_{\boldsymbol{1}}}^T & \boldsymbol{O} \end{pmatrix}\begin{pmatrix} \widehat{\boldsymbol{q}} \\ \widehat{\boldsymbol{h}} \end{pmatrix} = \begin{pmatrix} \widehat{\boldsymbol{A}}_{\boldsymbol{2}}\boldsymbol{e_l} \\ \widehat{\boldsymbol{d}} \end{pmatrix} \tag{3.13}$$

where $\widehat{\boldsymbol{G}} = \boldsymbol{PGP}^T, \widehat{\boldsymbol{A}}_{\boldsymbol{1}} = \boldsymbol{PA_1C}^T, \widehat{\boldsymbol{q}} = \boldsymbol{Pq}, \widehat{\boldsymbol{h}} = \boldsymbol{Ch}, \widehat{\boldsymbol{A}}_{\boldsymbol{2}} = \boldsymbol{PA_2}$, and $\widehat{\boldsymbol{d}} = \boldsymbol{Cd} + \boldsymbol{C}{\boldsymbol{A_1}}^T\boldsymbol{S}^T\boldsymbol{Sq}$.

The FCPA simplifies the problem by identifying the linear part of the problem and solving it separately from the core to avoid unnecessary computation in the iterative process.

3.6.5 Reformulated Co-Tree flows Method (RCTM)

A graph, with or without forest, can be partitioned into two sub-graphs: a spanning tree subgraph and a complementary co-tree subgraph. The reformulated co-tree flow method (RCTM) (Elhay et al. 2014) exploited the relationship between the spanning tree pipes and the co-tree pipes. The RCTM starts with a spanning tree search algorithm which identifies a spanning tree subgraph, $G_{st} = (V, E_{st})$, in which $\boldsymbol{K_1} \in \mathbb{N}^{n_{p_{st}} \times n_p}$ is the permutation matrix that identifies the pipes in the spanning tree, E_{st}, as distinct from the pipes in the co-tree, E_{ct}. $\boldsymbol{R}$ is the permutation matrix for the nodes which traverse the same sequence as the corresponding spanning tree pipes, E_{st}, and $\boldsymbol{K_2} \in \mathbb{N}^{n_{p_{ct}} \times n_p}$ is the permutation matrix for the pipes in the complementary co-tree edges, E_{ct}. It is important to note that there are many choices of spanning tree for any cyclic graph. The choice of spanning tree and co-tree combination does not affect the correctness of the method.

By exploiting the relationship between the spanning tree and cotree, Elhay et al. (2014) proposed the following equations to solve the WDS for the flows: first for the spanning tree flows $\boldsymbol{q_1^{(m+1)}}$,

$$\boldsymbol{q}_1^{(m+1)} = {\boldsymbol{L_{21}}}^T\boldsymbol{q}_2^{(m)} - \boldsymbol{R}_1^{-T}\widehat{\boldsymbol{d}} \tag{3.14}$$

and second for the co-tree flows $\boldsymbol{q_2^{(m+1)}}$:

$$\boldsymbol{W}^{(m+1)}\boldsymbol{q}_2^{(m+1)} = \boldsymbol{L_{21}}\left(\boldsymbol{F}_1^{(m+1)} - \boldsymbol{G}_1^{(m+1)}\right)\boldsymbol{q}_1^{(m+1)} + \left(\boldsymbol{F}_2^{(m)} - \boldsymbol{G}_2^{(m)}\right)\boldsymbol{q}_2^{(m)} + \boldsymbol{a_2} \tag{3.15}$$

where: $\boldsymbol{R_1} = \boldsymbol{K_1}\widehat{\boldsymbol{A_1}}\boldsymbol{R}^T$; $\boldsymbol{R_2} = \boldsymbol{K_2}\widehat{\boldsymbol{A_1}}\boldsymbol{R}^T$; $\boldsymbol{L_{21}} = -\boldsymbol{R_2}\boldsymbol{R_1}^{-T}$; $\boldsymbol{F}_1^{(m)} = \boldsymbol{K_1}\widehat{\boldsymbol{F}^{(m)}}\boldsymbol{K_1}^T$; $\boldsymbol{F}_2^{(m)} = \boldsymbol{K_2}\widehat{\boldsymbol{F}^{(m)}}\boldsymbol{K_2}^T$ $\boldsymbol{G}_1^{(m)} = \boldsymbol{K_1}\widehat{\boldsymbol{G}^{(m)}}\boldsymbol{K_1}^T$; $\boldsymbol{G}_2^{(m)} = \boldsymbol{K_2}\widehat{\boldsymbol{G}^{(m)}}\boldsymbol{K_2}^T$; $\boldsymbol{a_1} = \boldsymbol{K_1}\widehat{\boldsymbol{A_2}}\boldsymbol{e_l}$; $\boldsymbol{a_2} = \boldsymbol{L_{21}}\boldsymbol{K_1}\widehat{\boldsymbol{A_2}}\boldsymbol{e_l} + \boldsymbol{K_2}\widehat{\boldsymbol{A_2}}\boldsymbol{e_l}$; $\boldsymbol{W}^{(m)} = \boldsymbol{L_{21}}(\boldsymbol{F}_1^{(m)})^{-1}\boldsymbol{L_{21}}^T + (\boldsymbol{F}_2^{(m)})^{-1}$.

Note that in Eq. (3.14), an initial set of the co-tree flows $\boldsymbol{q_2^{(0)}}$ is needed to commence the solution process.

The heads are found after the iterative process of the RCTM has been completed by using a linear solution process:

$$\boldsymbol{R}_1\boldsymbol{h} = \boldsymbol{F}_1\boldsymbol{q}_1^{(m+1)} - (\boldsymbol{F}_1 - \boldsymbol{G}_1)\,\boldsymbol{q}_1^{(m)} - \boldsymbol{a_1} \tag{3.16}$$

This partitioning of the network equations reduces the size of the non-linear component of the solver to $n_p - n_j$ (the number of co-tree elements in the network). It has been proven by Elhay et al. (2014) that the RCTM and the GGA have identical iterative results and solutions if the same starting values are used. However, for the RCTM, the user only needs to set the initial flow estimates for the co-tree pipes, $\boldsymbol{q_2^{(0)}}$, in contrast to GGA where initial flow estimates are required for all pipes. The flows in the complementary spanning pipes are generated by Eq.(3.14) in the RCTM.

3.7 WDSLib Structure

WDSLib is a WDS simulation toolkit consisting of a set of C++ member functions, which henceforth will be referred to just as *functions*, that can be composed to solve for the steady state solution of a WDS. WDSLib can be used for a once-off simulation or a multi-run simulation. Pre-packaged driver code is provided to perform once-off simulations using a choice of solver methods. For a multi-simulation setting, where the use-cases are very diverse, the user is able to select the desired components of WDSLib to compose and compile their own driver.

Individual functions in WDSLib are classified according to their role in the simulation workflow. In any simulation workflow, there will be functions that will only have to be executed once. For example, functions to read the input file or partition the network will only have to execute once at the start of the simulation (or of all simulations). Likewise, code to reverse the network partitioning and write simulation results will only have to execute once at the end of the simulation. In this work, these functions that are only required to be run once are called level one (L_1) functions. L_1 functions relate to network topology, which is invariant for the whole simulation. In a multi-simulation setting, certain functions will need to be run once for every hydraulic-phase. An example of such a module is the module making the initial guesses of pipe flow rates for the updated network configuration. In this work, these, once-per-assessment functions, are called level two (L_2) functions.

Finally, for every hydraulic assessment there is a non-linear iterative phase in the solution process. The functions in this phase run many times for each hydraulic assessment until the stopping test has been satisfied. Examples of these include the functions to calculate the $\boldsymbol{G}$ and $\boldsymbol{F}$ matrices (see Eqs. (3) and (4)) and running the Cholesky solver. These iterative-phase functions are called level three (L_3) functions.

Fig. 3.1 illustrates the global structure of WDSLib under a once-off simulation setting and a multi-run simulation setting. The modular setup of WDSLib allows each module to be run the minimum number of times determined by its simulation setting. Under the module structure described above a once-off simulation setting can be viewed as a special case where the L_1 functions and L_2 functions are both run once. Note that after running

the initial L_1 functions it is possible to run hydraulic assessments of the network in parallel. This mode of execution might be used in a design setting such as using a genetic algorithm (GA) to optimize pipe diameter sizes.

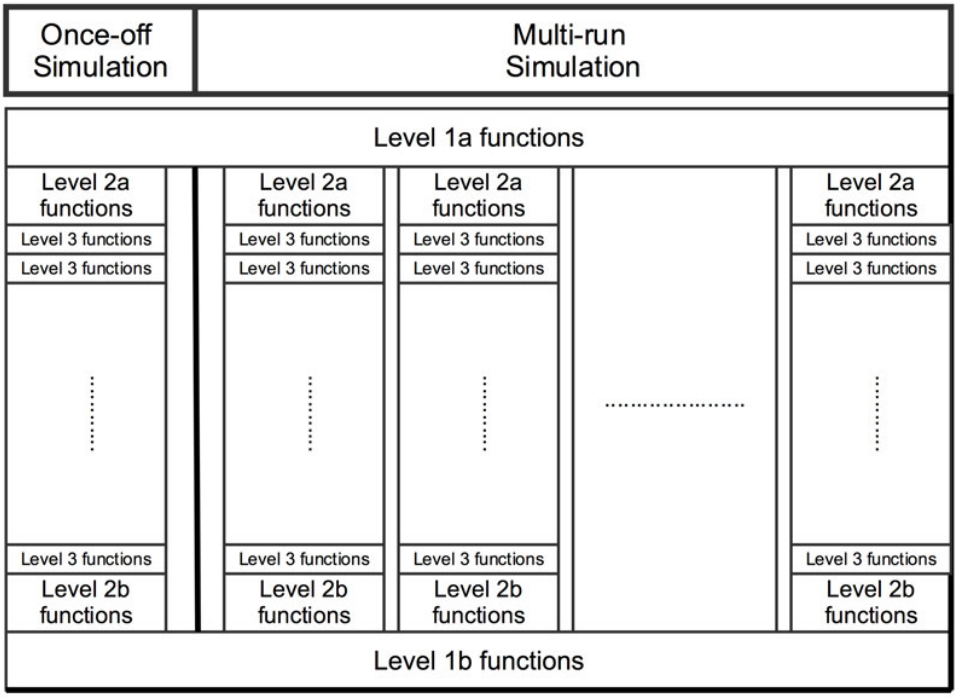


Fig. 3.1. Global structure of WDSLib for both simulation settings

L_1 and L_2 functions are classified into parts a and b according to whether they run before or after the lower level processing that they embed. These functions are detailed in Fig. 3.2. The L_1 functions that run at the start of the simulation are called L_{1a} functions. These include the module to read the configuration file and the EPANET $.inp$ file; partition the network; and solve the linear part(s) of the network. The corresponding L_{1b} functions are run at the end of the simulation. These include tasks such as reversing the network partitioning. Note that certain L_{1a} functions require their corresponding L_{1b} functions to be used. For example the forest search module needs to be paired with the reverse FCPA permutation. There is a similar structure for L_2 functions. L_{2a} functions are run at the start of each hydraulic assessment and L_{2b} functions run at the end. The functions that must be included for the FCPA method are denoted with single asterisks. Likewise the functions that must be included with the RCTM method are denoted with double asterisks. For these methods to work correctly all affiliated functions must be included in the simulation workflow. Note that it is also possible to run both the RCTM and FCPA in the same workflow. Also note that the user cannot run both GGA and RCTM in the same workflow – the user must choose between these solution methods.

Table 3.1 provides a mapping from the function descriptions in Fig. 3.2 to the function names in WDSLib. In addition, the dependencies between functions for each solution method are shown in Table 3.1a, Table 3.1b, Table 3.1c and Table 3.1d. The columns in each table list, respectively, the description of the function, its name in WDSLib, the C++ class in which it appears, its input parameters, and its output values. Note, that *void* is used in these latter two columns to denote that the function interacts with the class variables

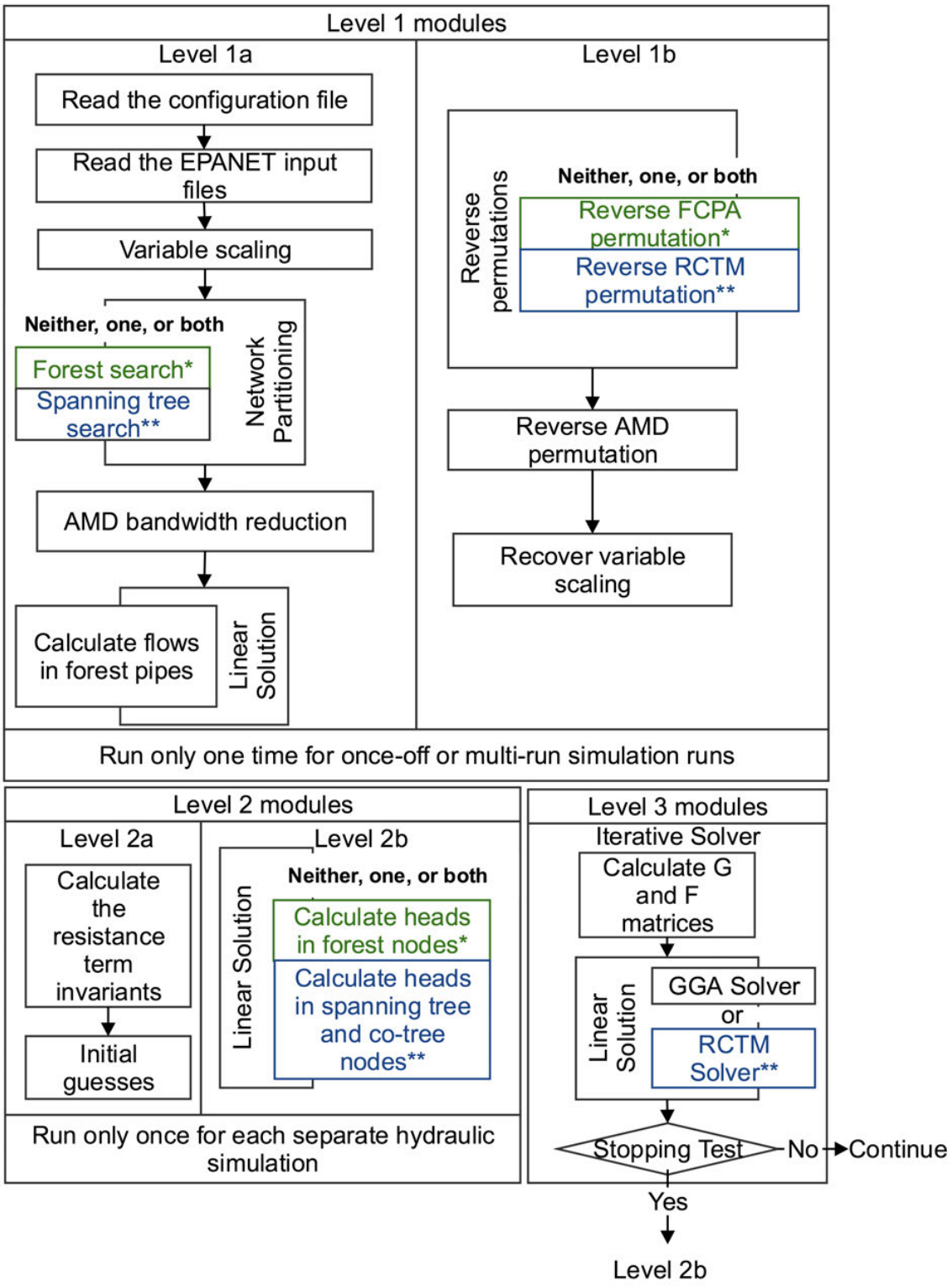


Fig. 3.2. Function classification in WDSLib. The functions marked with single asterisks must be used for the FCPA method. The functions marked with double asterisks must be used for the RCTM method. Note that it is possible to use both methods at the same time.

Table 3.1. Key function descriptions, names, their classes, inputs and outputs. The affiliated functions are shown in sub-tables (3.1a) (3.1b) (3.1c) (3.1d).

(a) Shared Modules

Description	Module name	Class	Input	Output
Read the configuration file	readConfig	runManager	config file name	void
Read EPANET input file	getInputData	Input	EPANET $.inp$ file	EPANET err code
Variables scaling	scale	Solver	void	void
AMD bandwidth reduction	AMD	Suitesparse	void	void
Calculate the resistance constants	getRf	Solver	net	resistance constant
Generate initial guesses of flows	init	Solver	diameter	flow rate
Calculate the head loss coefficients	getGF	Solver	net, resistance constant	void
Stopping test	stopTest	Solver	result	norm
Recover scaled variables	rScale	Solver	void	void

(b) Global gradient algorithm (GGA)

Description	Module name	Class	Input	Output
GGA Solver	runH	GGASolver(Solver)	void	void

(c) Forest-core algorithm (FCPA)

Description	Module name	Class	Input	Output
Forest search	forestSearch	topology	SN, EN	void
Calculate flows in forest	forestFlow	solver	demands	flows in forest pipes
Calculate heads in forest	forestHead	solv	result	heads in forest pipes
Reverse FCPA permutation	rFCPA	Solver	void	void

(d) Reformuated cotree flows method (RCTM)

Description	Module name	Class	Input	Output
Spanning tree search	STSearch	topology	SN, EN	void
RCTM solver	runH	RCTMsolver (Solver)	void	void
Calculate heads in ST and CT	RCTMHead	RCTMsolver	flows in ST and CT	void
Reverse RCTM permutation	rRCTM	RCTMsolver	void	void

rather than through its parameters and return value. Examples of how these functions can be coded are presented in section 3.9. The key data-access functions in WDSLib are described next.

Getter and Setter methods Each class in WDSLib has various methods available for setting the network parameters and retrieving the results of the WDS network. These methods allow the user to reconfigure the network before and during simulation runs. The names of the setter methods all start with a prefix *set* and the names of the getter methods all start with a prefix *get*. For example, a user can set (write-to) the diameter of pipe *index* to *value* by calling `pipe->setD(index,value)` and get (read-from) the head of node *index* by calling `h[index]=result->gethFinal(index)`. A summary of the variables that can be read-from (read-access through getter methods) and written-to (write-access through setter methods for each key classes is specified in Table 3.2. This concludes the discussion of the the broad structure of the WDSLib package. The next section describes key aspects of the implementation of the package.

Table 3.2. The getter and setter functions of each class and the variables they access

Class Name	Description	Read-Access	Write-Access
`Net`	Basic network properties, & `Pipe and Node`	`Node,Pipe,` n_p, n_j, n_s	
`Node`	Node properties	$\boldsymbol{d}$, $\boldsymbol{z_s}$, $\boldsymbol{z_u}$	
`Pipe`	Pipe properties	$\boldsymbol{SN}$, $\boldsymbol{EN}$, $\boldsymbol{L}$, $\boldsymbol{D}$, $\boldsymbol{R}$, pipe ID	
`Flag`	Flag information	getFlag("flagN",flagV)	setFlag("flagN")
`Parameter`	Parameter information	getPara("paraN",paraV)	setPara("paraN")
`Simulation`	Manage hydraulic simulation	-	-
`Solver`	Parent class of solution methods	-	-
`GGASolver`	GGA solution method	`Result`	-
`RCTMSolver`	RCTM solution method	`Result`	-
`Topology`	Network topology information	getCore, getForest	
`Result`	Results of the simulations	qIter, hIter, GIter , FIter, numIter, CresIter, EresIter, Time	

3.8 WDSLIB: Toolkit Implementation

This section outlines key implementation details of WDSLib. As previously mentioned, the overall aim of WDSLib is to provide a clearly-structured, flexible and extensible hydraulic simulation toolkit that allows testing, evaluation, and use, in production settings, of both existing and new WDS solution techniques. These aims require WDSLib be implemented so that it is fast to execute, flexible to configure, robust to challenging input data cases, and easy to understand and modify. The following describes aspects of the implementation of WDSLib that enable it to meet these requirements. The next subsection describes the general considerations that informed the design of the whole toolkit. This general discussion is followed by a summary of key improvements to the solution processes encoded in forest searching and spanning tree searching in the WDSLib package.

3.8.1 General capabilities and properties

This sub-section describes design aspects underpinning the utility and performance of WDSLib. In-turn, the following outlines measures taken to: (1) maximize code clarity and modularity; (2) increase the efficiency of memory access and storage; (3) maximize numerical robustness; (4) facilitate accurate timing of code execution; and (5) maximize simulation speed for different settings.

Design Considerations 1: Modularity

The modular design of WDSLib is central to the evaluation and testing of different WDS solution methods. All methods have been defined to perform a single, well–defined, function and each class can be compiled, used and tested independently. These features allow users to assemble the methods of interest from independently developed components to create a customized WDS solution method in a reliable way. WDSLib's modular design also allows the users to profile the computation time of each individual component of an algorithm. Functions communicate through well-defined interfaces and the function code has been factored to minimize development and testing cost. This architecture allows customized simulation applications (i) to combine the functions of interest and (ii) to implement new solution algorithms to extend the functionalities of WDSLib.

Design Considerations 2: Memory Considerations

Care was taken to minimize the memory footprint of executing code (in order to reduce memory requirements and prevent memory leaks) in the interest of the toolkit efficiency and toolkit robustness. Reducing memory requirements allows the solution of larger WDS problems for a given memory capacity. In WDSLib, memory reduction was achieved through both, using sparse matrix representations and the systematic allocation and deallocation of working structures in the C++ code. The matrices used in WDS simulation are often sparse, with the density of the full node-arc incidence matrix being only $2/n_j$. Consequently, it is more efficient to store these matrices using sparse storage schemes which store only the non-zero elements of the matrix and pointers to their locations (Davis et al. 2013). It is important to note that the choice of a sparse matrix representation is made based on (1) the storage requirements of the matrix and (2) common search orders to column elements and row elements. This latter factor means that the best format for sparse matrix representation varies with the preponderant orders of search, (row-wise, column-wise, or both), employed by each method. There is a number of common storage formats for sparse matrices (Compressed column storage (CCS) of Duff et al. (1989)), Compressed row storage (CRS), Block Compressed column storage (BCCS), Block Compressed row storage (BCRS), and Adjacency lists). As will be described shortly, WDSLib, uses a modified adjacency-list representation.

Other implementations use a variety of storage schemes. In EPANET 2, the $\boldsymbol{A_1}$ matrix is stored as two arrays of node indices, which represent start nodes (SN) and the end nodes (EN) of each pipe. The $i-th$ entry of the SN and EN arrays represent the start node and end node of $i-th$ pipe of the network. This storage format minimizes the memory required to store the $\boldsymbol{A_1}$ matrix because only the indices are required to be stored because $[\boldsymbol{A_1}]_{(i,SN_i)} = -1$ and $[\boldsymbol{A_1}]_{(i,EN_i)} = 1$. As shown in Table 3.4, searching through rows (pipes) of matrices that are stored in this format is efficient. However, searching though the columns (nodes) is relatively inefficient. This storage format is also used in CWSnet.

Both CCS and CRS are used in the FCPA implementation reported in Simpson et al. (2012), and the RCTM implementation reported in Elhay et al. (2014). The partial update null space method (Abraham and Stoianov 2015) used CCS. The memory requirement for storing the $\boldsymbol{A_1}$ matrix in CCS is $2 \times nnz + n_j + 1$ as shown in Table 3.4. This storage scheme is fast for searching through columns (nodes) of matrices that are stored in CCS and slow for searching though rows (pipes).

Table 3.3. The adjacency-list matrix presentation

Node Index	adjacent to	Size
1	$\{(v_i, e_j) \vert v_i \in N(v_1)$ e_j connects v_1 and $v_i\}$	Deg(v_1)
2	$\{(v_i, e_j) \vert v_i \in N(v_2)$ e_j connects v_2 and $v_i\}$	Deg(v_2)
3	$\{(v_i, e_j) \vert v_i \in N(v_3)$ e_j connects v_3 and $v_i\}$	Deg(v_3)
$\vdots$	$\vdots$	$\vdots$
n_j	$\{(v_i, e_j) \vert v_i \in N(v_{n_j})$ e_j connects v_{n_j} and $v_i\}$	Deg(v_{n_j})

In WDSLib, a modified adjacency list, described in Table 3.3, tailored for WDS hydraulic simulation, is used. An adjacency list for an undirected and unweighted graph consists of n_j unordered lists for each vertex n_i, which contains all the vertices to which vertex n_i is adjacent. The network that is shown in the Fig. 3.3 has one source, three

Table 3.4. Different sparse representations for $\boldsymbol{A_1}$

Types	size($\boldsymbol{A_1}$)	size($\boldsymbol{A_2}$)	size($[\boldsymbol{A_1}\ \boldsymbol{A_2}]$)	Column Search	Row Search
CCS	$2 \times nnz + n_j + 1$	$2 \times nnz + n_f + 1$	$4 \times n_p + n_n + 2$	$O(n)$	$O((n_j)n)$
CRS	$2 \times nnz + n_p + 1$	$2 \times nnz + n_p + 1$	$6 \times n_p + 2$	$O((n_p)n)$	$O(n)$
EPANET	-	-	$2 \times n_p$	$O(n)$	$O((n_j)n)$
WDSlib	-	-	$4 \times n_p$	$O(n)$	$O(n)$

nodes, and four pipes. The adjacency list for this network can be described by four lists $\{\{2,3\},\{1,4\},\{1,4\},\{2,3\}\}$. Each list describes the set of adjacent vertices of a vertex in the graph. For example, the first list, $\{2,3\}$, represents that the vertex 1 is adjacent to the vertex 2 and vertex 3.

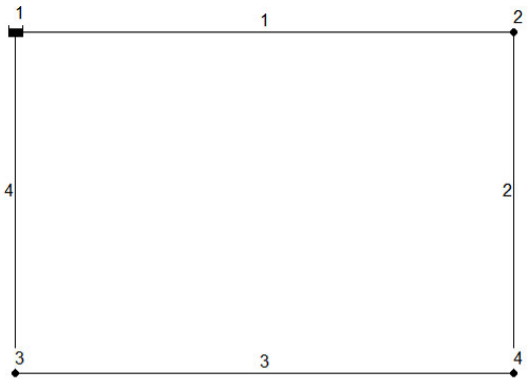


Fig. 3.3. A simple sample network. Numbers denote junction and pipe indices in the network.

The adjacency list is modified to include a directed and weighted graph for WDSLib. This modified adjacency list for a directed and weighted WDS graph consists of n_j unordered lists for each vertex n_i. This list contains all the vertex and edge pairs to which vertex n_i is adjacent. For example, the adjacency list for the same network that is shown in the Fig. 3.3 can be described by four lists $\{\{(2,1),(3,4)\},\{(1,1),(4,2)\},\{(1,4),(4,3)\},\{(2,2),(3,3)\}\}$. Each list represents the set of adjacent vertex and edge pair of a vertex in the graph. For example, the first list, $\{(2,1),(3,4)\}$, describes that the vertex 1 is adjacent to the vertex 2 by edge 1 and the vertex 3 by edge 4. It is fast to search through both the rows and columns of the $\boldsymbol{A_1}$ matrices that are stored in this format.

In addition to these optimized encodings, both $\boldsymbol{G}$ and $\boldsymbol{F}$ are diagonal square matrices, which require less storage when stored as vectors than in sparse matrix form. The storage methods used for the variables in WDSLib and their associated memory usage are given in Table 3.5.

As a final note, to offer further assurance of the correctness of memory management in WDSLib, Valgrind (Nethercote and Seward 2007), a programming debugging tool, was deployed during testing to detect any memory leaks, memory corruption, and double-freeing.

Table 3.5. Vectors and matrices in WDSLib

variables	type	size	storage method	memory requirements
$\boldsymbol{q}$, $\boldsymbol{L}$, $\boldsymbol{D}$, $\boldsymbol{r}$	vector	$n_p \times 1$	vector	$n_p\times$ double
$\boldsymbol{h}$, $\boldsymbol{d}$	vector	$n_j \times 1$	vector	$n_j\times$ double
$\boldsymbol{G}$, $\boldsymbol{F}$	matrix	$n_p \times n_p$	vector	$n_p\times$ double
$\boldsymbol{A_1}$, $\boldsymbol{A_2}$	matrix	$n_p \times n_j$	sparse matrix	$(2 \times n_p)\times$ integer
$\boldsymbol{L_{21}}$	matrix	$(n_p - n_j) \times n_j$	sparse matrix	$\leq (n_p - n_j) \times n_j \times integer$

Table 3.6. WDS variables and units

Variables	SI unit	US Customary unit	Scaling factor
Length	m	ft	$L_0 = max\,(\boldsymbol{L})$
Diameter	m	ft	$D_0 = max\,(\boldsymbol{D})$
Nodal head	m	ft	$h_0 = max\,(\boldsymbol{e_l})$
Source elevation	m	ft	$e_{l_0} = max\,(\boldsymbol{e_l})$
flow	m^3/s	ft^3/s	$q_0 = max\,(\boldsymbol{d})$
demand	m^3/s	ft^3/s	$d_0 = max\,(\boldsymbol{d})$
G, F	s/m^2	s/ft^2	$G_0 = \frac{L_0}{D_0^p}\lvert q_0\rvert^{n-1}$

Design Considerations 3: Numerical Considerations

The calculations in WDSLib are performed in C++ under IEEE-standard double precision floating point arithmetic with machine epsilon $\epsilon_{mach} = 2.22 \times 10^{-16}$. Invariant terms and parameters in every equation were evaluated in advance and replaced by full 20-decimal digit accuracy constants. Intermediate results of calculations, (which are not easily accessible in EPANET), can be output at the user's request. The stopping tolerance and stopping test can be set by the user either through the configuration file or by the relevant setter method in the `Parameter` class.

In the construction of any numerical solver, there are two primary dangers that are associated with floating point arithmetic that cannot be ignored: (i) subtractive cancellation and (ii) overflow and underflow. To avoid problems associated with these, all input variables are scaled to a similar range to minimize the risk of avoidable computational inaccuracy or failure in floating point arithmetic. It is important to note that unscaled or poorly scaled variables can unnecessarily confound a computation. These scaled input variables are physically dimensionless, which allows computation which is independent of the system of measurement units.

The variables, that are provided in EPANET input file for the package, and their corresponding units in US customary and SI units are shown in Table 3.6. As with the input variables, the system equations were modified to use dimensionless variables. Once the stopping test is satisfied, the original variables are then recovered by reversing the initial scaling. Details of the scaling are shown in the Appendix A in Section 3.13.

To help ensure that WDSLib solution methods are both fast and reliable. The sparse matrix operations are implemented using SuiteSparse (Davis et al. 2013). SuiteSparse is a state-of-the-art sparse arithmetic suite with exceptional performance, from which the approximate minimum degree permutation (AMD) and the sparse Cholesky decomposition routines have been used.

Design Considerations 4: Timing Considerations

When executing WDSLib, each function reports the time spent in it by sampling wall clock time at the start and end of its execution. Although the overhead for sampling wall clock time is small, there are at least two special considerations involved in the interpretation of these timings: (i) the operating system, at its own discretion, may launch background processes (for example anti-virus software), which will distort the timings and (ii) extrapolating the timing for multiple hydraulic simulations from a single analysis (as may be required, for example, in a genetic algorithm or other evolutionary algorithm run) must be done with care because the relationship between the different settings is not linear.

This concludes the discussion of the main considerations concerning the global design of WDSLib. In the following, key details of the implementation of selected parts of the solution processes are described.

3.8.2 Key Improvements to Solution Processes

The WDSLib implementation makes several improvements to extant solution processes. This section focuses on the improvement of the network partitioning processes in FCPA and RCTM.

Key Improvement 1: Improvement to partitioning in Forest Search

The forest-core partitioning algorithm (FCPA) in this paper is a substantial improvement over the algorithm of the original paper (Simpson et al. 2012). Specifically the original FCPA algorithm almost always required many sweeps of the columns (nodes) of the $\boldsymbol{A_1}$ matrix in order to reduce the forest component down to the core component. The refined algorithm exploits the adjacency list representation of the $\boldsymbol{A_1}$ matrix so that the partitioning process is achieved in a single sweep. This improves the speed of the partitioning process from being $O(an_p)$ to $O(n_p + n_f)$ where a is the depth of the deepest tree-component in the forest. This can lead to substantial time savings in the case when a is relatively large.

The pseudo-code for this refined forest search algorithm is shown in Appendix B in Section 3.14 This algorithm traverses each tree component in turn from its leaf nodes, which maximizes the locality of operations with respect to the graph representation. In this algorithm, a node is identified as a leaf node when its node degree is one. Every time a leaf node, node k, is identified, the node pointer is moved to its adjacent node, node k, and the node degree of node k is reduced by one. This process repeats if the adjusted node degree of node k is one. Otherwise, node k is the root node for this tree and the algorithm progresses to the next tree in the forest.

Key Improvement 2: Improvement to Spanning Tree Search

The reformulated co-tree flows method (RCTM) in this paper is also a substantial improvement over the algorithm of the original paper (Elhay et al. 2014). The original spanning tree search algorithm sweeps the rows of the $\boldsymbol{A_1}$ matrix (pipes) in order to identify the singleton rows and their corresponding columns. The spanning tree search in the original RCTM required a sweep of of the $\boldsymbol{A_1}$ matrix to identify the next pipe in the spanning tree. This algorithm is $O(n_p n_j)$, which is relatively inefficient.

The pseudo-code for the refined spanning tree search algorithm is shown in Appendix C in Section 3.15 This improved algorithm takes as input the adjacency list describing the

network and the pipe indexes of the core component of the network from the Algorithm 1 (if the FCPA is used). In this algorithm, all water sources are the starting point of the search process, SN, and marked as *visited*. The nodes in SN are then used as to identify a spanning tree within the WDS. This is achieved by repeatedly finding all adjacent pairs, node t and pipe s, of and removing the first node in SN by using the adjacency list. If the adjacent node t is not visited then node t is inserted into the spanning-tree node vector, STN, and search node vector, SN, and node t is marked as visited and pipe s to the spanning-tree pipe vector, STP, and pipe s is marked as visited. If the adjacent node t is visited and the pipe s is not visited then the pipe s is inserted into the co-tree pipe vector, CTP and mark pipe s as visited. This process is repeated until SN is empty. The overall time-complexity of this algorithm is $O(n_p + n_j)$ (compared to O($n_p n_j$) as mentioned above) is the same as the best asymptotic complexity of breadth-first search on a graph.

3.9 Example Applications

WDSLib consists of a collection of functions which can be used either as a standalone application for fast one-off simulations or as a library of software components that can be integrated into a user's own WDS solution processes. This section presents two example applications. The first application is the setup for a basic one-off simulation of a WDS. The second application (described in subsection 3.9.1) presents an example using WDSLib to implement a simple 1+1 Evolutionary Strategy (Beyer and Schwefel 2002) (1+1-ES or, more commonly, 1+1EA) for sizing pipes in a WDS.

Example 1 - Once-off Simulation

The setup for WDSLib as a standalone application is straightforward. The user provides a configuration text file that specifies input and output filenames; the name of the solver; the desired output variables; and simulation parameters. These values have sensible defaults so the user can set up the solver by using a minimal configuration such as that shown in Fig. 3.4. By using this config file, WDSLib is configured to run a single hydraulic analysis of the network that is stored as say "hanoi.inp", an EPANET- formatted input file, under "Network/" sub-directory, using the reformulated co-tree flows method with the forest-core partitioning algorithm. The full set of configuration parameters for once off simulations is shown in Fig. 3.10 in Appendix 3.16.

3.9.1 Example 2 - A Simple Network Design Application

As a minimalist example of the application of WDSLib to a WDS network design problem, the following example uses 1+1EA for optimally sizing pipe diameters. This algorithm takes an existing network with randomly generated pipe diameters and optimizes the network to minimize cost, subject to given pressure head constraints. A 1+1EA is a very simple evolutionary strategy (Beyer and Schwefel 2002) which starts with a randomly generated individual (in this case a WDS diameter configuration). This 1+1EA then progresses by applying a mutation to a random pipe diameter size, and then evaluating the new individual. If the new individual is better it replaces the old network. This process continues in a loop until a given number of evaluations is reached.

```
1  [InputFile]
2  <directory>        %the input file directory
3  Network/
4  <file>             %the input file name
5  hanoi.inp
6  <end>
7  %--------------------------------------------------%
8  [controlFlag]
9  <SolverFlag>       %1 for GGA 2 for RCTM
10 2
11 <FCPAFlag>
12 1
13 <end>
14 %--------------------------------------------------%
```

Fig. 3.4. A minimal configuration file to run the GGA in WDSLib

The C++ code for this example is shown in Figs. 3.5, 3.6, 3.7, and 3.8. If the name of the file containing this code is: `simpEA.cc` then the simplest command to compile this code is:

```
g++ simpEA.cc -o simpEA -Llib -lWDSLib
```

To run this code the user would type:

```
./simpEA config.txt
```

where `config.txt` contains the same configuration text as for the previous example.

Starting with the `main` function in Fig. 3.5, line 15 points to the config file specified by the command line. The next two lines initialize the `result` and the `simulation` according to the configuration file. This is followed by the L_{1a} module to perform the user selected L_{1a} functions. Line 19 generates the initial pipe diameters of the network and line 20 initializes the workspace for the mutated string. Line 23 sets the pipe diameters of the network. Line 24 evaluates the current network configuration. The permutation and scaling for the current individual is reversed by L_{1b} in line 25 of Fig. 3.5. Line 26 calculates the fitness of the current network configuration by using the `evaluate` function in Fig. 3.8. This function applies a penalty for pressure head constraint violations and pipe material costs. The body of the 1+1EA is contained in the selection operator and mutation operator that follow. Lines 27 to 31 compare the string in the current generation with the current best string if the individual `p1`, as measured by `evaluate` is better than the individual `p2` then `p1` replaces `p2`. Line 32 mutates the current network, `p2`, using `mutate` (see Fig. 3.7). The `mutate` function changes the diameter of a randomly selected pipe in the network to a randomly selected diameter, chosen from a set of commercially available pipe diameters. The mutated individual, stored in the workplace `p1`, is used as the network configuration for the next iteration. Until the total number of generations is reached, the user selected information about the best individual is outputted by `dispResult` in line 34 of Fig. 3.5.

It should be noted that the algorithm described above can be used to design a simple WDS but is not optimal in terms of speed of convergence. Other EA's such as genetic algorithms (Simpson et al. 1994) will perform better. However the above example has the advantage of simplicity and contains all the basic elements that a GA would use when interacting with WDSLib.

```
1  #include "Simulation.h"
2  #include "result.h"
3  using namespace std;
4  #define GTN 100000                                                    //
       the total number of generations
5  /*available pipe diameters (inches)*/
6  vector<int> ADiameter
       ={36,48,60,72,84,96,108,120,132,144,156,168,180,192,204};
7  /*dollar per unit length*/
8  vector<int>unitCost
       ={93.6,133.7,176.3,221,267.6,315.8,365.4,416.5,468.7,
9                      522.1,576.6,632.1,688.5,745.1,804.1};
10 vector<int>  generateinitialDiameters(int);      // see fig. 6
11 vector<int>  mutate(vector<int>);                  // see fig. 7
12 double evaluate(Net*, Result*);                   // see fig. 8
13 int  main(int argc, char *argv[]){
14    srand (1);
15    char *config=argv[1];
16    Result *result=new Result();
17    Simulation *simulation1=new Simulation();        //initialize the
          simulation class
18    Net *net=simulation1->L1a(result,config);        //perform the
          L1a functions
19    vector<int> p1=generateinitialDiameter(net->getNp());//initial
          guesses of diameter
20    vector<int> p2;
          //work space for current best
21    double cost,currentbest;
22    for(int i=0;i<GTN;i++){
23       simulation1->setD(&p1,&ADiameter,net->getPIPE()->Diascale());
24       simulation1->solve(result);                                  //
             perform the L2 and L3 functions
25       simulation1->L1b(result);                                    //
             reverse the permutation
26       cost=evaluate(result,net,&p1);                          //evaluate
             the network cost
27       if(cost<currentbest||i==0){                                  //
             selection operator
28          currentbest=cost;
29          p2=p1;
30          cout<<i<<"\t"<<currentbest<<"\t"<<penaltyCost<<endl;
31       }
32       p1=mutate(&p2);
             //mutation operator
33    }
34    simulation1->dispResult(result);
35    delete simulation1;
36    delete result;
37    return 0;
38 }
```

Fig. 3.5. Example code for 1+1EA for Pipe Sizing

```
vector<int> generateinitialDiameters(int np)
{
   vector<int> Diameter = vector<int>(np);
   for (int i=0;i<np;i++)
      Diameter[i]=rand()%ADiameter.size();    //set the index for
         pipe diameters
   return Diameter;
}
```

Fig. 3.6. Implementation code for pipe size initialization

```
vector<int>  mutate(vector<int>* string){
   vector<int> string1;
   string1=*string;
   int aa=rand()%(string->size());  //choose which pipe to mutate
   int a=rand()%ADiameter.size();   //choose a pipe diameter after
      the mutation
   (string1)[aa]=a;                        //set the pipe index
   return string1;
}
```

Fig. 3.7. Implementation code for the mutation operator

```
double evaluate (Result* result,Net* net,vector<int> *p1) {
   PIPE* pipe =net->getPIPE();
   double np=net->getNp();
   double nj=net->getNj();
   double P=1e7;
   double cost1=0;
   penaltyCost=0;
   vector<double> hsol=result->getHsol();       //get the vector of
      nodal heads
   vector<double>* L=pipe->getL();                    //get the
      vector of pipe lengths
   vector<double>* zu=net->getNODE()->getZU();  //get the vector of
      nodal elevations
   double Lscale=pipe->Lenscale();                    //get the
      scaling factor for length
   for (int i = 0; i<np; i++)
      cost1+=(*L)[i]*Lscale*unitCost[(*p1)[i]]; //calculate the
         material cost
   for (int i = 0; i<nj; i++)
      if ((hsol)[i]-(*zu)[i]<minP)
         penaltyCost+=(minP-(hsol)[i]+(*zu)[i])*scaleP;//calculate
            penalty cost
   return cost1+penaltyCost;
}
```

Fig. 3.8. Implementation code for the evaluation function

This concludes the presentation of examples in this work. The next section presents a case study that illustrates the performance of WDSLib in a multi-simulation setting.

3.10 Case Study

The following presents timing results for WDSLib running the 1+1EA described in the previous section. The results below compare the four different solvers plus EPANET2. Note, that detailed timings for once-off simulations comparing the four methods can be found in Qiu et al. (2018). Three networks were benchmarked in these experiments. These were the N_1, N_3, and N_4 case-study networks used in Simpson et al. (2012). Table 3.7 summarizes the characteristics of these networks.

Table 3.7. Benchmark networks summary

	Full Network			Forest & Core Networks			Co-tree Network
Network	n_p	n_j	n_s	$n_f(n_f/n_p^{\#})$	n_{j_c}	n_{p_c}	n_{ct}
N_1	934	848	8	361 (38%)	573	487	84
N_3	1975	1770	4	823 (42%)	1152	947	205
N_4	2465	1890	3	429 (17%)	2036	1461	757

Table 3.8 shows the results of the 1+1EA from Fig. 3.5 for the GGA, GGA with FCPA, RCTM, RCTM with FCPA and the EPANET2 solvers. For each of the four WDSLib solvers above, the timings are given for running the EA with and without the L1 modules hoisted out the main EA loop. Each experiment evaluates the WDS network 100,000 times. And the best performing method for each network is highlighted in bold. It is important to note that 1+1EA using both the GGA and the WDSLib

Table 3.8. The actual 1+1 Evolutionary Algorithm run-time with 100,000 evaluations (min.) for each of the four solution methods applied to networks N_1, N_3, and N_4

	GGA	GGA with FCPA	RCTM	RCTM with FCPA	EPANET
	min.	min.	min.	min.	min.
N_1	6.73	4.64	4.53	**4.13**	9.81
N_3	15.21	**9.79**	13.75	10.30	26.43
N_4	21.14	**16.29**	23.92	21.93	67.11

The results show that the EA runs using WDSLib are substantially faster than the runs using the EPANET2 solver. This is, in part, due to the fact that the EPANET2 solver is designed as a standalone solver which does not facilitate lifting out of invariant computations from the EA loop.

As a demonstration of how the performance of an EA can be traced Fig. 3.9 shows the evolution of the fitness values of the N_1 network. These traces were extracted from a file written to in line 9 in Fig. 3.8. As can be seen, the cost and the pressure head violation terms drop during the EA run. Note that there will be considerable variation between 1+1EA runs due to its highly stochastic nature.

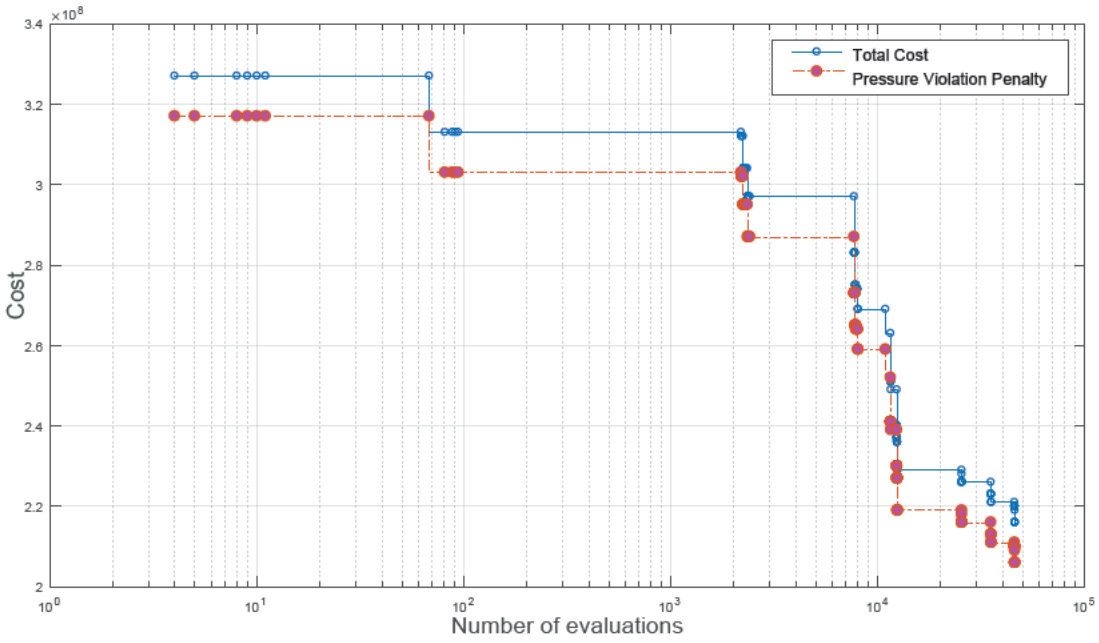


Fig. 3.9. The evaluation of the fitness value of network N_1

3.11 Conclusions

This paper has described WDSLib, a library for steady-state hydraulic simulation of WDS networks. WDSLib is fast, modular, and portable with implementation of several standard and recently published hydraulic solution methods. We have outlined the supported solution methodologies, the structure of the package and key aspects of WDSLib's implementation. Two example applications have been presented including a design case study using a simple EA. The EA results were benchmarked for different solvers demonstrating a substantial improvement in speed over the industry standard EPANET2 package. These benchmarks also have illustrated how the modular structure of WDSLib could be exploited to speed up execution time in a design setting.

As well as providing a fast simulation platform for both once-off and multi-run simulations, WDSLib also provides a testbed for comparing different solution methods in different settings and network topologies. As such WDSLib is designed with a pluggable architecture which can extended to efficiently incorporate new solution methods as they are created. This will enhance the capability of the research community to demonstrate the efficacy of new methods without having to re-engineer the content of shared WDSLib functions and data representations.

WDSLib accepts the input file format used by EPANET2 which allows for simple deployment. The amount of coding required to use the provided solvers in WDSLib is minimized by the use of a simple and extensible configuration file.

WDSLib is currently limited to finding solutions of demand-driven WDS systems without control devices. The development of software components to simulate control devices and pressure driven models is future work.

3.12 References

References are included in bibliography Chapter 7. In addition, the submitted paper is given in Appendix A.

3.13 Appendix A Scaling

In WDSlib, all input variables are scaled to a similar range to minimize the risk of avoidable computational inaccuracy or failures in floating point arithmetic.

The variables are scaled as following: $\widehat{\boldsymbol{G}} = \boldsymbol{G}/G_0$, $\widehat{\boldsymbol{q}} = \boldsymbol{q}/q_0$, $\widehat{\boldsymbol{h}} = \boldsymbol{h}/h_0$, $\widehat{\boldsymbol{e_l}} = \boldsymbol{e_l}/e_{l0}$, $\widehat{\boldsymbol{d}} = \boldsymbol{d}/d_0$ where $\boldsymbol{G}$, $\boldsymbol{h}$, $\boldsymbol{e_l}$, and $\boldsymbol{d}$ are the original input vectors, G_0, h_0, e_{l0}, and d_0 are the scaling factors and $\widehat{\boldsymbol{G}}$, $\widehat{\boldsymbol{h}}$, $\widehat{\boldsymbol{e_l}}$, and $\widehat{\boldsymbol{d}}$ are the scaled input vectors.

By substituting $\boldsymbol{G} = \widehat{\boldsymbol{G}}G_0$, $\boldsymbol{q} = \widehat{\boldsymbol{q}}q_0$, $\boldsymbol{h} = \widehat{\boldsymbol{h}}h_0$, $\boldsymbol{e_l} = \widehat{\boldsymbol{e_l}}e_{l0}$, $\boldsymbol{d} = \widehat{\boldsymbol{d}}d_0$, Eq. (3.1) and Eq. (3.2) become:

$$G_0 q_0 \widehat{\boldsymbol{G}}\widehat{\boldsymbol{q}} - h_0 \boldsymbol{A_1}\widehat{\boldsymbol{h}} - e_{l0}\boldsymbol{A_2}\widehat{\boldsymbol{e_l}} = 0 \tag{3.17}$$

$$q_0 \boldsymbol{A_1}^T \widehat{\boldsymbol{q}} - d_0 \widehat{\boldsymbol{d}} = 0 \tag{3.18}$$

Eq. (3.17) and Eq. (3.18) can be further simplified by introducing the notation: $a = h_0/\left(G_0 q_0\right)$, $b = e_{l0}/\left(G_0 q_0\right)$, and $c = d_0/q_0$:

$$\widehat{\boldsymbol{G}}\widehat{\boldsymbol{q}} - a\boldsymbol{A_1}\widehat{\boldsymbol{h}} - b\boldsymbol{A_2}\widehat{\boldsymbol{e_l}} = 0 \tag{3.19}$$

$$\boldsymbol{A_1}^T \widehat{\boldsymbol{q}} - c\widehat{\boldsymbol{d}} = 0 \tag{3.20}$$

with the following matrix form:

$$\begin{pmatrix} \widehat{\boldsymbol{G}} & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \widehat{\boldsymbol{q}} \\ a\widehat{\boldsymbol{h}} \end{pmatrix} - \begin{pmatrix} b\boldsymbol{A_2}\widehat{\boldsymbol{e_l}} \\ c\widehat{\boldsymbol{d}} \end{pmatrix} = 0. \tag{3.21}$$

Finally, the network can be solved by using Eq. (3.22) and Eq. (3.23):

$$\widehat{\boldsymbol{h}}^{(m+1)} = \frac{1}{a}\boldsymbol{U}^{-1}[-c\widehat{\boldsymbol{d}} - \boldsymbol{A_1}\widehat{\boldsymbol{F}}^{-1}[(\widehat{\boldsymbol{F}} - \widehat{\boldsymbol{G}})\widehat{\boldsymbol{q}}^{(m)} + b\boldsymbol{A_2}\widehat{\boldsymbol{e_l}}]] \tag{3.22}$$

$$\widehat{\boldsymbol{q}}^{(m+1)} = \widehat{\boldsymbol{q}}^{(m)} + a\widehat{\boldsymbol{F}}^{-1}\boldsymbol{A_1}\widehat{\boldsymbol{h}}^{m+1} - \widehat{\boldsymbol{F}}^{-1}(\widehat{\boldsymbol{G}}\widehat{\boldsymbol{q}}^{(m)} - b\boldsymbol{A_2}\widehat{\boldsymbol{e_l}}) \tag{3.23}$$

Choice of scaling factors

The choice of the scaling factor, despite much research, is not well understood. In this subsection, a choice for each scaling factor, based on the experience of the authors, is recommended. There are two types of variables and parameters that need to be scaled: invariants and variants.

Data sets that have very wide range of values can confound numerical accuracy. As a result, it may be preferable to scale the data to a narrower range. The default scaling factor for each of the input data is chosen to be its maximum absolute value. For example, the scaling factor for demand is max($\boldsymbol{d}$), so that its values range from zero to one.

In contrast, it is more difficult to choose a scaling factor a priori for values that vary between iterations (variants). This is because the range of variants can change as the iteration progresses. As a result, the intermediate and the final results might not be within the same range as the initial guesses.

There are two variants that need to be scaled: $\boldsymbol{q}$, $\boldsymbol{h}$. A good choice of the scaling factor for the flow rate is $\frac{\sum \boldsymbol{d}}{n_f}$ because the demand at each node must be satisfied by the water sources in the WDS and it is a reasonable assumption that the all demands are equally carried by each pipe that is directly connected to a water source and a good choice of the scaling factor for nodal head is $max(\boldsymbol{e_l})$ because the maximum nodal head cannot exceed the maximum elevation head of the fixed nodes.

During the process of the computation, the matrices $\boldsymbol{G}$ and $\boldsymbol{F}$ are scaled because their input variables are scaled. For the Darcy-Weisbach head loss model, the diagonal elements of the matrix $\boldsymbol{G}$ are modelled by:

$$[G]_{jj} = diag\left\{\left(\frac{8}{\pi^2 g}\right)\frac{L_j}{D_j^5} f_j |q_j|\right\} \text{ for } j = 1, \ldots, n_p$$

where the friction factor f is modelled by the Swamee-Jain formula:

$$f_j = \begin{cases} \dfrac{64}{\mathbb{R}_j} & \text{if } \mathbb{R}_j \leq 2000 \\ \sum_{k=0}^{3}(\alpha_k + \beta_k/\theta_j)(\mathbb{R}_j/2000)^k & \text{if } 2000 < \mathbb{R}_i < 4000 \\ \dfrac{0.25}{\log^2(\theta_j)} & \text{if } \mathbb{R}_j \geq 4000 \end{cases} \tag{3.24}$$

where $\theta_j = \dfrac{\epsilon_j}{3.7 D_j} + \dfrac{5.74}{\mathbb{R}_j^{0.9}}$ and $\mathbb{R}_j = \dfrac{4 q_j}{\pi \upsilon D_j}$ for $j = 1, \ldots, n_p$. Note that α_k and β_k are constant, the values of which can be found in Elhay and Simpson (2011)

In order to make sure the Reynolds number, a dimensionless variable, is not affected by scaling, $\widehat{\boldsymbol{\nu}} = \dfrac{D_0}{q_0}\boldsymbol{\nu}$ is introduced, Reynolds number $\mathbb{R}_j$ becomes $\dfrac{4\widehat{q}_j q_0}{\pi \widehat{\nu}(q_0/D_0)\widehat{D}_j D_0}$, which can be further simplified to $\dfrac{4\widehat{q}_j}{\pi \widehat{\nu} \widehat{D}_j}$, where the scaling factors are canceled.

In order to make sure f is also not affected by the scaling, $\widehat{\boldsymbol{\epsilon}} = D_0 \boldsymbol{\epsilon}$ is introduced, θ_j becomes

$$\theta_j = \frac{\widehat{\epsilon}_j D_0}{3.7 \widehat{D}_j D_0} + \frac{5.74}{\mathbb{R}_j^{0.9}} \text{ for } j = 1, \ldots, n_p$$

which can be further simplified to

$$\theta_j = \frac{\widehat{\epsilon}_j}{3.7 \widehat{D}_j} + \frac{5.74}{\mathbb{R}_j^{0.9}} \text{ for } j = 1, \ldots, n_p$$

where the scaling factors are canceled. It is evident that the friction factors remain the same because the values for the only two variables $\mathbb{R}$ and θ are unchanged.

Finally, diagonal elements of $\boldsymbol{G}$ can be expressed as: $G_0 \widehat{\boldsymbol{G}}$, where $G_0 = (\dfrac{8}{\pi^2 g})\dfrac{L_0}{D_0^5}|q_0|$ and $[\widehat{G}]_{jj} = diag\left\{\dfrac{\widehat{L}_j}{\widehat{D}_j^5} f_j |\widehat{q}_j|\right\}$ for $j = 1, \ldots, n_p$.

For the Hazen-Williams head loss model, the diagonal elements of the matrix $\boldsymbol{G}$ are modelled by: $[G]_{jj} = diag\left\{\dfrac{10.67 L_j}{C_j^{1.852} D_j^{4.871}}|q_j|^{(n-1)}\right\}$ for $j = 1, \ldots, n_p$, where C_j

is the Hazen-Williams coefficient for the j-th pipe. The Hazen-Williams coefficient, unlike the friction factor in the Darcy-Weisbach head loss model, is independent of flow rate, pipe wall condition, and flow regimes, which means it is an independent variable. As a result, the scaling factor for Hazen-Williams $\boldsymbol{G}$ can simply be derived by: $[G]_{jj} = diag\left\{\frac{10.67\widehat{L}_j L_0}{\widehat{C}_j^{1.852} C_0^{1.852} \widehat{D}_j^{4.871} D_0^{4.871}} |\widehat{q}_j|^{(n-1)} |q_0|^{(n-1)}\right\}$ and the equation for diagonal elements of $\boldsymbol{G}$ for Hazen-Williams equation can be expressed as: $\boldsymbol{G} = diag\left\{G_0 \widehat{\boldsymbol{G}}\right\}$, where $G_0 = \frac{10.67 L_0}{C_0^{1.852} D_0^{4.871}} |q_0|^{(n-1)}$ and $[G]_{jj} = diag\left\{\frac{\widehat{L}_j}{\widehat{C}_j^{1.852} \widehat{D}_j^{4.871}} |\widehat{q}_j|^{(n-1)}\right\}$ for $j = 1, \ldots, n_p$.

3.14 Appendix B Forest Search Algorithm

Algorithm 1: Forest Search Algorithm

input : Adjacency list, $\boldsymbol{d}$ and Deg
output : Forest, RootNode, $\boldsymbol{q}$ and $\boldsymbol{d}$

```
1  k ← 1;
2  for i ← 1 to n_j do
3      n = i;
4      while Deg(n) == 1 do
5          for (Adj_p,Adj_v) ∈ Adj(n) do
6              if Deg(Adj_v) == 1 then
7                  Forest[k] ←Union(Forest[k], (Adj_p, n))
8              else
9                  q(Adj_p) ← d(n);
10                 d(Adj_v) ← d(Adj_v) + d(n);
11                 Deg(Adj_v) ← Deg(Adj_v) − 1;
12                 n = Adj_v;
13                 if Deg[n] ≥ 2 then
14                     RootNode[k] = n;
15                     k ← k + 1;
16                 end if
17             end if
18         end for
19     end while
20 end for
```

3.15 Appendix C Spanning Tree Search Algorithm

Algorithm 2: Spanning Tree Search Algorithm

input : adjList
output : Spanning Tree and Co-Tree elements

$STP \leftarrow \{\}$; // initialize an empty vector for spanning tree pipes
$STN \leftarrow \{\}$; // initialize an empty vector for spanning tree nodes
$CTP \leftarrow \{\}$; // initialize an empty vector for co-tree pipes
$VN \leftarrow \{\}$; // initialize a boolean vector for visited nodes
$VP \leftarrow \{\}$; // initialize a boolean vector for visited pipes
$SN \leftarrow \{\}$; // initialize an empty set of searching nodes
for $i \leftarrow n_j$ **to** n_f **do**
    $VN[i]$ = $true$; // mark the source i as visited
    $SN \leftarrow i$; // insert this node into the search node vector
end for
while *SN is not empty* **do**
    $i \leftarrow$ the first element in SN;
    foreach *(*Adj_p*,*Adj_v*)* $\in Adj[i]$ **do**
        if *VN[*Adj_v*] == false* **then**
            $STP \leftarrow Adj_p$; // mark pipe as a spanning tree pipe
            $STN \leftarrow Adj_v$; // mark node as a corresponding spanning tree pipe
            $SN \leftarrow Adj_v$;
            VN[Adj_v]=true;
            VP=[Adj_p]=true;
        end if
        else if *VN[*Adj_p*]==false* **then**
            $CTP \leftarrow Adj_p$; // removed pipe j as a singleton pipe
            VP[Adj_p]=true;
        end if
    end foreach
    remove i from SN;
end while

3.16 Appendix D Complete configuration files

```
[InputFile]
<directory>        %REQUIRED INFORMATION the input file directory
WDSnetwork/
<file>             %REQUIRED INFORMATION the input file name
n1.inp
<end>              %end of input file
%--------------------------------------------------%
[Parameter]
<maxIter>          %maximum number of iteration
50
<initV>            %initial veloc ity
0.3048          %1ft/s
<StopTol>          %stopping tolerance used
1.000e-6
<NormTyp>       %norm type 1 for 1-norm, 2 for 2-norm, 3 for inf-
   norm
3.0
<StopTest>      %stopping test used 1 for q-norm, 2 for h-norm 3 for
    q&h-norm
1
<SerP>             %service pressure
20.0
<MinP>             %Minimum pressure
40.0
<Demandfuc>        %type of consumption function
1.00
<MaxNpIterResult> %np treshold for disping iterates result
50.0
<MinImproTol>
1.0000e-3
<end>        %end of parameter
%--------------------------------------------------%
[dispFlag]
<BasicFlag>
false
<NetInfoFlag>
false
<ConvergenceFlag>
false
<StatFlag>
false
<ScalingFlag>
0
<NodalResultflag>
0
<LinkResultflag>
false
<QitersFlag>
false
<HitersFlag>
false
<timeFlag>
false
<end>              %end of dispFlag
%--------------------------------------------------%
[controlFlag]
<SolverFlag>       %1 for GGA 2 for RCTM 3 for SMPA 4 for PDM
1
<FCPAFlag>
0
<end>        %end of controlFlag
%--------------------------------------------------%
```

Fig. 3.10. A configuration file to run the RCTM in WDSLib

3.17 Appendix E Nomenclature

Acronyms

CT Co-tree
CTP Co-tree pipes
DW Darcy-Weisbach head loss formula
FCPA Forest-core partitioning algorithm
GGA Global gradient algorithm
HW Hazen-William head loss formula
WDS Water distribution system
RCTM Reformulated co-tree flows method
nnz Number of non-zeros
STP Spanning tree pipes
STN Spanning tree nodes
ST Spanning tree

Constants

g Gravitational acceleration constant
ν Kinematic viscosity of water

FCPA variables

$\boldsymbol{C}$ Permutation matrix for the nodes in the core, E_c, of the network
E_c Set of core pipes (edges) in G_c
E_f Set of forest pipes (edges) in G_f
G_c Core subgraph
G_f Forest subgraph
$\boldsymbol{P}$ Permutation matrix for the pipes in the core, E_c, of the network
$\boldsymbol{S}$ Permutation matrix for the pipes in the forest, E_f,of the network
$\boldsymbol{T}$ Permutation matrix for the nodes in the forest, E_f, of the network
V_c Set of core nodes (vertices) in G_c
V_f Set of forest nodes (vertices) in G_f

Hydraulic variables for GGA

$\boldsymbol{A_1}$ Unknown-head node-arc incidence matrix
$\boldsymbol{A_2}$ Fixed-head node-arc incidence matrix
$\boldsymbol{d}$ Vector of nodal demands
$\boldsymbol{d}_i$ Demand of node i
$\boldsymbol{D}$ Vector of pipe diameters
$\boldsymbol{D}_j$ Diameter of pipe j
$\boldsymbol{e_l}$ Vector of Fixed-head nodes elevation heads
$\boldsymbol{e_l}_k$ Fixed-head nodes elevation heads at node k
E Set of pipes in graph G
$\boldsymbol{EN}$ Vector of end nodes
$\boldsymbol{EN}_j$ End nodes of pipe j
$\boldsymbol{f}$ Vector of Darcy-Weisbach friction factors
$\boldsymbol{f}_j$ Darcy-Weisbach friction factor of pipe j
$\boldsymbol{F}$ Diagonal Matrix of generalized headloss derivatives when the headloss is modelled by either the HW and the DW
$[F]_{jj}$ Generalized headloss derivatives for pipe j
G Full WDS graph
$\boldsymbol{G}$ Diagonal matrix with elements $r_j|q_j|^{n-1}$

$[G]_{jj}$ $r_j|q_j|^{n-1}$
$\boldsymbol{h}$ Vectors of unknown heads
$\boldsymbol{h}_i$ Heads at node i
$\boldsymbol{J}$ Jacobian matrix
L_{1a} Functions run once before multiple simulation
L_{2a} Functions run once before hydraulic assessment
L_3 Functions run every iteration
L_{2b} Functions run once after hydraulic assessment
L_{1b} Functions run once after multiple simulation
$\boldsymbol{L}$ Vector of pipe lengths
$\boldsymbol{L}_j$ Length of pipe j
n Head loss exponent
n_f Number of forest pipes and nodes
n_j Number of junctions
n_p Number of pipes
n_r Number of fixed-head nodes
n_{st} Number of ST pipes and nodes
n_{ct} Number of CT pipes
$\boldsymbol{q}$ Vector of unknown flows
$\boldsymbol{q}_j$ Flow in pipe j
$\mathbb{R}$ Vector of Reynolds numbers
$\mathbb{R}_j$ Reynolds number for pipe j
$\boldsymbol{SN}$ Vector of start nodes
$\boldsymbol{SN}_j$ Start nodes of pipe j
$\boldsymbol{U}$ Diagonal matrix of Schur Complement when headloss is modelled by HW
$\boldsymbol{V}$ Generalized Schur Complement when the headloss is modelled by both the HW and the DW
V Set of node in graph G
z_i Elevation at node i
α_k Interpolating spline coefficient
β_k Interpolating spline coefficient
$\boldsymbol{\theta}$ Vector as defined in Eq. (3.24)
$\boldsymbol{\epsilon}$ Vector of pipe roughness heights
$\boldsymbol{\epsilon}_j$ Roughness height for pipe j

RCTM variables

E_{st} Set of ST pipes (edges)
E_{ct} Set of complementary CT pipes (edges)
$\boldsymbol{K_1}$ Orthogonal permutation matrix for pipes in the ST
$\boldsymbol{K_2}$ Orthogonal permutation matrix for pipes in the CT
$\boldsymbol{L_{21}}$ A part of a basis for the null space of the permuted node-arc incidence for the RCTM
$\boldsymbol{R}$ Orthogonal permutation matrix for nodes in the ST
V_{st} Set of ST nodes (vertices)
$\boldsymbol{W}$ Schur complement for the RCTM

Chapter 4

Publication 2: A Benchmarking Study of Water Distribution System Solution Methods

4.1 Synopsis

In Chapter 3, WDSLib, a numerically robust, efficient and accurate C++ library, was developed for steady-state hydraulic simulation of WDS networks. However, different solution methods exploit different properties of the WDS network. For example, the FCPA exploits the linear forest component of the network, while the RCTM exploits the relationship between the flows in the spanning tree pipes and the flows in the co-tree pipes. The relative performance of solvers in terms of speed, rate-of-convergence, and accuracy depends, amongst other things, on the topology of the target network: the size of the forest component, the number of network loops and the density of these network loops. It is difficult to evaluate the impact of these topology factors by only examining the incidence matrix that describes the pipe network connectivity. As a result, the best method to use for a particular network cannot be easily determined a priori.

In this research, efficient implementations of four solution methods, the global gradient algorithm (GGA), the GGA with the forest-core partitioning algorithm (FCPA), reformulated co-tree flow method (RCTM), and the RCTM with the FCPA, are compared on eight case study benchmark networks containing between 934 and 19647 pipes and between 848 and 17971 nodes. These simulations were carried out under both a once-off simulation setting and a multiple simulation setting.

An important objective of this research is to ensure a fair comparison between solution methods. To achieve this, a unified framework has been developed for the mathematical formulations of four solution methods, the GGA with and without FCPA, RCTM with and without FCPA, for WDSs. Each of the solution methods is presented in terms of pure orthogonal permutations of the system of equations to minimise the intermediate steps to ensure a fair comparison between the solution methods.

4.1.1 Citation

Qiu, M, Simpson, AR, Elhay, S & Alexander, B 2018, 'A Benchmarking Study of Water Distribution System Solution Methods', Manuscript submitted for publication to *Journal of Water Resources Planning and Management*.

Statement of Authorship

Title of Paper	A Benchmarking Study of Water Distribution System Solution Methods
Publication Status	☐ Published ☐ Accepted for Publication ☑ Submitted for Publication ☐ Unpublished and Unsubmitted work written in manuscript style
Publication Details	Qiu, M, Simpson, AR, Elhay, S & Alexander, B 2018a, 'A Benchmarking Study of Water Distribution System Solution Methods', Manuscript submitted for publication to Journal of Water Resources Planning and Management.

Principal Author

Name of Principal Author (Candidate)	Mengning Qiu
Contribution to the Paper	Developed the software package, conducted numerical analysis and prepared manuscript.
Overall percentage (%)	75%
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.
Signature	Date: 30/04/2018

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Sylvan Elhay.
Contribution to the Paper	Helped in data interpretation, model development, manuscript evaluation and manuscript editing. (15%)
Signature	Date: 30-Apr-2018

Name of Co-Author	Angus R. Simpson
Contribution to the Paper	Provided manuscript evaluation and manuscript editing. (5%)
Signature	Date: 30 April 2018

Name of Co-Author	Bradley Alexander		
Contribution to the Paper	Provided manuscript evaluation and manuscript editing. (5%)		
Signature		Date	1/05/18.

4.2 Abstract

In recent years a number of new WDS solution methods have been developed. These methods have been aimed at improving the speed and reliability of WDS simulations. However, to date, these methods have not been benchmarked against each other in a reliable way. This research addresses this problem by using a newly developed software platform, **WDSLib**, as a fair basis for a detailed comparison of the performance of these methods under different settings. In this work, efficient implementations of three solution methods, the Global Gradient Algorithm (GGA), the forest-core partitioning algorithm (FCPA), and the reformulated co-tree flow method (RCTM), and combinations of these, are compared on eight case study benchmark networks containing between 934 and 19647 pipes and between 848 and 17971 nodes. These simulations were carried out under both a once-off simulation setting and a multiple simulation setting (such as occurs in a genetic algorithm). Timings for these benchmark runs are decomposed into stages so that the performance of these methods can be easily estimated for different settings. The results of this study will help inform the choice of solution methods for given combinations of network features and given design settings. In addition, timing results are compared with EPANET2.

4.2.1 Keywords

water distribution systems solution; Forest-Core Partitioning Algorithm; Global Gradient Algorithm; Reformulated Co-tree Flow Method; hydraulic analysis; EPANET.

4.3 Introduction

Water Distribution Systems (WDSs) are frequently modeled by a system of nonlinear equations, the steady-state solutions of which, the flows and heads in the system, are used in WDS design, management and operation. In a design setting, the solutions might be used as part of an optimization problem to determine the best choices of some network parameters such as pipe diameters. In a management setting, the solutions might be used for the calibration of network parameters such as demand patterns. In an operational environment, new solutions might be needed to adjust control device settings whenever new supervisory control and data acquisition (SCADA) information becomes available.

The most widely used WDS simulation method in current use is the Global Gradient Algorithm (GGA) (Todini and Pilati 1988), which solves the non-linear system of equations representing the WDS. The GGA and its implementations exhibit excellent convergence characteristics for a wide range of starting values and a wide variety of WDS problems. However, some networks have structural properties which can be exploited to further improve the efficiency of the solution process. The GGA, a range space method, exploits the block structure of the full Jacobian matrix in order to produce a smaller key matrix in the linearization of the Newton method. The reformulated co-tree flows method (RCTM) (Elhay et al. 2014), a null-space method (Benzi et al. 2005), can further exploit the block structure of the Jacobian matrix to produce, in realistic WDSs, an even smaller key matrix. This is achieved by dealing separately with the spanning tree and the co-tree in the Newton method linearization.

Another avenue for reducing computation can be exploited by using the Forest-Core Partitioning Algorithm (FCPA) (Simpson et al. 2012) to separate the problem into its linear and non-linear components. The observation underpinning the FCPA is that most

WDSs have trees, the collections of which are called forests. The complement of the forest in a network is called the core. The flows in a forest can be computed a-priori by a linear process. Hence, the dimension of the key matrices in the solution process can be significantly reduced when the forest is a large part of the network.

With the development of different solution methods, WDS simulation package users are faced with a choice of which solution method or methods to apply. Previous publications performed case studies comparing the performances of their respective methods to the GGA. However, these comparisons were often done using different implementation languages, and different levels of code optimization – which makes cross-comparison of methods difficult. Consequently, there is a need for a study which reliably compares the relative performance of these methods using a fast, carefully designed code implementation. To this end, this work presents a thorough benchmark study to compare the performance of GGA, GGA-with-FCPA, RCTM, and RCTM-with-FCPA for a range of case study networks using a fast C++ implementation. The timings for these runs are decomposed according to how often each solution component is executed in different simulation settings. From these timings it is possible to accurately predict runtimes for long-run multiple simulation settings. To confirm the relevance of these results, the timings have been compared with the speed of the industrial and research standard toolkit of EPANET2 (Rossman 2000) and was found to be faster in all cases.

This paper is organized as followed. A detailed review of existing solution methods is given in the next section. The section following presents the mathematical formulation of each method. The motivation for a benchmark study is then given, followed by the methodology used in this paper to carry out a benchmark study. The description of the module categorization is then presented. This is followed by a case study of the four solution methods applied to the eight case study networks. The results are discussed in the next section. The last section offers some conclusions.

4.4 Literature Review

This section provides a review of the algorithms that are tested in this paper. A brief development history of WDS solution algorithms is presented in the first subsection. The next subsection gives an overview of the GGA and its development, followed by an overview of solution methods which use the null space approach (such as co-trees flow method (CTM) and RCTM). Finally, a review of the methods that use graph theory to simplify problem complexity are presented.

4.4.1 Development history of the WDS algorithms

This research considers a water distribution model made up of energy conservation equations and the demand driven model continuity equations. The Hardy Cross method (Cross 1936), also known as the loop flow corrections method, is one of the oldest methods and uses successive approximations, solving for each loop flow correction independently. It is a method that was widely used for its simplicity at the time when it was introduced. More than three decades later, Epp and Fowler (1970) developed a computer version of Cross's method and replaced the numerical solver with the Newton method, which solves for all loop flow corrections simultaneously. However, this method has not been widely used because of the need (i) to identify the network loops, (ii) to find initial flows that satisfy continuity equation and (iii) to use pseudo-loops.

Many methods have been proposed to improve the computational efficiency of the WDS model. These include: matrix decomposition (Todini and Pilati 1988; Elhay et al. 2014; Deuerlein et al. 2015), graph partitioning (Rahal 1995; Simpson et al. 2012; Alvarruiz et al. 2015), network skeletonization (Saldarriaga et al. 2008; Shamir and Salomons 2008), and network clustering (Anderson and Al-Jamal 1995; Perelman and Ostfeld 2011). Both network skeletonization and network clustering can produce a smaller network to solve. However, they are not considered in this study because the solutions from both methods are approximations to the solutions for the full networks, unlike the *exact solutions* produced by the methods used in this study. A summary of methods that improve the computational efficiency of the steady-state demand-driven WDS solution process follows.

4.4.2 Global gradient algorithm

The GGA is a range space method that solves for both flows and heads. It was the first algorithm, in the field of hydraulics, to exploit the block structure of the Jacobian matrix to reduce the size of the key matrix in the linearization of the Newton method. The GGA has gained popularity through its rapid convergence rate for a wide range of starting values. This is the result of using the Newton method on an optimization problem that has a quadratic surface. However, it was reported by Elhay and Simpson (2011) that the GGA fails catastrophically in the presence of zero flows in a WDS when the head loss is modeled by the Hazen-Williams formula. Regularization methods have been proposed by both Elhay and Simpson (2011) and Gorev et al. (2012) to deal with zero flows when the head loss is modeled by the Hazen-Williams formula.

The GGA as it was first proposed, applied only for the WDSs in which the head loss is modeled by the Hazen-Williams formula, where the resistance factor was independent of flow. In EPANET2, Rossman (2000) extended the GGA to allow the use of the Darcy-Weisbach formula. It has been pointed out in Simpson and Elhay (2010), however, that Rossman incorrectly treated the Darcy-Weisbach resistance factor as independent of the flow. They introduced the correct Jacobian matrix to deal with this. It has been demonstrated that once the correct Jacobian matrix is used, the quadratic convergence rate of the Newton method is restored. Furthermore, Elhay and Simpson (2011) reported that the GGA no longer fails in the presence of zero flows when the derivative of the Jacobian matrix is correctly computed with the Darcy-Weisbach formula.

4.4.3 Null space method

The co-trees flow method (CTM) (Rahal 1995) is a null space method that solves for the co-tree flows and spanning tree flows separately. The CTM, unlike the loop flow corrections method, does not require the initial flows to satisfy continuity. However, it does require: (i) the identification of the associated circulating graph; (ii) the determination of the demands that are to be carried by tree branches; (iii) finding the associated chain of branches closing a circuit for each co-tree chord; (iv) computing pseudo-link head losses. The RCTM (Elhay et al. 2014) is also a null space method that solves co-tree flows and spanning trees flows separately. It represents a significant improvement on the CTM by removing requirements (i) to (iv) above. It uses the Schilders' factorization (Schilders 2009) to permute the node-arc incidence matrix into an invertible spanning tree block and a co-tree block. This permutation reduces the size of the Jacobian matrix from the number of junctions (as in the GGA) to the approximate number of loops in the network.

Abraham and Stoianov (2015) proposed a novel idea to speed-up the solution process when using a null space method to solve a WDS network. Their idea exploits the fact that a significant proportion of runtime is spent computing the head losses. At the same time, flows within some pipes exhibit negligible changes after a few iterations. As a result, there is no point in wasting computer resources to re-compute the pipe head losses for the pipes that have little or no change in flows. This partial update can be used to economise the computational complexity of the GGA, the RCTM and their variations.

4.4.4 Graph theory

The forest-core partitioning algorithm (FCPA) (Simpson et al. 2012) speeds up the solution process. This algorithm permutes the system equations to partition the linear component of the problem, which is the forest of the WDS, from the non-linear component, which is the core of the WDS. It can be viewed as a method that simplifies the problem by solving for the flows and the heads in the forest just once instead of at every iteration. The FCPA reduces the number of pipes, number of junctions, and the dimension of the Jacobian matrix in the core by the number of forest pipes (or nodes).

The graph matrix partitioning algorithm(GMPA) (Deuerlein et al. 2015) exploited the linear relationships between flows of the internal trees and the flows of the corresponding super-links after the forest of the network has been removed. This was a major breakthrough. The GMPA permutes the node-arc incidence matrix in such a way that all of the nodes with degree two in the core can be treated as a group. By partitioning the network this way, the network can be solved by a global step, which solves for the nodes with degree greater than two (super nodes) and the pipes which connect to them (path chords), and a local step, which solves for the nodes with degree two (interior path nodes) and pipes connected to them (path-tree links).

In a recent paper by Elhay et al. (2018), they proposed a single framework for both the FCPA and GMPA and extended the methods applicability to the pressure dependent problem. Although the flows and heads in the forest component of a pressure driven WDS cannot be determined by a linear process, they can be solved by a similar linear iterative process as the local step in the GMPA.

4.5 Motivation

Thus far, this paper has discussed the recent developments in the solution methods. Previous work on WDS simulation has focused on two research areas: (i) hydraulic solution methods (Nielsen 1989; Simpson et al. 2012; Elhay et al. 2014; Deuerlein et al. 2015) and (ii) solver software design (Morley et al. 2000; van Zyl et al. 2003; Guidolin et al. 2010). Two observations can be made when comparing these two areas of research focus:

(i) Different platforms have been used to compare algorithm implementations. Some methods have been compared against EPANET, some methods have been implemented by using parts of the EPANET toolkit, some methods have been benchmarked using MATLAB, and others use purpose-written C or C++ code. Comparing timing results between all of these different platforms is especially difficult because MATLAB is a modeling programming language which is not necessarily intended to produce fast production code. As a consequence, the execution of MATLAB code will typically be slower than carefully written C++ code even if the solution method implemented in MATLAB is potentially faster. This will later be discussed in detail.

(ii) Timing results can be accurately extrapolated to different design settings only if the implementation code is sufficiently modularized and the timings are available for each module.

To address the problems described above, this work describes the methodology employed to ensure a fair comparison between solution methods.

4.6 Network Formulations

This section provides an unified framework for the mathematical formulations of four solution methods, the GGA with and without FCPA, RCTM with and without FCPA, for WDSs. Each of the solution methods is presented in terms of pure orthogonal permutation of the system of equations to minimize the intermediate steps to ensure a fair comparison between the solution methods. The following starts with the basic definitions and notation, followed by the system equations. The next subsection focuses on the use of network partitioning methods to speed up the solution process for WDSs. Finally, the equations for different solution methods are shown.

4.6.1 Definitions and Notation

Consider a water distribution system that contains n_p pipes, n_j junctions, n_r fixed head nodes, n_f forest pipes and nodes, n_{p_c} pipes in the core network, n_{j_c} nodes in the core network, n_{ct} pipes in the co-tree network and n_{st} pipes in the spanning tree network. The $j-th$ pipe of the network can be characterized by its diameter D_j, length L_j, resistance factor r_j. The $i-th$ node of the network has two properties: its nodal demand d_i and its elevation z_{u_i}. Let $\boldsymbol{q} = \left(q_1, q_2, \ldots, q_{n_p}\right)^T$ denote the vector of unknown flows, $\boldsymbol{h} = \left(h_1, h_2, ...h_{n_j}\right)^T$ denote the vector of unknown heads, $\boldsymbol{r} = \left(r_1, r_2, ...r_{n_p}\right)^T$ denote the vector of resistance factors, $\boldsymbol{d} = \left(d_1, d_2, ...d_{n_j}\right)^T$ denote the vector of nodal demands, $\boldsymbol{e_l} = \left(e_{l_1}, e_{l_2}....e_{l_{n_f}}\right)^T$ denote the vector of fixed head elevations.

The head loss exponent n is assumed to be $n = 2$ for the Darcy-Weisbach head loss model and $n = 1.852$ for Hazen-Williams head loss model. The head loss within the pipe j, which connects the node i and the node k, is modeled by $h_i - h_k = r_j q_j |q_j|^{n-1}$. Denote by $\boldsymbol{G}(\boldsymbol{q}) \in \mathbb{R}^{n_p \times n_p}$, a diagonal square matrix with elements $[\boldsymbol{G}]_{ii} = r_i |q_i|^{n-1}$ for $i = 1, 2, ...n_p$. Denote by $\boldsymbol{F}(\boldsymbol{q}) \in \mathbb{R}^{n_p \times n_p}$, a diagonal square matrix where the k-th element on its diagonal $[\boldsymbol{F}]_{kk} = \frac{d}{dq_k}[\boldsymbol{G}]_{kk}\, q_k$. $\boldsymbol{A_1}$ is the full rank, unknown head, node-arc incidence matrix, where $[\boldsymbol{A_1}]_{ij}$ is used to represent the relationship between pipe i and node j; $[\boldsymbol{A_1}]_{ij} = -1$ if pipe i enters node j, $[\boldsymbol{A_1}]_{ij} = 1$ if pipe i leaves node j, and $[\boldsymbol{A_1}]_{ij} = 0$ if pipe i is not connected to node j. $\boldsymbol{A_2}$ is the fixed-head node-arc incidence matrix, where $[\boldsymbol{A_2}]_{ij}$ is used to represent the relationship between pipe i and fixed head node j, $[\boldsymbol{A_2}]_{ij} = -1$ if pipe i enters fixed head node j, $[\boldsymbol{A_2}]_{ij} = 1$ if pipe i leaves fixed head node j, and $[\boldsymbol{A_2}]_{ij} = 0$ if pipe i is not connected to fixed head node j.

Denote by E_f, the set of indices of the pipes in the forest; by V_f, the set of indices of the nodes in the forest; by E_c, the set of indices of the pipes in the core; and by V_c, the set of indices of the nodes in the core. Denote by E_{st}, the set of indices of the pipes in the spanning tree; by V_{st}, the node indices that correspond with the spanning tree pipes; and by E_{ct}, a set of indices of the pipes in the co-tree.

4.6.2 System of Equations

There are two primary equations that model the underlying relationship of the flows and the heads of a WDS: the demand-driven model (DDM) continuity equations (4.1) and the energy conservation equations (4.2):

$$-\boldsymbol{A_1}^T \boldsymbol{q} - \boldsymbol{d} = \boldsymbol{O} \tag{4.1}$$

$$\boldsymbol{G}(\boldsymbol{q})\,\boldsymbol{q} - \boldsymbol{A_1}\boldsymbol{h} - \boldsymbol{A_2}\boldsymbol{e_l} = \boldsymbol{O}. \tag{4.2}$$

This non-linear system of equations can be expressed in matrix form as

$$\begin{pmatrix} \boldsymbol{G}(\boldsymbol{q}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{h} \end{pmatrix} - \begin{pmatrix} \boldsymbol{A_2}\boldsymbol{e_l} \\ \boldsymbol{d} \end{pmatrix} = 0 \tag{4.3}$$

and it is sometimes referred to as a nonlinear saddle point problem (Benzi et al. 2005).

This non-linear system is normally solved by the Newton method, in which $\boldsymbol{q}^{(m+1)}$, a vector of flows at (m+1)-th iteration, and $\boldsymbol{h}^{(m+1)}$, a vector of heads at (m+1)-th iteration, are repeatedly computed from $\boldsymbol{q}^{(m)}$, a vector of flows at (m)-th iteration, and $\boldsymbol{h}^{(m)}$, a vector of heads at (m)-th iteration, by

$$\begin{pmatrix} \boldsymbol{F}^{(m)}(\boldsymbol{q}^{(m)}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}^{(m+1)} - \boldsymbol{q}^{(m)} \\ \boldsymbol{h}^{(m+1)} - \boldsymbol{h}^{(m)} \end{pmatrix} = - \begin{pmatrix} \boldsymbol{G}^{(m)}\boldsymbol{q}^{(m)} - \boldsymbol{A_1}\boldsymbol{h}^{(m)} - \boldsymbol{A_2}\boldsymbol{e_l} \\ -\boldsymbol{A_1}^T\boldsymbol{q}^{(m)} - \boldsymbol{d} \end{pmatrix} \tag{4.4}$$

until the differences $(\boldsymbol{q}^{(m+1)} - \boldsymbol{q}^{(m)})$ and $(\boldsymbol{h}^{(m+1)} - \boldsymbol{h}^{(m)})$ are sufficiently small.

The Global Gradient Algorithm

The GGA takes advantage of the block structure of Eq. (4.3) to obtain a two-step solver: Eq. (4.5) for the heads and Eq. (4.6) for the flows when the head-loss is modeled by Hazen-William equation.

$$\boldsymbol{U}\boldsymbol{h}^{(m+1)} = -n\boldsymbol{d} + \boldsymbol{A_1}^T \left[(1-n)\,\boldsymbol{q}^{(m)} - \boldsymbol{G}^{-1}\boldsymbol{A_2}\boldsymbol{e_l}\right] \tag{4.5}$$

where $\boldsymbol{U} = \boldsymbol{A_1}^T\boldsymbol{G}^{-1}\boldsymbol{A_1}$

$$\boldsymbol{q}^{(m+1)} = \frac{1}{n}\{(n-1)\,\boldsymbol{q}^{(m)} + \boldsymbol{G}^{-1}(\boldsymbol{A_2}\boldsymbol{e_l} + \boldsymbol{A_1}\boldsymbol{h})\}. \tag{4.6}$$

Later, Simpson and Elhay (2010) proposed

$$\boldsymbol{V}\boldsymbol{h}^{(m+1)} = -\boldsymbol{d} + \boldsymbol{A_1}^T\boldsymbol{F}^{-1}\left[(\boldsymbol{G}-\boldsymbol{F})\,\boldsymbol{q}^{(m)} - \boldsymbol{A_2}\boldsymbol{e_l}\right] \tag{4.7}$$

where $\boldsymbol{V} = \boldsymbol{A_1}^T\boldsymbol{F}^{-1}\boldsymbol{A_1}$

$$\boldsymbol{q}^{(m+1)} = \boldsymbol{q}^{(m)} + \boldsymbol{F}^{-1}\boldsymbol{A_1}\boldsymbol{h}^{(m+1)} - \boldsymbol{F}^{-1}\left[\boldsymbol{G}\boldsymbol{q}^{(m)} - \boldsymbol{A_2}\boldsymbol{e_l}\right] \tag{4.8}$$

as the generalized equations that can be applied when the head-loss is modeled by the Hazen-William equation or the Darcy-Weisbach equation. The correct Jacobian matrix with the formula for $\boldsymbol{F}$, when head loss is modeled by Darcy-Weisbach equation, can be found in Simpson and Elhay (2010). They showed that the use of the correct Jacobian matrix restores the quadratic rate of convergence.

4.6.3 Network Partitioning

Associated with a WDS is a graph G=(V, E), where the elements of V are the nodes (vertices) of the graph G and elements of E are the pipes (links) of the graph G. In this subsection, the permutation of the system equations (4.3) for the FCPA is introduced, followed by a description of the RCTM, which further exploits the block structure of the Jacobian matrix.

Forest-Core Partitioning Algorithm

In a demand-driven model, it is possible to exploit the fact that every WDS can be divided into two subgraphs: a treed subgraph (forest) $G_f = \left(V_f, E_f\right)$ and a looped subgraph (core) $G_c = (V_c, E_c)$, so that $E_f \cup E_C = E$, $E_f \cap E_C = \emptyset$, $V_f \cup V_C = V$. All flows and heads in both the forest and the core must be found. The flows in the forest can be found by a linear process before the iterative solution phase and the heads in the forest can be found linearly after the iterative phase.

Simpson et al. (2012) proposed the FCPA, which partitions the network into a treed component and a looped component (referred to as the core) thereby reducing the computation time where the network has a significant forest component. The FCPA starts by generating a permutation matrix

$$\boldsymbol{P_1} = \begin{array}{c} \\ n_f \\ n_{p_c} \\ n_{j_c} \\ n_f \end{array} \begin{array}{c} \begin{array}{cc} n_p & n_j \end{array} \\ \begin{pmatrix} \boldsymbol{S} & \boldsymbol{O} \\ \boldsymbol{P} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{C} \\ \boldsymbol{O} & \boldsymbol{T} \end{pmatrix} \end{array} \tag{4.9}$$

, where $\begin{bmatrix} \boldsymbol{S} \\ \boldsymbol{P} \end{bmatrix} \in \mathbb{R}^{n_p \times n_p}$ is the square orthogonal permutation matrix for the pipes, $\boldsymbol{S} \in \mathbb{R}^{n_f \times n_p}$ is the permutation matrix which identifies the pipes in the forest as distinct from those of the core of the WDS, $\boldsymbol{P} \in \mathbb{R}^{n_{pc} \times n_p}$ is the permutation matrix for the pipes in the core of the WDS, $\begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{T} \end{bmatrix} \in \mathbb{R}^{n_j \times n_j}$ is the square orthogonal permutation matrix for the nodes, $\boldsymbol{C} \in \mathbb{R}^{n_{j_c} \times n_j}$ is the permutation matrix for the nodes in the core of the WDS, $\boldsymbol{T} \in \mathbb{R}^{n_f \times n_j}$ is the permutation matrix which identifies the nodes in the forest as distinct from those of the core of the WDS.
A new lemma is proposed as follows:

LEMMA 1. *Suppose*

$$\mathbf{Q} = \begin{array}{c} \\ m_1 \\ m_2 \end{array} \begin{array}{c} \begin{array}{c} n \end{array} \\ \begin{pmatrix} \mathbf{P} \\ \mathbf{S} \end{pmatrix} \end{array},$$

$\mathbf{Q} \in \mathbb{R}^{n \times n}$*, is an orthogonal permutation matrix and that* $\mathbf{D} = diag\{d_1, d_2, \cdots, d_n\} \in \mathbb{R}^{n \times n}$ *is diagonal. Then*

$$\mathbf{PDS}^T = \mathbf{0} \tag{4.10}$$

Proof. $P = \left(e_{i_1}, e_{i_2}, \ldots, e_{i_{m_1}}\right)^T$ for a set of indices $T = \{i_1, i_2, \cdots, i_{m_1}\}$ and $S = \left(e_{j_1}, e_{j_2}, \ldots, e_{j_{m_2}}\right)^T$ for a set of indices $V = \{j_1, j_2, \cdots, j_{m_2}\}$. Note that $\mathbf{T} \cap \mathbf{S} = \emptyset$. Now for some $1 \leq i \leq m_1, 1 \leq j \leq m_2$ there exist $i_t \neq j_s$ such that

$$e_i^T \mathbf{P}\mathbf{D}\mathbf{S}^{\mathbf{T}} e_j = e_{it}^T d_{it} e_{js} = 0$$

from which (4.10) follows. End of LEMMA 1□

After applying the FCPA permutation, the system equations become

$$\boldsymbol{P_1} \times \begin{bmatrix} \boldsymbol{G} & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{bmatrix} \times \boldsymbol{P_1}^T \times \boldsymbol{P_1} \times \begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{h} \end{pmatrix} - \boldsymbol{P_1} \times \begin{pmatrix} \boldsymbol{A_2 e_l} \\ \boldsymbol{d} \end{pmatrix} = \boldsymbol{O} \tag{4.11}$$

and with this permutation, Eq. (4.3) leads to

$$\begin{pmatrix} \boldsymbol{SGS}^T & \boldsymbol{O} & -\boldsymbol{SA_1C}^T & -\boldsymbol{SA_1T}^T \\ \boldsymbol{O} & \begin{bmatrix} \boldsymbol{PGP}^T & -\boldsymbol{PA_1C}^T \\ -\boldsymbol{CA_1}^T\boldsymbol{P}^T & \boldsymbol{O} \end{bmatrix} & \boldsymbol{O} \\ -\boldsymbol{TA_1}^T\boldsymbol{S}^T & \boldsymbol{O} & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{Sq} \\ \boldsymbol{Pq} \\ \boldsymbol{Ch} \\ \boldsymbol{Th} \end{pmatrix} - \begin{pmatrix} \boldsymbol{SA_2 e_l} \\ \boldsymbol{PA_2 e_l} \\ \boldsymbol{Cd} \\ \boldsymbol{Td} \end{pmatrix} = \boldsymbol{O} \tag{4.12}$$

where (i) $\begin{pmatrix} -\boldsymbol{SA_1C}^T & -\boldsymbol{SA_1T}^T \\ -\boldsymbol{PA_1}^T\boldsymbol{C}^T & -\boldsymbol{PA_1T}^T \end{pmatrix}$, which is the original top right two-by-two block in the first matrix of Eq. (4.12), is the permuted $\boldsymbol{A_1}$ matrix, in which the (2,2) block, which is $-\boldsymbol{PA_1T}^T$, becomes $\boldsymbol{O}$ because the pipes in the core do not connect to any nodes in the forest which are not root nodes, and (ii) $\begin{pmatrix} \boldsymbol{SGS}^T & \boldsymbol{SGP}^T \\ \boldsymbol{PGS}^T & \boldsymbol{PGP}^T \end{pmatrix}$, which is the original top left two-by-two matrix of Eq. (4.12), is the permuted $\boldsymbol{G}$ matrix, in which it is evident from the Lemma 1 that the (1,2) and (2,1) blocks, which are $\boldsymbol{SGP}^T$ and $\boldsymbol{PGS}^T$, become $\boldsymbol{O}$.

The top right block (the (1,2) block) of the permuted $\boldsymbol{A_1}$ matrix, $-\boldsymbol{SA_1T}^T$, is invertible and can be permuted to be lower triangular form because it represents the union of the trees. The flows of the pipes in the forest, $\boldsymbol{Sq}$ can be found directly from

$$\boldsymbol{Sq} = -\left(\boldsymbol{TA_1}^T\boldsymbol{S}^T\right)^{-1}\boldsymbol{Td}. \tag{4.13}$$

Rewriting the second and third block equations of Eq. (4.12) gives:

$$\begin{bmatrix} \boldsymbol{PGP}^T & -\boldsymbol{PA_1C}^T \\ -\boldsymbol{CA_1}^T\boldsymbol{P} & \boldsymbol{O} \end{bmatrix} \begin{pmatrix} \boldsymbol{Pq} \\ \boldsymbol{Ch} \end{pmatrix} = \begin{pmatrix} \boldsymbol{PA_2 e_l} \\ \boldsymbol{Cd} + \boldsymbol{CA_1}^T\boldsymbol{S}^T\boldsymbol{Sq} \end{pmatrix}, \tag{4.14}$$

which is the system for the reduced non-linear problem (for the core heads and flows). This can be expressed as:

$$\begin{bmatrix} \widehat{\boldsymbol{G}} & -\widehat{\boldsymbol{A}}_1 \\ -\widehat{\boldsymbol{A}}_1^T & \boldsymbol{O} \end{bmatrix} \begin{pmatrix} \widehat{\boldsymbol{q}} \\ \widehat{\boldsymbol{h}} \end{pmatrix} = \begin{pmatrix} \widehat{\boldsymbol{A}}_2 \boldsymbol{e_l} \\ \widehat{\boldsymbol{d}} \end{pmatrix} \tag{4.15}$$

where $\widehat{\boldsymbol{G}} = \boldsymbol{PGP}^T, \widehat{\boldsymbol{A}}_1 = \boldsymbol{PA_1C}^T, \widehat{\boldsymbol{q}} = \boldsymbol{Pq}, \widehat{\boldsymbol{h}} = \boldsymbol{Ch}, \widehat{\boldsymbol{A}}_2 = \boldsymbol{PA_2}$, and $\widehat{\boldsymbol{d}} = \boldsymbol{Cd} + \boldsymbol{CA_1}^T\boldsymbol{S}^T\boldsymbol{Sq}$ and then the Newton iterative method is applied to Eq. (4.15).

Finally, once the iterative solution process for the core has stopped, the forest heads can be found by solving a linear system:

$$\boldsymbol{Th} = \left(-\boldsymbol{SA_1T}^T\right)^{-1}\left(\boldsymbol{SA_2 e_l} - \boldsymbol{SGS}^T\boldsymbol{Sq} + \boldsymbol{SA_1C}^T\boldsymbol{Ch}\right) \tag{4.16}$$

after the flows and heads of the core network are found.

The FCPA simplifies the problem by identifying the linear part of the problem and solving it separately from the core to avoid unnecessary computation in the iterative process.

Reformulated co-tree flows method

We first introduce some graph notation before we describe the RCTM in more detail. A spanning tree is an acyclic graph which traverses every node in a graph, such that the addition of any co-tree element creates a loop. A WDS, with or without a forest, can be partitioned into two subgraphs: a spanning tree $G_{st} = (V_{st}, E_{st})$, and a co-tree $G_{ct} = (V_{ct}, E_{ct})$, so that $E_{st} \cup E_{ct} = E_c$, $E_{st} \cap E_{ct} = \emptyset$. The flows in the spanning tree can be found directly from the co-tree flows.

Elhay et al. (2014) proposed the reformulated co-tree flow method (RCTM) to exploit this relationship between the co-tree flows and spanning tree flows. This is achieved by applying the Schilders' factorization to permute the $\boldsymbol{A_1}$ matrix into a lower triangular square block at the top, representing a spanning tree, and a rectangular block below, representing the corresponding co-tree. The RCTM starts by generating a permutation matrix:

$$\boldsymbol{P_2} = \begin{array}{c} \\ n_{st} \\ n_{ct} \\ n_j \end{array} \begin{array}{c} \begin{array}{cc} n_p & n_j \end{array} \\ \begin{pmatrix} \boldsymbol{K_1} & \boldsymbol{O} \\ \boldsymbol{K_2} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{R} \end{pmatrix} \end{array} \tag{4.17}$$

where $\begin{bmatrix} \boldsymbol{K_1} \\ \boldsymbol{K_2} \end{bmatrix} \in \mathbb{R}^{n_p \times n_p}$ is the square orthogonal permutation matrix for the pipes, in which $\boldsymbol{K_1} \in \mathbb{R}^{n_{st} \times n_p}$ is the permutation matrix that identifies the pipes in the spanning tree as distinct from those in the co-tree and $\boldsymbol{K_2} \in \mathbb{R}^{n_{ct} \times n_p}$ is the permutation matrix for the pipes in the co-tree, $\boldsymbol{R}$ is the permutation matrix for the nodes to have the same sequence that are traversed by the corresponding spanning tree pipes.

The permuted system equation of the RCTM is:

$$\boldsymbol{P_2} \times \begin{bmatrix} \widehat{\boldsymbol{G}} & -\widehat{\boldsymbol{A}}_1 \\ -\widehat{\boldsymbol{A}}_1{}^T & \boldsymbol{O} \end{bmatrix} \times \boldsymbol{P_2}{}^T \times \boldsymbol{P_2} \times \begin{pmatrix} \widehat{\boldsymbol{q}} \\ \widehat{\boldsymbol{h}} \end{pmatrix} - \boldsymbol{P_2} \times \begin{pmatrix} \widehat{\boldsymbol{A}}_2 \boldsymbol{e}_l \\ \widehat{\boldsymbol{d}} \end{pmatrix} = \boldsymbol{O} \tag{4.18}$$

and (4.14) becomes:

$$\begin{pmatrix} \boldsymbol{K_1}\widehat{\boldsymbol{G}}\boldsymbol{K_1}{}^T & \boldsymbol{O} & -\boldsymbol{K_1}\widehat{\boldsymbol{A}}_1\boldsymbol{R}^T \\ \boldsymbol{O} & \boldsymbol{K_2}\widehat{\boldsymbol{G}}\boldsymbol{K_2}{}^T & -\boldsymbol{K_2}\widehat{\boldsymbol{A}}_1\boldsymbol{R}^T \\ -\boldsymbol{R}\widehat{\boldsymbol{A}}_1{}^T\boldsymbol{K_1}{}^T & -\boldsymbol{R}\widehat{\boldsymbol{A}}_1{}^T\boldsymbol{K_2}{}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{K_1}\widehat{\boldsymbol{q}} \\ \boldsymbol{K_2}\widehat{\boldsymbol{q}} \\ \boldsymbol{R}\widehat{\boldsymbol{h}} \end{pmatrix} - \begin{pmatrix} \boldsymbol{K_1}\widehat{\boldsymbol{A}}_2\boldsymbol{e}_l \\ \boldsymbol{K_2}\widehat{\boldsymbol{A}}_2\boldsymbol{e}_l \\ \boldsymbol{R}\widehat{\boldsymbol{d}} \end{pmatrix} = \boldsymbol{O} \tag{4.19}$$

in which the (1,2) and (2,1) blocks, which are $\boldsymbol{K_1}\boldsymbol{G}\boldsymbol{K_2}{}^T$ and $\boldsymbol{K_2}\boldsymbol{G}\boldsymbol{K_1}{}^T$, become $\boldsymbol{O}$ for the reasons shown in Lemma 1.

The complexity of the problem is reduced because the (1,3) block of the key matrix, $-\boldsymbol{K_1}\widehat{\boldsymbol{A}}_1\boldsymbol{R}^T$, is lower triangular and invertible. As a result, the Newton solver can be partitioned into a different two-step process: (i) solve for the non-linear co-tree flows, (ii) solve for the corresponding spanning tree flows (a linear computation). The heads can be solved once after the iterative process has completed.

By permuting the network equations into (4.19), Elhay el al. (2014) proposed the following equations to solve the WDS for the flows, first for the spanning tree flows $\boldsymbol{q}_1^{(m+1)}$:

$$\boldsymbol{q}_1^{(m+1)} = \boldsymbol{L_{21}}{}^T \boldsymbol{q}_2^{(m)} - \boldsymbol{R}_1^{-T} \widehat{\boldsymbol{d}} \tag{4.20}$$

and second for the co-tree flows $\boldsymbol{q}_2^{(m+1)}$:

$$\boldsymbol{W}^{(m+1)} \boldsymbol{q}_2^{(m+1)} = \boldsymbol{L_{21}} \left(\boldsymbol{F}_1^{(m+1)} - \boldsymbol{G}_1^{(m+1)} \right) \boldsymbol{q}_1^{(m+1)} + \left(\boldsymbol{F}_2^{(m)} - \boldsymbol{G}_2^{(m)} \right) \boldsymbol{q}_2^{(m)} + \boldsymbol{a_2} \tag{4.21}$$

where: $\boldsymbol{R_1} = \boldsymbol{K_1}\widehat{\boldsymbol{A}}_{\boldsymbol{1}}\boldsymbol{R}^T$; $\boldsymbol{R_2} = \boldsymbol{K_2}\widehat{\boldsymbol{A}}_{\boldsymbol{1}}\boldsymbol{R}^T$; $\boldsymbol{L_{21}} = -\boldsymbol{R_2}\boldsymbol{R_1}^{-T}$; $\boldsymbol{F}_1^{(m)} = \boldsymbol{K_1}\widehat{\boldsymbol{F}^{(m)}}\boldsymbol{K_1}^T$; $\boldsymbol{F}_2^{(m)} = \boldsymbol{K_2}\widehat{\boldsymbol{F}^{(m)}}\boldsymbol{K_2}^T$ $\boldsymbol{G}_1^{(m)} = \boldsymbol{K_1}\widehat{\boldsymbol{G}^{(m)}}\boldsymbol{K_1}^T$; $\boldsymbol{G}_2^{(m)} = \boldsymbol{K_2}\widehat{\boldsymbol{G}^{(m)}}\boldsymbol{K_2}^T$; $\boldsymbol{a_1} = \boldsymbol{K_1}\widehat{\boldsymbol{A}}_{\boldsymbol{2}}\boldsymbol{e_l}$; $\boldsymbol{a_2} = \boldsymbol{L_{21}}\boldsymbol{K_1}\widehat{\boldsymbol{A}}_{\boldsymbol{2}}\boldsymbol{e_l} + \boldsymbol{K_2}\widehat{\boldsymbol{A}}_{\boldsymbol{2}}\boldsymbol{e_l}$; $\boldsymbol{W}^{(m)} = \boldsymbol{L_{21}}(\boldsymbol{F}_1^{(m)})^{-1}\boldsymbol{L_{21}}^T + (\boldsymbol{F}_2^{(m)})^{-1}$. Note that in Eq. (4.20), an initial set of the co-tree flows $\boldsymbol{q}_2^{(0)}$ is needed to commence the solution process.

The heads are found after the iterative process of the RCTM by using a linear solution process:

$$\boldsymbol{R}_1\boldsymbol{h} = \boldsymbol{F}_1\boldsymbol{q}_1^{(m+1)} - (\boldsymbol{F}_1 - \boldsymbol{G}_1)\,\boldsymbol{q}_1^{(m)} - \boldsymbol{a_1} \tag{4.22}$$

This partitioning of the network equations reduces the size of the non-linear component of the solver to $n_p - n_j$ (the number of co-tree elements in the network). It has been proven by Elhay et al. (2014) that the RCTM and the GGA have identical iterative results and solutions if the same starting values are used. However, for RCTM, the user only needs to set the initial flow estimates for the co-tree pipes, $\boldsymbol{q}_2^{(0)}$, in contrast to GGA where initial flow estimates are required for all pipes. The flows in the complementary spanning tree pipes are generated by Eq.(4.20).

4.7 Methodology

This section describes the methodology used to carry out a comparative study of the WDS solution methods. The following describes the software platform used to run the benchmarking simulations. This description is followed by the proposed algorithm evaluation method.

4.7.1 The Software Platform

To run the benchmark tests required by this study a hydraulic simulation toolkit, **WDSLib**, was created. This toolkit, written in C++, incorporated the solution methods studied in this paper, which include the GGA, the GGA with the FCPA, the RCTM, and the RCMT with the FCPA. In order to provide a useful platform for comparison, the solution methods were implemented using fast and modularized code. A focus of attention in this research has been the implementation correctness, robustness and efficiency. The correctness* of the toolkit has been validated against a reference MATLAB implementation. The differences between all results (intermediate and final) produced by the C++ toolkit and the MATLAB implementation were shown to be smaller than 10^{-10}. In the interest of toolkit robustness, special attention has been paid to numerical processes to guard against avoidable failures, such as loss of significance through subtractive cancellation, and numerical errors, such as division by zero. The data structures and code libraries in the toolkit are shared and all solution method implementations have been carefully designed to ensure fairness of performance comparisons between algorithms.

The following subsections describe the measures taken in the implementation the solution methods to help ensure the validity of the timing experiments for the case study results. These include measures to ensure accurate timing results, minimization of memory use, and numerical robustness.

*terms recognized in Computer Science will be designated by asterisk superscript

Timing Considerations

C++ was chosen as the implementation language because timings in MATLAB are confounded by a variety of factors. The MATLAB programming language is a hybrid of interpreted language and compiled language: some codes are interpreted by MATLAB with no compilation, some codes are partially compiled by a closed-source just-in-time (JIT) compiler, and some codes are fully compiled. MATLAB may also perform additional work and bookkeeping between the interpretation of one line and the next.

In contrast, C++ is a compiled language: the compiler translates the code into native machine instructions which are later executed by the hardware. This faster and much simpler model of execution overcomes many of the problems associated with MATLAB timing. As a consequence, a C++ implementation forms a better basis for a fair comparison of different WDS solution methods.

When executing the timing experiments in this work, each code module reports the time spent in it by sampling wall clock time at the start and end of its execution. Although the overhead for sampling wall clock time is small, there are at least two special considerations involved in the interpretation of these timings: (i) the operating system, at its own discretion, may launch background processes (for example anti-virus software), which distort the timings and (ii) extrapolating the timing for multiple simulations (as may occur, for example, in a genetic algorithm or other evolutionary algorithm run) from a single analysis must be done with care because the relationship between the different settings is not linear. More detail on these issues is given in a later section describing the proposed algorithm evaluation method.

Memory Considerations

Memory management for each method was very carefully handled to advantage that method in the interest of a fair comparison. To offer further assurance of the correctness of memory management, Valgrind (Nethercote and Seward 2007), a programming tool for memory debugging, memory leak detection and profiling tool, was deployed during testing to detect any memory leaks, memory corruption, and double-freeing.

Numerical considerations

The calculations in this paper were performed in C++ under IEEE-standard double precision floating point arithmetic with machine epsilon $\epsilon_{mach} = 2.2204^{-16}$. The constants and parameters in every equation were gathered and replaced by full 20-decimal digit accuracy values. In addition, all dependent constants in mathematical expressions were removed.

The stopping test used in this benchmark study is $\frac{||\boldsymbol{q}^{(m+1)} - \boldsymbol{q}^{(m)}||_\infty}{||\boldsymbol{q}^{(m+1)}||_\infty} \leq 10^{-6}$ to ensure a fair comparison between the GGA and the RCTM because one of the benefits of using the RCTM is that it only solves for the pipe flows within the iterative phase.

In the setup of the benchmark experiments, there are two primary dangers that are associated with floating point arithmetic that cannot be ignored: (i) subtractive cancellation and (ii) overflow and underflow. To avoid problems associated with these, all input variables are scaled to a similar range to minimize the risk of avoidable computational inaccuracy or failure in floating point arithmetic. It is important to note that unscaled or poorly scaled variables can unnecessarily confound the computation. After scaling, variables become

Table 4.1. WDS variables and units

Variables	SI unit	US unit	Scaling factor
Length	m	ft	$L_0 = max\,(\boldsymbol{L})$
Diameter	m	ft	$D_0 = max\,(\boldsymbol{D})$
Nodal head	m	ft	$h_0 = max\,(\boldsymbol{e_l})$
Source elevation	m	ft	$e_{l_0} = max\,(\boldsymbol{e_l})$
flow	m^3/s	ft^3/s	$q_0 = max\,(\boldsymbol{d})$
demand	m^3/s	ft^3/s	$d_0 = max\,(\boldsymbol{d})$
G[1], F[1]	s/m^2	s/ft^2	$G_0 = \frac{L_0}{D_0^p}\lvert q_0\rvert^{n-1}$

physically dimensionless, which allows computation which is independent of the system of measurement units.

The variables that are provided in EPANET input files for the experiments and their corresponding units in US Customary and SI system are shown in Table 4.1. The system equations were modified to use dimensionless variables. Once the stopping test has been satisfied, the original variables can then be recovered by reversing the initial scaling.

In these experiments approximate minimum degree permutation (AMD) and sparse Cholesky decomposition from SuiteSparse (Davis et al. 2013) have been used. SuiteSparse is a state-of-the-art sparse arithmetic package with exceptional performance.

4.7.2 Proposed algorithm evaluation method

In this work, there are two settings of interest: a once-off network simulation setting and a multi-run setting such as in a genetic algorithm or evolutionary algorithm (EA) that requires many simulations where say the network topology is invariant but pipe diameters can vary. In the case studies presented in this paper results are presented for: (1) a once-off simulation setting and (2) a multi-simulation setting, as might be used in an EA setting for WDS design.

In the experiments, in order to avoid unnecessary computations, each module of implementation code is categorized according to the number of times it needs to be invoked in the context of the given setting. This categorization is described in the following subsection.

Module Categorization

For the purposes of modeling execution times, code modules in a multiple simulation run can be divided into three categories: (i) modules that run only once for every *multiple simulation* are called level-1 modules (L_1). The level of a module is determined by the number of times it would be run. Examples of L_1 modules include the module that loads the WDS network configuration file and the module that identifies the forest pipes in FCPA; (ii) modules that are run once for every *simulation* are called level-2 modules (L_2). Examples of L_2 modules are those that initialize, respectively, all pipe flows in the GGA, core flows in FCPA and co-tree flows in RCTM; (iii) modules that are run once for each *iteration* of every simulation are called level-3 modules (L_3). An example of an L_3 module is the module computing the $\boldsymbol{G}$ and $\boldsymbol{F}$ matrices in any of the solution methods described here.

In a once-off network simulation setting, for each trial, a given solver configuration is used to solve an input network and the time to complete the solution is measured. In this setting, the FCPA and RCTM modules require certain computations which only need to be done once every iterative phase. The computation for these so-called invariants can be lifted* out of the iterative phase and executed once per evaluation, thus saving computation time. The second setting considered here is a multiple simulation run, such as one might find in a GA to optimize the design of a WDS, for example. In this setting, a network with a fixed topology is solved multiple times for say different pipe diameters. In this case, because of the fixed topology, the FCPA and RCTM have modules that need only be run once for each multiple simulation run. This again reduces the overall simulation runtime.

4.8 Case Studies

The implementation described above was used to evaluate the efficiency of the four solution methods in two simulation settings: a once-off simulation setting, in which the steady-state heads and flows are computed just once with the given WDS parameters, and a design setting, in which the steady-state heads and flows need to be computed many times to, say, find the least-cost design by EA optimization. In the methodology section, the four solution methods, namely GGA, GGA with FCPA, RCTM, and RCTM with FCPA, were decomposed into modules. These modules were categorized into levels by using the method described in the previous section. Fig.4.1 shows the module classifications and the level of repetition of different modules for the different solution methods. The columns of the block diagram show different solution methods and the rows of the block diagram show the levels of repetition of the steps as they would be executed in a multiple simulation setting. In the body of the table, the different methods are separated by double vertical lines where column(s) intersect a box, which means the modules that are represented by that box are used by the method(s) that are presented by that column(s). For example, the modules for RCTM that are required to be carried out once before a multiple simulation include: (i) load the configuration file and read EPANET input file, (ii) find the Schilders' spanning tree co-tree factorization and (iii) find and apply the AMD bandwidth reduction.

Table 4.2. Benchmark networks summary

	Full Network			Forest & Core Networks			Co-tree Network
Network	n_p	n_j	n_s	$n_f(n_f/n_p^{\#})$	n_{j_c}	n_{p_c}	n_{ct}
N_1	934	848	8	361 (38%)	573	487	84
N_2	1118	1039	2	321 (29%)	797	718	79
N_3	1975	1770	4	823 (42%)	1152	947	205
N_4	2465	1890	3	429 (17%)	2036	1461	757
N_5	2509	2443	2	702 (28%)	1087	1741	66
N_6	8585	8392	2	1850 (22%)	6735	6542	193
N_7	14830	12523	7	2932 (20%)	11898	9591	2307
N_8	19647	17971	15	4414 (22%)	15232	13557	1676

$^{\#}n_f/n_p$ shows the proportion of the forest

Eight benchmark networks were used to study the effectiveness of each method under different design settings. The networks used here were derived from Simpson et al. (2012)

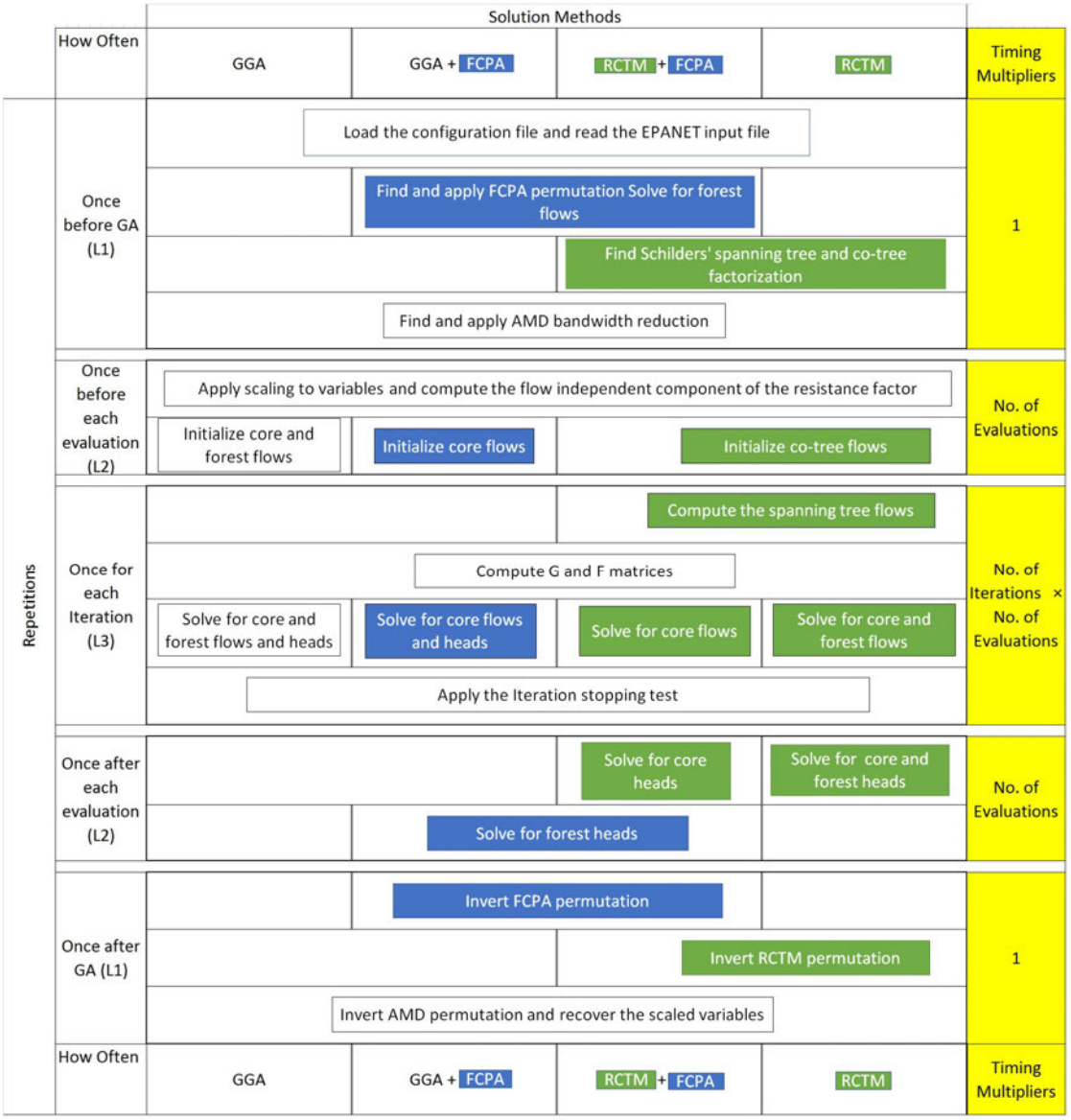


Fig. 4.1. Module classification for GGA, GGA and FCPA, RCTM and RCTM with FCPA

with some slight modifications to remove control devices, patterns, curves and pumps. Details of these networks are given in that paper. The basic network characteristics of the case study networks are summarized in Table 4.2. All the case study networks are realistic. The ratio between the number of the forest pipes and the total number of pipes ranges between 17% and 42%. The ratio between the number of the co-tree pipes and the total number of the pipes ranges between 3% and 31%. Each of the four solution methods and the GGA implementation in the EPANET are applied to these eight benchmark networks. It has been pointed out by Guidolin et al. (2010) that the code implementation in EPANET are highly optimized for its performance and not written to be readily decomposed into modules for different tasks.. As a result, it is difficult, if not impossible, to apply the module categorization method proposed in the current paper to EPANET. The times taken by both `ENopen`, the EPANET function for reading the input file and memory allocation, and `ENclose`, the EPANET function for memory deallocation are not counted in the final

EPANET timing.

The next section presents the timing analysis for these case study networks. Of course, the same benchmark tests performed on another computing platform will produce quite different results, but the authors believe that the relative timings will remain the same.

4.9 Results and Discussion

The benchmark tests were performed on a Intel(R) Xeon(R) CPU E5-2698 v3 running at 2.30 GHz with 16 cores and 40MB L_3 cache on a High Performance Computing machine called Phoenix at the University of Adelaide. The number of cores allocated to each test was limited to one. Each timing test was repeated 15 times on each benchmark network.

Table 4.3. Detailed statistics of the time of each module of the GGA applied to network N_1 (15 runs)

Type	Function	Statistical Properties (milliseconds)						
		Min	Max	Mean(%)		Median	Std. Dev	Std. Err
Computed once every multiple simulation								
L_1	AMD	0.54	1.45	0.67(66.9%)		0.57	0.20	0.04
	Housekeeping	0.28	0.58	0.33 (33.1%)		0.29	0.09	0.02
Sub-Total Statistics		0.82	1.74	1.00 (14.8%)		0.90	0.20	0.04
Computed once every simulation								
L_2	GetGF-1	0.01	0.01	0.01(23.1%)		0.01	0.00	0.00
	init	0.00	0.00	0.00(5.4%)		0.00	0.00	0.00
	scaling	0.04	0.05	0.04 (66.3%)		0.04	0.00	0.00
	Housekeeping	0.00	0.00	0.00(5.2%)		0.00	0.00	0.00
Sub-Total Statistics		0.05	0.07	0.06 (0.8%)		0.06	0.00	0.00
Iterative Phase								
					T_{L_3} /N_I			
L_3 (N_I=8)	GetGF-2	1.19	2.52	1.42(24.9%)	0.18	1.28	0.30	0.02
	Linear Solver	3.11	4.32	3.53(62.1%)	0.44	3.49	0.31	0.06
	2nd Phase	0.31	0.59	0.36(6.4%)	0.05	0.33	0.08	0.02
	normTest	0.11	0.57	0.14(2.4%)	0.02	0.12	0.09	0.02
	Other	0.18	0.85	0.24(4.2%)	0.03	0.19	0.14	0.03
Sub-Total Statistics		4.91	6.52	5.69(84.3%)	0.71	5.69	0.41	0.00
Computed once every multiple simulation								
L_1	UndoPermutation	0.00	0.00	0.00(100%)		0.00	0.00	0.00
Sub-Total Statistics		0.00	0.00	0.00(0.05%)		0.00	0.00	0.00
Grand-Total Statistics		6.02	7.55	6.75		6.67	0.44	0.08

4.9.1 Once-off Simulation Setting

The mean, minimum, maximum and median wall clock times for all modules were collected. As an example, the detailed statistics of the time for each module of the GGA method

applied to N_1, the first case study network. Table 4.3 presents the detailed timing results of all modules used in the toolkit implementation of the GGA without FCPA at the three levels of repetition: once every multiple simulation (L_1), once every simulation (L_2) and once every solver iteration (L_3). The sub-total for each level is summarized after each level of repetition and the grand-total is shown in the last row. The percentage inside the bracket shows the contribution of each of the modules towards its level of repetition and the contribution of each of the levels towards the total runtime. For example, the AMD permutation contributes 66.9% of the L_1 time and the all L_1 modules contribute 14.8% towards the total runtime. The mean time for a once-off simulation of the N_1 network is 6.75 ms. Of the total time, 84.3% was spent on L_3 tasks. The two most time-consuming tasks are the linear solver in the iterative process, which solves the linearization of the non-linear problem by using Eq. (4.7), and getGF-2, which computes the derivatives of the head-loss equations.

Table 4.4. The mean time of once-off simulation run averaged over 15 once-off simulations for each of the four solution methods applied to the eight case study networks (milliseconds±standard error) and the % diff. refers to relative difference compared to the GGA mean time

	GGA	GGA with FCPA		RCTM		RCTM with FCPA		EPANET	
	Mean time	Mean time	%diff.	Mean time	%diff.	Mean time	%diff.	Mean time	%diff.
N_1	6.75 ± 0.08	4.66 ± 0.07	-31	5.37 ± 0.09	-20	$\mathbf{4.56 \pm 0.08}$	-32	9.09	+35
N_2	8.48 ± 0.07	$\mathbf{6.61 \pm 0.07}$	-22	9.98 ± 0.12	+18	8.97 ± 0.08	+6	16.75	+98
N_3	13.88 ± 0.11	$\mathbf{8.72 \pm 0.09}$	-37	11.52 ± 0.10	-17	9.05 ± 0.06	-35	21.46	+55
N_4	14.63 ± 0.32	$\mathbf{12.68 \pm 0.53}$	-19	17.09 ± 0.85	+47	16.28 ± 0.47	+35	26.45	+81
N_5	16.87 ± 0.24	12.19 ± 0.13	-28	12.67 ± 0.14	-25	$\mathbf{10.20 \pm 0.13}$	-40	28.46	+69
N_6	49.53 ± 0.19	44.79 ± 0.18	-28	35.34 ± 0.17	-29	$\mathbf{32.53 \pm 0.15}$	-39	172.84	+249
N_7	83.39 ± 0.42	$\mathbf{63.06 \pm 0.65}$	-24	169.50 ± 1.61	+103	156.61 ± 1.16	+88	307.17	+268
N_8	192.10 ± 3.85	$\mathbf{131.82 \pm 4.0}$	-31	352.44 ± 9.25	+83	307.16 ± 7.2	+60	600.08	+212

Table 4.5. The number of non-zeros in the key matrices of each of the four solution methods applied to the eight case studies networks and the "relative diff." refers to the relative difference compared to the number of non-zeros in the key matrix of the GGA

	GGA	GGA with FCPA	Relative diff. using FCPA	RCTM	RCTM with FCPA	Relative diff. using RCTM and RCTM with FCPA
N_1	2684	1609	-40%	350	350	-87%
N_2	3265	2302	-29%	1219	1219	-63%
N_3	5708	3239	-43%	2534	2534	-56%
N_4	6714	5429	-19%	6951	6951	+3.5%
N_5	7451	5345	-28%	551	551	-93%
N_6	25554	20004	-22%	2514	2514	-90%
N_7	41147	32351	-21%	32389	32389	-21%
N_8	57233	43991	-23%	73252	73252	+28%

Table 4.4 shows the summary statistics of the 15 repetitions of each solution method applied to the eight benchmark networks. Under a once-off simulation setting, the GGA implementation in this paper has been able to achieve between a 26% and 73% speedup when compared with the GGA implementation in EPANET by implementing the proposed

Table 4.6. The mean of the per-iteration timings for each of the modules in L_3 for the four solution methods applied to the eight case studies (milliseconds)

	GGA				GGA+FCPA			
	GetGF	Linear Solver[#]	2nd Phase	norm test	GetGF	Linear Solver[#]	2nd Phase	norm test
N_1	0.18	0.44	0.36	0.14	0.14	0.27	0.03	0.01
N_2	0.22	0.61	0.02	0.02	0.21	0.39	0.04	0.01
N_3	0.20	1.11	0.03	0.03	0.12	0.56	0.07	0.02
N_4	0.66	1.67	0.05	0.03	0.47	1.36	0.03	0.02
N_5	0.42	1.47	0.04	0.03	0.33	0.98	0.03	0.02
N_6	0.48	1.49	0.05	0.04	0.40	0.96	0.03	0.01
N_7	1.92	5.70	0.23	0.09	1.57	3.94	0.22	0.07
N_8	3.17	12.38	0.38	0.21	2.86	7.72	0.33	0.12
	RCTM				RCTM+FCPA			
	GetGF	Linear Solver[#]	2nd Phase	norm test	GetGF	Linear Solver[#]	2nd Phase	norm test
N_1	0.16	0.14	0.02	0.01	0.14	0.11	0.02	0.01
N_2	0.22	0.29	0.05	0.02	0.20	0.26	0.04	0.01
N_3	0.20	0.50	0.07	0.02	0.11	0.41	0.06	0.02
N_4	0.56	1.54	0.11	0.03	0.45	1.47	0.10	0.03
N_5	0.43	0.39	0.07	0.03	0.32	0.31	0.05	0.02
N_6	0.47	0.37	0.07	0.03	0.42	0.30	0.05	0.02
N_7	1.90	5.49	0.43	0.10	1.65	5.30	0.40	0.08
N_8	3.51	17.71	1.68	0.22	3.08	15.97	1.35	0.17

[#] within the iterative solution process

module categorization. The best performing algorithm combination for each network is highlighted in **bold**. Both the GGA and the RCTM benefit from the use of the FCPA (between 19.3% and 37.2% of time saved for the GGA, between 7.6% and 21.4% saved for the RCTM).

The number of non-zeros in the key matrices is commonly used as an indicator of the computational complexity of the Cholesky factorization when sparse arithmetic is used. The numbers of non-zeros in the key matrices of the four WDS solution methods are summarized in Table 4.5. The number of non-zeros in the key matrix of the GGA is a topology-related constant whereas the number of non-zeros in the key matrix of the RCTM is determined by the choice of spanning tree. Network N_8 is the only case where the number of non-zero elements in the key matrix of the RCTM is significantly greater than that of the GGA, therefore network N_8 is the only case where the per-iteration runtime of the RCTM linear solver is greater than that of the GGA (Table 4.6). Using the FCPA with the GGA can reduce the number of non-zeros in its key matrix. Moreover, the dimension of the non-linear problem reduces from n_p to n_{p_c} which reduces the per-iteration execution time when computing the head loss derivatives, second phase and the stopping test. Although the number of non-zeros in the key matrix of the RCTM is independent of whether or not the FCPA is used, using the FCPA does: (i) reduce the computation time of the matrix multiplication in the linear solver, (ii) reduce the dimension of the search space which speeds up the process of partitioning the co-tree pipes from the spanning tree pipes in the

RCTM, and (iii) reduce the number of pipes in the spanning tree. This can be seen by the per-iteration execution times for each of the L_3 modules, which are shown in the Table 4.6.

Table 4.7. The number of iterations required for each of the four solution methods to satisfy the stopping test for the eight case studies networks. The "relative diff." refers to the relative difference compared to the number of iterations for the GGA

	GGA	GGA with FCPA	RCTM	RCTM with FCPA	Relative diff. using RCTM
N_1	8	8	12	12	+50%
N_2	8	8	13	13	+62.5%
N_3	8	8	9	9	+12.5%
N_4	9	9	13	13	+44.4%
N_5	8	8	10	10	+25%
N_6	10	10	12	12	+20%
N_7	9	9	13	13	+44.4%
N_8	9	9	11	11	+22.2%

The number of iterations required for each of the four solution methods to satisfy the stopping test for the eight case studies networks is shown in the Table 4.7. It is evident from Table 4.7 that the GGA took exactly the same number of iterations to satisfy the stopping test with or without the FCPA. The flows in the forest network satisfy a linear system, which does not change from one iteration to the next. Therefore, the flows in the forest pipes reach their steady-state after the first iteration. Similarly, the RCTM with or without FCPA takes the same number of iterations. In the cases that were analyzed in this study, the RCTM required a greater number of iterations to satisfy the stopping test compared to the GGA. This is because different mechanisms are used to generate a set of initial flows for the two methods as discussed previously.

It is worth using the FCPA in conjunction with both the GGA and RCTM for a once-off simulation given that FCPA decreases the L_3 per-iteration time without increasing the number of iterations per module. Interestingly, a smaller per-iteration time is required by the L_3 modules of the RCTM except for network N_8. However, RCTM requires a greater number of iterations for all the case study networks. This sometimes causes a greater time for the RCTM to satisfy the stopping test.

4.9.2 Multiple Simulation Setting

The performance of the four solution methods under the multiple simulation setting are compared. Pipe diameters for the eight case study networks were randomly generated at each evaluation to simulate an evolutionary algorithm run. It is important to note that the use of randomly generated pipe diameters gives an overestimate of the total runtime. This is because, as EA's progress, the pipe diameters in its population become increasingly realistic, which, on average, should reduce the number of iterations at the L_3 level.

Table 4.8 and Table 4.9 show the detailed timing results of multiple simulations with number of evaluations $N_E = 100,000$ for each of the four solution methods applied to the networks N_1 and N_8. Table 4.8 shows that exploiting the treed nature of network N_1 gives the FCPA a 29% time saving over the GGA and 15% time saving over the RCTM. A smaller saving is achieved by the use of the FCPA for network N_8: 14% for the GGA and 9% for the RCTM. In a multiple simulation setting, the RCTM is more timing-consuming

Table 4.8. The actual time required to perform a multiple simulation, where number of evaluations $N_E = 100,000$, of each of the four solution methods applied to N_1 network (ms unless otherwise stated) and "% diff." refers to the relative difference compared to the GGA

		GGA	GGA with FCPA		RCTM		RCTM with FCPA	
		(ms)	(ms)	% diff.	(ms)	% diff.	(ms)	% diff.
L_1	AMD	1.36	0.64		0.19		0.14	
	FCPA	-	0.66		-	-	0.16	
	RCTM	-	-		0.53		0.33	
	scaling	0.09	0.05		0.04		0.02	
	HouseKeeping	2.05	1.78		3.93		0.36	
Sub-Total		3.50	3.01	-14%	4.69	+34%	1.01	-71%
L_2	GetGF-1	2790.15	1899.44		588.75		422.93	
	init	1345.87	887.32		1703.29		1311.09	
	HouseKeeping	533.19	380.98		811.72		626.93	
Sub-Total		4669.21	3167.74	-32%	3103.76	-34%	2360.95	-49%
L_3	GetGF-2	105292.0	89779.9		105596.0		98439.6	
	Linsolve	166072.0	100730.0		122200.0		95539.0	
	second phase	36483.3	23477.1		19716.9		14872.4	
	normTest	50892.4	34836.7		12753.1		9440.8	
	HouseKeeping	15748.3	12593.3		6605.0		6340.2	
Sub-Total		374488	261417	-30%	266871	-29%	224632	-40%
L_2	reverseFCPA	-	6053.5		-		1776.4	
	reverseRCTM	-	-		1335.5		824.5	
Sub-Total		0	6053.5	—	1335.5	—	2600.93	—
L_1	undo permutation	0.02	0.01		0.002		0.002	
Sub-Total		0.02	0.01	-40%	0.002	-89%	0.002	-91%
		(min.)	(min.)		(min.)		(min.)	
EA runtime		6.35	4.53	-29%	4.53	-29%	3.83	-40%

than the GGA when applied to network N_8 because of the greater number of nonzero elements in its key matrix (Table 4.5).

Table 4.10 shows the actual multiple simulation runtime with 100,000 evaluations for each of the four solution methods applied to the eight case study networks. Under a multi-run simulation setting, the GGA implementation in this paper has been able to achieve between a 35% and 81% speedup when compared with the GGA implementation in EPANET by implementing the proposed module categorization. Note that both the upper and lower range values of the speed-up achieved by implementing the proposed module categorization in a multi-run simulation are higher than those in a once-off simulation. This is because the effectiveness of proposed module categorization and the number of evaluation are directly proportional. The fastest solution methods for each of the case study networks are highlighted in bold. Both the GGA and the RCTM benefit from the use of the FCPA, which is also observed under the once-off simulation setting. The relative time saving accruing from the use of the FCPA is smaller for the RCTM than it is for the GGA.

4.10 Conclusions

This paper presents a reliable benchmark study on four water distribution system demand-driven steady-state solution methods, namely the Global Gradient Algorithm (GGA), the GGA with Forest-Core Partitioning Algorithm (FCPA), the Reformulated Co-Tree flow Method (RCTM), and the RCTM with FCPA. These solution methods were implemented using fast, carefully designed, and modularized C++ code in order to provide a fair basis for comparing these methods.

Table 4.9. The actual time required to perform a multiple simulation, where number of evaluations $N_E = 100,000$, of each of the four solution methods applied to N_8 network (ms unless otherwise stated). The "% diff." refers to the relative difference compared to the GGA

		GGA	GGA with FCPA		RCTM		RCTM with FCPA	
		(ms)	(ms)	% diff.	(ms)	% diff.	(ms)	% diff.
L_1	AMD	14.16	8.49		10.68		8.60	
	FCPA	-	1.75		-		1.64	
	RCTM	-	-		45.30		24.11	
	scaling	0.69	0.36		0.47		0.34	
	housekeeping	7.86	5.39		8.02		5.70	
	Sub-Total	22.71	15.99	-30%	64.47	+184%	40.39	+78%
L_2	GetGF-1	10069.00	7481.70		9920.71		7375.28	
	init	2362.45	1782.70		73221.90		65377.80	
	housekeeping	1686.00	1342.04		42063.10		43556.10	
	Sub-Total	14117.45	10606.44	-25%	125205.71	+787%	116309.18	+723%
L_3	GetGF-2	2331270.0	2173440.0		3732280.0		3561510.0	
	Linsolve	6884030.0	5689170.0		11339200.0		10975000.0	
	second phase	314826.0	280212.0		1129820.0		995822.0	
	normTest	162986.0	112226.0		257123.0		183340.0	
	housekeeping	40008.0	33472.0		54777.0		47128.0	
	Sub-Total	9733120	8288520	-15%	16513200	+70%	15762800.0	+62%
L_2	reverseFCPA	-	18405.5		-		19017.2	
	reverseRCTM	-	-		24182.3		16772.6	
	Sub-Total	0	18405.5		24182.3		35789.8	
L_1	undo permutation	0.03	0.03		0.06		0.06	
	Sub-Total	0.03	0.03	-13%	0.06	+83%	0.06	+64%
		(min.)	(min.)		(min.)		(min.)	
	EA runtime	162.51	138.68	-15%	277.80	+71%	265.34	+63%

Table 4.10. The actual multiple simulation runtime (in minutes) with 100,000 evaluations for each of the four solution methods applied to each of the eight case study networks and the "% diff." refers to relative difference compared to the GGA time

	GGA	GGA with FCPA		RCTM		RCTM with FCPA		EPANET	
	min.	min.	%diff.	min.	%diff.	min.	%diff.	min.	%diff.
N_1	6.35	4.35	-31	4.53	-29	**3.83**	-40	9.70	+53
N_2	8.39	**5.83**	-31	8.67	+3	7.78	-7	13.96	+66
N_3	14.88	**9.64**	-35	12.45	-16	10.30	-31	25.35	+70
N_4	20.86	**16.86**	-19	24.02	+15	21.93	+5	68.08	+226
N_5	16.47	12.50	-24	10.90	-34	**9.26**	-44	32.18	+95
N_6	70.66	58.62	-17	39.58	-44	**36.25**	-49	238.64	+238
N_7	128.01	**94.71**	-26	216.88	+69	204.59	+60	422.62	+230
N_8	162.511	**138.68**	-15	277.80	+71	265.34	+63	843.03	+419

The correctness of the implemented solution methods in C++ has been validated against a MATLAB implementation. The robustness of the implementation was achieved by: (i) incorporating necessary precautions in the numerical processes to guard against avoidable computational failures, (ii) using Valgrind to detect any memory leaks and (iii) scaling the variables to avoid overflow, underflow and subtractive cancellation. Implementation efficiency was achieved by, (i) identifying the program loop invariants and hoisting them out of the program loop to avoid any unnecessary computations and (ii) gathering the constants and parameters in every equation to minimize the number of parameters.

The following observations can be made for the four solution methods by comparing the detailed timing of four WDS solution methods applied to the eight case study networks:

1. In the case studies that have been analyzed, the per-iteration time required to perform the RCTM is less than the GGA except for N_8. However, the RCTM requires a greater number of iterations to satisfy the stopping test which leads to the RCTM requiring more time than the GGA for some case study networks. This is because of the different mechanisms used to generate the initial pipe flow guesses in these methods.
2. In the case studies analyzed, the mean time per-iteration of the L_3 modules (iterative solution procedure to solve the nonlinear equations) is affected by the number of non-zeros in the key matrix and the dimension of the non-linear problem. The smaller the number of non-zeros and the smaller the dimension of the non-linear problem, the smaller the solution time will be.
3. Both the GGA and the RCTM benefit from partitioning the forest component from the core component. The FCPA saves less time for the RCTM than it does for the GGA because the forest component is a part of the spanning tree calculation.
4. Significant time savings have been observed when comparing the implemented solution methods with EPANET for a multiple run simulation setting.

As a final note, a significant proportion of the runtime savings, in the method implementation, can be attributed to the decomposition of the modules of the solution methods into different levels of repetition. This decomposition exploits invariants in the solution process in order to avoid unnecessary computations.

4.11 Acknowledgement

This work was supported with supercomputing resources provided by the Phoenix HPC service at the University of Adelaide.

4.12 Supplemental Data

The data for case study networks N_1, N_3, N_4, and N_7, which are modifications of networks in the public domain, are available on-line. The data for case study networks N2, N5, N6, and N8 are not freely available either because they are proprietary or because of security concerns on the part of the relevant water utilities.

4.13 References

References are included in bibliography Chapter 7. In addition the submitted paper is given in Appendix B.

Chapter 5

Publication 3: A Bridge-Block Partitioning Algorithm for Speeding up Analysis of Water Distribution Systems

5.1 Synopsis

In Chapter 4, WDSLib, a water distribution system simulation toolkit that was developed in Chapter 3, was used as a fair basis for a detailed comparison of the performance of four water distribution system solution methods, namely the global gradient algorithm (GGA), the GGA with the forest-core partitioning algorithm (FCPA), the reformulated co-tree flows method (RCTM), and the RCTM with the FCPA under different settings. Another type of graph property, bridge and block components, has been investigated in this chapter. The bridge-block partitioning algorithm (BBPA) begins by using the FCPA to separate the forest component from the core component. Then, the BBPA further partitions the core component of the network into block and bridge components.

Bridge components are the pipes in the core that are not part of any loop. The solutions for the bridge components can be found by a linear process – in the same way as can the forest component in the FCPA. The remainder of the network is consisting of blocks and solutions for these block components can be found separately. It is possible to separate two blocks with a single node called a cut-vertex. The advantages in speed and reliability for the BBPA arise, in part, from the smaller systems that result from partitioning the network into these smaller blocks, if the core component of the WDS graph is one-connected.

The BBPA exploits the fact that the flows and heads in one block component are weakly coupled with those of the other block components and the solution of the flows and heads in a bridge component is a linear process. The convergence rate for the solution of the core component of a WDS, without the BBPA, is restricted to that of the worst block of the network. The number of iterations required by each block is bounded above by that required by the unpartitioned system.

The use of BBPA can also improve the reliability of the solution. The numerical reliability of the solution can be determined by the condition number of the Schur complement. The condition number of a matrix is the ratio of the largest to the smallest singular value of any square matrix. In most cases, the condition numbers for all the individual blocks will be smaller than the condition number of the full matrix.

In this Chapter, the advantage of using BBPA is demonstrated on eight case studies with between 932 to 19647 pipes and between 848 and 17971 nodes. The global gradient

algorithm (GGA) with the forest-core partitioning algorithm (FCPA) is between 30% and 70% faster if the BBPA is used. The BBPA has the following attractions: (1) it significantly reduces the computation time by partitioning the non-linear system of equations into a number of linear bridges and a number of independent non-linear blocks, (2) it improves the reliability of the solution because the condition number of each block is bounded above by that of the full system, (3) it minimises the need to regularise in the presence of zero flows when the head loss is modelled by the Hazen-William formula, (4) blocks with unchanged demands do not need recomputing in a management setting, and (5) the solution of the blocks can be solved in parallel.

5.1.1 Citation

Qiu, M, Elhay, S, Simpson, AR & Alexander, B 2018, 'A Bridge-Block Partitioning Algorithm for Speeding up Analysis Of Water 1 Distribution Systems', Manuscript submitted for publication to *Journal of Water Resources Planning and Management*.

Statement of Authorship

Title of Paper	A Bridge-Block Partitioning Algorithm for Speeding up Analysis of Water 1 Distribution Systems
Publication Status	☐ Published ☐ Accepted for Publication ☑ Submitted for Publication ☐ Unpublished and Unsubmitted w ork w ritten in manuscript style
Publication Details	Qiu, M, Elhay, S, Simpson, AR & Alexander, B 2018b, 'A Bridge-Block Partitioning Algorithm for Speeding up Analysis Of Water 1 Distribution Systems', Manuscript submitted for publication to Journal of Water Resources Planning and Management.

Principal Author

Name of Principal Author (Candidate)	Mengning Qiu		
Contribution to the Paper	Developed the numerical method, implemented into the WDSLib, conducted numerical analysis and prepared manuscript.		
Overall percentage (%)	80%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	30/04/2018

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Angus R. Simpson		
Contribution to the Paper	Helped in model development, manuscript evaluation and manuscript editing. (10%)		
Signature		Date	30 April 2018

Name of Co-Author	Sylvan Elhay		
Contribution to the Paper	Provided manuscript evaluation, manuscript editing and checking numerical proof. (5%)		
Signature		Date	30-Apr-2018

Name of Co-Author	Bradley Alexander			
Contribution to the Paper	Provided manuscript evaluation and manuscript editing. (5%)			
Signature			Date	01/05/18

5.2 Abstract

Many water distribution system (WDS) solution methods have been developed to perform demand-driven steady-state analysis. These methods are used to solve the non-linear system of equations that model a WDS. WDS networks have structural properties that can often be exploited to speed up these solution methods. One solution method that exploits these structural properties is the forest-core partitioning algorithm that was proposed as a pre-processing and post-processing method that can be used to separate the network into a linear forest component and a non-linear core component. This paper presents a complementary method for pre-and post-processing called the bridge-block partitioning algorithm (BBPA). This method further partitions the core component of the network into a number of linear bridge components and a number of non-linear block components. The use of BBPA to partition a WDS network provides significant advantages over current solution methods in terms of both speed and solution reliability.

5.2.1 Keywords

Global gradient algorithm (GGA); Graph Theory; Bridge-Block Partitoning; Water distribution systems; Hydraulic analysis.

5.3 Introduction

Hydraulic simulation algorithms use mathematical models designed to simulate the hydraulic performance of a water distribution system (WDS) and have played a critical role in the design, operation, and management of WDSs in research and industry. These models have been used for (1) optimizing WDS network design parameters (such as pipe diameters), (2) for calibrating network parameters (such as demand patterns), (3) conducting real-time monitoring and calibration of the network elements in a supervisory control and data acquisition (SCADA) operational setting, and (4) adjusting control devices (such as valves). In hydraulic simulation, the system of equations can be formulated as a large and sparse non-linear saddle-point problem. There are several well-known iterative methods for solving the non-linear saddle-point problem. These include: range space methods, null space methods, and loop-based methods.

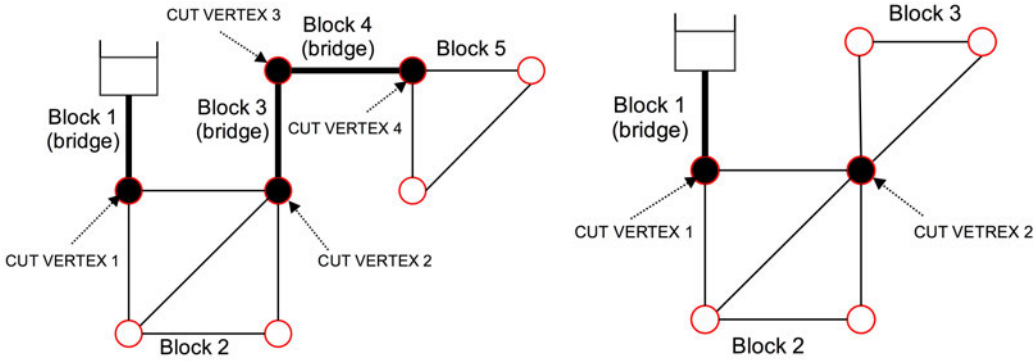
The most widely used WDS solution method is the Global Gradient Algorithm (Todini and Pilati 1988). The GGA, a range space method, takes advantage of the block structure of the full Jacobian matrix to achieve a smaller key matrix in the linearization of the Newton method. Since the development of the GGA, numerous new WDS hydraulic solution methods have been proposed and improvements have been made to existing WDS hydraulic solution methods. Most of these new WDS hydraulic solution methods employ graph theory to decompose or partition the WDS network graph into sub-graphs which results in a smaller system of equations. Deuerlein (2008) introduced a decomposition model for a WDS network graph, in which the one-connected components are categorized as the forest component and the biconnected components are categorized as the core component. After removing the forest component, the core component can be further partitioned into blocks that are connected by bridge elements. After the partitioning processes, a loop flow corrections method is then used. Simpson et al. (2012) proposed a matrix based identification method for the forest component and the core component and introduced the forest-core partitioning algorithm (FCPA). In the FCPA, flows and heads in the forest

component can be solved for just once. The remaining system of equations, representing the core – which has a smaller dimension if the network has a significant forest component – is then solved iteratively by the Newton method. Deuerlein et al. (2015) proposed another graph partitioning algorithm which exploits the properties of network components in series within the core component of the network. This algorithm exploits the fact that flows in the internal tree pipes are linearly dependent on the topological minor. This relationship has been used to partition the non-linear Newton solver into a non-linear global step and a linear local step.

The loop-based method is a solution method which attempts to reduce the size of the simulation problem. The oldest loop-based method (and the oldest method overall) is the Hardy Cross method (Cross 1936). In the Hardy Cross method, there are two sets of equations –(i) mass conservation equations and (ii) loop energy conservation equations– which are used to model the underlying relationship of the flows and heads of a WDS. This non-linear system of equations is solved by successive approximation, in which a set of initial flows that satisfies continuity is successively corrected until a predefined stopping test has been met. The Hardy Cross method is an iterative manual method that was popular for its simplicity before the introduction of computers. Epp and Fowler (1970) also explored the possibility of using a loop formulation to perform hydraulic simulations. They proposed a programmable version of the Hardy Cross method. However, this method is not widely used because it required (1) the identification of loops, (2) the use of pseudo-loops if the network has more than one source, and (3) the finding of a set of initial flows that satisfies continuity. Later, Creaco and Franchini (2013) incorporated the concept of minimum cycle basis to identify a set of loops that can be used to achieve the sparsest key matrix for loop formulation. It is reported in their paper that, although the loop method requires less computation time than the GGA, the time taken for identifying the minimum cycle basis can be a major disadvantage. More recently, Alvarruiz et al. (2015) presented two methods to identify the minimum cycle basis that used significantly less time.

The null space method is a special loop-based method: all null space formulations can be rewritten as loop-based formulations, but not all loop-based formulations can be rewritten as null space formulations. The co-tree flows method (CTM) is the first null space method, which partitions the network component into a spanning tree and a co-tree. The CTM has the same disadvantages as the loop flow correction method. Later, the reformulated co-tree flow method (RCTM) was introduced by Elhay et al. (2014) to address the initialization requirements by incorporating Schilders' factorizations (Schilders 2009). Abraham and Stoianov (2015) proposed a partial update method for the null space methods, that is also applicable to the GGA, in which computation time is saved through minimizing the calculation of the head loss component by only calculating the friction factors and the head loss components of the pipes that have not satisfied the stopping test.

In this paper, a new graph partitioning algorithm, referred to as the bridge-block partitioning algorithm (BBPA), is proposed. The BBPA begins by using the FCPA to separate the core component from the forest component. Then, the BBPA further partitions the core component of the network into block and bridge components. Bridge components can be defined as the pipes in the core that are not part of any loop. For example, in Fig. 5.1(a) the bridge pipes are highlighted in bold. The solutions for these bridge components (block 1, block 3, and block 4) can be found by a linear process – as can the forest component in the FCPA. The remainder of the network consists of blocks, labeled **block 2** and **block 5** and solutions for these components can be found separately. It is possible to separate two blocks with a single node called a cut-vertex. This scenario is



(a) Example A **(b)** Example B

Fig. 5.1. Two example networks of blocks, bridges, and cut-vertices

illustrated in Fig. 5.1(b). The node (cut-vertex 2) is a cut-vertex that separates the two blocks. These two blocks can also also be solved separately, as was the case in part (a) of the example. The advantages in speed and reliability for the BBPA arise, in part, from the smaller systems that result from partitioning the network into these smaller blocks if the core component of the WDS graph is one-connected.

The BBPA exploits the fact the flows and heads in one block component are weakly coupled with these of the other block components and the solution of the flows and heads in a bridge component is a linear process. The convergence rate for the solution of the core component of a WDS, without the BBPA, is restricted to that of the worst block of the network. Solving each block separately reduces the number of iterations executed to the number of iterations required by that block.

There is a number of advantages to using the BBPA to identify the linear bridge components and the block components of a WDS network:

1. The number of iterations required by each block is bounded by that required by the unpartitioned system – solving the flows and heads in each block separately significantly reduces the overall computational time for the non-linear solver in almost all cases.
2. It improves the numerical reliability of the solution. The numerical reliability of the solution can be determined by the condition number of the Schur complement. The condition number of a matrix is the ratio of the largest to the smallest singular value of any square matrix. A rough rule of thumb is: one digit of reliability in the solution is lost for every power of ten in the condition number. If a square matrix is partitioned into block diagonal form by orthogonal permutations, the condition numbers of blocks can be no greater than that of the full matrix. In most cases, the condition numbers for all the individual blocks will be smaller than the condition number of the full matrix. This phenomenon is illustrated later in this paper.
3. It reduces the need to regularize for the presence of zero flows (Elhay and Simpson 2011). It has been pointed out by Simpson et al. (2012) that solving for the flows and heads separately can avoid the numerical failure that occurs when there are nodes with zero demand present in the forest. It is shown in this paper that there are blocks, in some networks, that have zero accumulative demands. The solutions of these networks need a regularization method to deal with the presence of the zero flows

to avoid catastrophic numerical failure when the Hazen-William head loss model is used. Using the BBPA avoids this failure which reduces the need for regularization.

4. It reduces the computational time in a management setting because the flows in the blocks with unchanged nodal demands do not need to be solved again and the heads in the corresponding block only need to be adjusted a posteriori.
5. The solution of each block can be found in parallel in a demand-driven model because the flows and heads in one block component can be found separately from those of the other block components.

The main contributions of this paper are: (1) to extend the concept of using bridge and block components in the loop flow correction method, proposed in Deuerlein (2008), to a generalized graph partitioning algorithm that can be used with any demand-driven WDS solution method, (2) to establish the theoretical advantages of using the BBPA in terms of reducing computational load and improving numerical reliability, (3) to provide a detailed case study to demonstrate BBPA's usefulness in terms of performance and accuracy.

This paper is organized as follows. Some definitions and notations are given in the next section. The section following provides the derivation of the method with some examples. The algorithmic description of the BBPA is then given, followed by the a discussion of the relation of the BBPA and other methods. This is followed by a benchmark analysis of the BBPA applied to the eight case study networks that supports the claim about the advantages of using the BBPA. These results are then discussed in the section that follows. Finally, the last section summarizes the overall findings.

5.4 General WDS Demand-Driven Steady-State Problem

This section describes the general WDS demand-driven steady-state problem. The following starts with the basic definition and notations, followed by the system equations. Finally, the Global Gradient Algorithm, which is used as the hydraulic solver to separately solve each block, are shown.

5.4.1 Definitions and Notation

Consider a water distribution system that contains n_p pipes, n_j junctions, n_r fixed head nodes and n_f forest pipes and nodes. The $j-th$ pipe of the network can be characterized by its diameter D_j, length L_j, resistance factor r_j. The $i-th$ node of the network has two properties: its nodal demand d_i and its elevation head z_i.

Let $\boldsymbol{q} = (q_1, q_2, ...q_{n_p})^T$ denote the vector of unknown flows, $\boldsymbol{h} = (h_1, h_2, ...h_{n_j})^T$ denote the vector of unknown heads, $\boldsymbol{r} = (r_1, r_2, ...r_{n_p})^T$ denote the vector of resistance factors, $\boldsymbol{d} = (d_1, d_2, ...d_{n_j})^T$ denote the vector of nodal demands, $\boldsymbol{e_l} = (e_{l_1}, e_{l_2}....e_{l_{nr}})^T$ denote the vector of fixed head elevations.

The head loss exponent n is assumed to be dependent only on the head loss model: $n = 2$ for the Darcy-Weisbach head loss model and $n = 1.852$ for Hazen-Williams head loss model. The head loss within the pipe j, which connects the node i and the node k, is modelled by $h_i - h_k = r_j q_j |q_j|^{n-1}$. Denote by $\boldsymbol{G}(\boldsymbol{q}) \in \mathbb{R}^{n_p \times n_p}$, a diagonal square matrix with elements $[\boldsymbol{G}]_{jj} = r_j |q_j|^{n-1}$ for $j = 1, 2, ...n_p$. Denote by $\boldsymbol{F}(\boldsymbol{q}) \in \mathbb{R}^{n_p \times n_p}$, a diagonal square matrix where the j-th element on its diagonal $[\boldsymbol{F}]_{jj} = \frac{d}{dq_j}[\boldsymbol{G}]_{jj} q_j$. The matrix $\boldsymbol{A_1}$

is the full rank, unknown head, node-arc incidence matrix. The matrix $\boldsymbol{A_2}$ is the fixed-head node-arc incidence matrix.

5.4.2 System of Equations

The steady-state flows and heads in a WDS system are modeled by the demand-driven model (DDM) continuity equations (1) and the energy conservation equations (2):

$$-\boldsymbol{A_1}^T\boldsymbol{q} - \boldsymbol{d} = \boldsymbol{O} \tag{5.1}$$

$$\boldsymbol{G}(\boldsymbol{q})\boldsymbol{q} - \boldsymbol{A_1}\boldsymbol{h} - \boldsymbol{A_2}\boldsymbol{e}_l = \boldsymbol{O}, \tag{5.2}$$

which can be expressed as

$$\begin{pmatrix} \boldsymbol{G}(\boldsymbol{q}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{h} \end{pmatrix} - \begin{pmatrix} \boldsymbol{A_2}\boldsymbol{e}_l \\ \boldsymbol{d} \end{pmatrix} = 0, \tag{5.3}$$

where its Jacobian matrix used in the solution process is

$$\boldsymbol{J} = \begin{pmatrix} \boldsymbol{F}(\boldsymbol{q}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \tag{5.4}$$

and it is sometimes referred to as a nonlinear saddle point problem (Benzi et al. 2005).

This non-linear system is often solved by the Newton method, in which $\boldsymbol{q}^{(m+1)}$ and $\boldsymbol{h}^{(m+1)}$ are repeatedly computed from $\boldsymbol{q}^{(m)}$ and $\boldsymbol{h}^{(m)}$ by

$$\begin{pmatrix} \boldsymbol{F}^{(m)}(\boldsymbol{q}^{(m)}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}^{(m+1)} - \boldsymbol{q}^{(m)} \\ \boldsymbol{h}^{(m+1)} - \boldsymbol{h}^{(m)} \end{pmatrix} = - \begin{pmatrix} \boldsymbol{G}^{(m)}\boldsymbol{q}^{(m)} - \boldsymbol{A_1}\boldsymbol{h}^{(m)} - \boldsymbol{A_2}\boldsymbol{e}_l \\ -\boldsymbol{A_1}^T\boldsymbol{q}^{(m)} - \boldsymbol{d}, \end{pmatrix} \tag{5.5}$$

until the relative differences $\frac{||\boldsymbol{q}^{(m+1)} - \boldsymbol{q}^{(m)}||}{||\boldsymbol{q}^{(m+1)}||}$ and $\frac{||\boldsymbol{h}^{(m+1)} - \boldsymbol{h}^{(m)}||}{||\boldsymbol{h}^{(m+1)}||}$ are sufficiently small.

5.4.3 Global Gradient Algorithm

Todini and Pilati (1988) applied block elimination to Eq. (5.5) to yield a two-step Hazen-Williams only solver: Eq. (5.6) for the heads and Eq. (5.7) for the flows.

$$\boldsymbol{U}\boldsymbol{h}^{(m+1)} = \left\{-n\boldsymbol{d} + \boldsymbol{A_1}^T[(1-n)\boldsymbol{q}^{(m)} - \boldsymbol{G}^{-1}\boldsymbol{A_2}\boldsymbol{e}_l]\right\} \tag{5.6}$$

where $\boldsymbol{U} = \boldsymbol{A_1}^T\boldsymbol{G}^{-1}\boldsymbol{A_1}$ is the Schur complement, and

$$\boldsymbol{q}^{(m+1)} = \frac{1}{n}\left\{(n-1)\boldsymbol{q}^{(m)} + \boldsymbol{G}^{-1}(\boldsymbol{A_2}\boldsymbol{e}_l + \boldsymbol{A_1}\boldsymbol{h})\right\} \tag{5.7}$$

Later, Simpson and Elhay (2010) proposed

$$\boldsymbol{V}\boldsymbol{h}^{(m+1)} = -\boldsymbol{d} + \boldsymbol{A_1}^T\boldsymbol{F}^{-1}\left[(\boldsymbol{G} - \boldsymbol{F})\,\boldsymbol{q}^{(m)} - \boldsymbol{A_2}\boldsymbol{e}_l\right] \tag{5.8}$$

where $\boldsymbol{V} = \boldsymbol{A_1}^T\boldsymbol{F}^{-1}\boldsymbol{A_1}$ is the Schur complement, and

$$\boldsymbol{q}^{(m+1)} = \boldsymbol{q}^{(m)} + \boldsymbol{F}^{-1}\boldsymbol{A_1}\boldsymbol{h}^{(m+1)} - \boldsymbol{F}^{-1}\left[\boldsymbol{G}\boldsymbol{q}^{(m)} - \boldsymbol{A_2}\boldsymbol{e}_l\right] \tag{5.9}$$

as the generalized equations that can be applied when the head-loss is modeled by the Hazen-William equation or the Darcy-Weisbach equation. The correct Jacobian matrix with the formula for $\boldsymbol{F}$, when head loss is modeled by Darcy-Weisbach equation, can be found in Simpson and Elhay (2010). They showed that the use of the correct Jacobian matrix restores the quadratic rate of convergence.

5.5 Derivation of the Bridge-Block Partitioning Algorithm

The following terminology will be used in this paper. Associated with a WDS is a graph G=(V, E), where the elements of V are the nodes (vertices) of the graph G and elements of E are the pipes (links or edges) of the graph G. Every WDS can be divided into two subgraphs: a treed subgraph (forest) $G_f = \left(V_f, E_f\right)$ and a looped subgraph (core) $G_c = (V_c, E_c)$, so that $E_f \bigcup E_c = E$, $E_f \bigcap E_c = \emptyset$, $V_f \bigcup V_c = V$. A cut-vertex is a node in a WDS graph, the removal of which will increase the number of connected components, and a bridge is a pipe in a WDS graph, the removal of which will separate its two end nodes. A block is a maximal connected subgraph without a cut-vertex. A WDS graph can be decomposed into a tree of blocks, cut-vertices, and bridges called a block-cut tree (Diestel 2005). A root block is a block which includes one or more water sources. Note that every water source is defined to be within the root block of its network component. That is, all water sources are in the root block of their connected component of the network. The level of block i in a rooted block-cut tree is the length of the unique path, composed of blocks, from the root block to block i. The parent of block i is the block connected to block i on the path to the root block. If block i is the parent of block j, then block j is the child of block i. A block of a graph G containing only one cut-vertex is called an end block of G. Note that any block except for the root block has a unique parent block, and any block except for an end block can have multiple child blocks.

A WDS graph can be divided into n_b subgraphs, G_{b_1}=(V_{b_1},E_{b_1}), G_{b_2}=(V_{b_2},E_{b_2}), ..., $G_{b_{n_b}}$ = ($V_{b_{n_b}}$,$E_{b_{n_b}}$). If two blocks, G_{b_i}=(E_{b_i},V_{b_i}) and G_{b_j}=(E_{b_j},V_{b_j}), are adjacent, then $E_{b_i} \cap E_{b_j} = \emptyset$ and $V_{b_i} \cap V_{b_j} = c_{ij}$ where c_{ij} is the cut-vertex that connects the parent block i and child block j. The cut-vertex, c_{ij}, in the parent block, b_i, is a cluster of the demands of this cut-vertex and all its descendant blocks. A block except for the end block can have multiple cut-vertices behaving as clusters of demands because a parent block can have multiple child blocks. The cut-vertex, c_{ij}, in the child block, b_j, is considered as a pseudo-source. The head of the cut-vertex, c_{ij}, that is found in the parent block, b_i, is used as the elevation head of the pseudo-source for the corresponding child block. With the exception of the root block, every block has a single cut-vertex that behaves as a pseudo-source. The ancestors of a block are the blocks in the path from the root block to this block, excluding the block itself and including the root block. The descendants of block i are all the blocks that have block i as an ancestor.

The BBPA is now derived by generating two orthogonal permutation matrices and using them to manipulate the matrix $\boldsymbol{A_1}$ to find n_b unknown-head node-arc incidence matrices for each block, $\boldsymbol{B_{11}}$, $\boldsymbol{B_{22}}$, ...$\boldsymbol{B_{n_b n_b}}$, and $n_b - 1$ fixed head node-arc incidence matrices, $\boldsymbol{C_1}$, $\boldsymbol{C_2}$, ..., $\boldsymbol{C_{n_b-1}}$. Note that in the following, $\boldsymbol{B_{ij}}$, the block in the i-th block row and the j-th block column, is used to denote the fixed head node-arc incidence matrices, where the subscripts i and j are used to indicate the location of the block, row j and column i, and also to indicate a direct connection between the block i and block j.

Recall that all blocks except for the root block have exactly one cut-vertex that behaves as a pseudo-source. The terms involving these pseudo-sources are moved to the right-hand-side of the system leaving the remaining node-arc incidence matrix full rank. This is because each of the diagonal block matrices of $\boldsymbol{A_1}$, a full rank matrix, is also full rank.

The permutation matrix that is used to permute the system equation, Eq. (5.3), is

$$\boldsymbol{P_1} = \begin{matrix} \\ n_p \\ n_j \end{matrix} \begin{matrix} \begin{matrix} n_p & n_j \end{matrix} \\ \begin{pmatrix} \boldsymbol{P} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{R} \end{pmatrix} \end{matrix}, \tag{5.10}$$

where $\boldsymbol{P} = \begin{pmatrix} \boldsymbol{P}_{e_{b_1}} & \boldsymbol{P}_{e_{b_2}} & \dots \boldsymbol{P}_{e_{n_b}} \end{pmatrix}^T \in \mathbb{Z}^{n_p \times n_p}$ is the square orthogonal permutation matrix for the pipes in each block, in which $\boldsymbol{P}_{e_{b_i}} \in \mathbb{Z}^{n_p \times n_{p_{b_i}}}$, for $i = 1, 2, ...n_b$, is the permutation matrix that identifies the pipes in the block i as distinct from the pipes in other blocks and $\boldsymbol{R} = \begin{pmatrix} \boldsymbol{R}_{v_{b_1}} & \boldsymbol{R}_{v_{b_2}} & \dots \boldsymbol{R}_{v_{n_b}} \end{pmatrix}^T \in \mathbb{Z}^{n_j \times n_j}$ is the square orthogonal permutation matrix for the nodes in each block, in which $\boldsymbol{R}_{v_{b_i}} \in \mathbb{R}^{n_j \times n_{v_{b_i}}}$, for $i = 1, 2, ...n_b$, is the permutation matrix that identifies the nodes in the block i as distinct from the nodes in other blocks.

The permuted system of the BBPA equations is:

$$\boldsymbol{P_1} \begin{pmatrix} \boldsymbol{G} & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \boldsymbol{P_1}^T \boldsymbol{P_1} \begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{h} \end{pmatrix} - \boldsymbol{P_1} \begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{d} \end{pmatrix} = \boldsymbol{O} \tag{5.11}$$

where $\boldsymbol{a} = \boldsymbol{A_2 e_l}$. With this permutation, Eq. (5.3) becomes:

$$\begin{pmatrix} \boldsymbol{P G P}^T & -\boldsymbol{P A_1 R}^T \\ -\boldsymbol{R A_1}^T \boldsymbol{P}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{Pq} \\ \boldsymbol{Rh} \end{pmatrix} - \begin{pmatrix} \boldsymbol{Pa} \\ \boldsymbol{Rd} \end{pmatrix} = \boldsymbol{O} \tag{5.12}$$

where

$$\boldsymbol{P A_1 R}^T = \begin{pmatrix} \boldsymbol{B_{11}} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \boldsymbol{B_{21}} & \boldsymbol{B_{22}} & \dots & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{B_{n_b 1}} & \boldsymbol{B_{n_b 2}} & \dots & \boldsymbol{B_{n_b n_b}} \end{pmatrix},$$

in which all the block entries above the diagonal blocks become zero matrices because there is no pipe in a parent block that connects to any node in any of its child blocks. The block entries below the diagonal blocks, $\boldsymbol{B_{ij}}$ represent the connection between the nodes in the parent block, block j, and the pipes in the child block, block i, which are $\boldsymbol{O}$ when block j and block i are not adjacent blocks. It has been pointed out above that any block, except for the end block, can have multiple child blocks. Furthermore, any block, except for the root block, can have only one parent block. As a result, each block column can have more than two non-zero block entries (including the diagonal block in that block column) and each block row, except for the root block row, has exactly two non-zero block entries (including the diagonal block in that block row).

$$\boldsymbol{P G P}^T = \begin{pmatrix} \boldsymbol{G}_{b_1} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{G}_{b_2} & \dots & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{G}_{b_{n_b}} \end{pmatrix}, \boldsymbol{Pq} = \begin{pmatrix} \boldsymbol{q}_{b_1} \\ \boldsymbol{q}_{b_2} \\ \vdots \\ \boldsymbol{q}_{b_{n_b}} \end{pmatrix}, \boldsymbol{Rh} = \begin{pmatrix} \boldsymbol{h}_{b_1} \\ \boldsymbol{h}_{b_2} \\ \vdots \\ \boldsymbol{h}_{b_{n_b}} \end{pmatrix}, \boldsymbol{Pa} = \begin{pmatrix} \boldsymbol{a}_{b_1} \\ \boldsymbol{a}_{b_2} \\ \vdots \\ \boldsymbol{a}_{b_{n_b}} \end{pmatrix}$$

in which any block that is not a root block becomes $\boldsymbol{O}$, and $\boldsymbol{Rd} = \begin{pmatrix} \boldsymbol{d}_{b_1}^T & \boldsymbol{d}_{b_2}^T & \dots & \boldsymbol{d}_{b_{n_b}}^T \end{pmatrix}^T$. The matrix $\boldsymbol{P A_1 R}^T$ in Eq. (5.12) can be divided into two block matrices: a block diagonal

matrix:

$$\boldsymbol{A_B} = \begin{pmatrix} \boldsymbol{B_{11}} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{B_{22}} & \dots & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{B_{n_b n_b}} \end{pmatrix}, \tag{5.13}$$

where each block matrix on its block diagonal represents the node-arc incidence matrix of the corresponding graph block, and a lower block triangular matrix that only has entries below its block diagonal:

$$\boldsymbol{A_C} = \begin{pmatrix} \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \boldsymbol{B_{21}} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{-B_{n_b 1}} & \boldsymbol{-B_{n_b 2}} & \dots & \boldsymbol{O} \end{pmatrix}, \tag{5.14}$$

where each matrix block represents the connection from the cut-vertex acting as a pseudo-source to a child block (row) and the connection from the same cut-vertex acting as a cluster of demand nodes to the parent block (column). Recall that, each block row of the matrix $\boldsymbol{A_B} + \boldsymbol{A_C}$, except for the block row representing the root graph blocks, has exactly two non-zero block entries: one of two non-zero block entries is on the block diagonal of the matrix $\boldsymbol{A_B}$ and the other one of the two non-zero block entries is in the lower triangular part of the matrix $\boldsymbol{A_C}$.

Defining $\boldsymbol{G}_B = \boldsymbol{P}\boldsymbol{G}\boldsymbol{P}^T$, $\boldsymbol{q}_B = \boldsymbol{P}\boldsymbol{q}$, $\boldsymbol{h}_B = \boldsymbol{R}\boldsymbol{h}$, $\boldsymbol{a_B} = \boldsymbol{P}\boldsymbol{a}$, and $\boldsymbol{d}_B = \boldsymbol{R}\boldsymbol{d}$, Eq. (5.12) can be rewritten as

$$\begin{pmatrix} \boldsymbol{G}_B & \boldsymbol{-A_C - A_B} \\ \boldsymbol{-A_C}^T \boldsymbol{- A_B}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_B \\ \boldsymbol{h}_B \end{pmatrix} = \begin{pmatrix} \boldsymbol{a}_B \\ \boldsymbol{d}_B \end{pmatrix}. \tag{5.15}$$

The matrix $\boldsymbol{A_C}$ can be moved from the left-hand-side of Eq. (5.15) to its right-hand-side and Eq. (5.15) becomes:

$$\begin{pmatrix} \boldsymbol{G}_B & \boldsymbol{-A_B} \\ \boldsymbol{-A_B}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_B \\ \boldsymbol{h}_B \end{pmatrix} = \begin{pmatrix} \boldsymbol{a}_B + \boldsymbol{A_C h}_B \\ \boldsymbol{d}_B + \boldsymbol{A_C}^T \boldsymbol{q}_B \end{pmatrix} \tag{5.16}$$

Defining $\widehat{\boldsymbol{a}}_B = \boldsymbol{a_B} + \boldsymbol{A_C h}_B$ and $\widehat{\boldsymbol{d}}_B = \boldsymbol{d}_B + \boldsymbol{A_C}^T \boldsymbol{q}_B$, Eq. (5.16) expands to

$$\left(\begin{array}{cccc|cccc} \boldsymbol{G}_{b_1} & \boldsymbol{O} & \dots & \boldsymbol{O} & \boldsymbol{-B_{11}} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{G}_{b_2} & \dots & \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{B_{22}} & \dots & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{G}_{b_{n_b}} & \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{B_{n_b n_b}} \\ \hline \boldsymbol{-B_{11}}^T & \boldsymbol{O} & \dots & \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{-B_{22}}^T & \dots & \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots & \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{-B_{n_b n_b}}^T & \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{O} \end{array}\right) \left(\begin{array}{c} \boldsymbol{q}_{b_1} \\ \boldsymbol{q}_{b_2} \\ \vdots \\ \boldsymbol{q}_{b_{n_b}} \\ \boldsymbol{h}_{b_1} \\ \boldsymbol{h}_{b_2} \\ \vdots \\ \boldsymbol{h}_{b_{n_b}} \end{array}\right) = \left(\begin{array}{c} \widehat{\boldsymbol{a}}_{\boldsymbol{b_1}} \\ \widehat{\boldsymbol{a}}_{\boldsymbol{b_2}} \\ \vdots \\ \widehat{\boldsymbol{a}}_{\boldsymbol{b_{n_b}}} \\ \widehat{\boldsymbol{d}}_{b_1} \\ \widehat{\boldsymbol{d}}_{b_2} \\ \vdots \\ \widehat{\boldsymbol{d}}_{b_{n_b}} \end{array}\right). \tag{5.17}$$

It is evident from the expanded system of equations, Eq. (5.17), of the BBPA that the amount of computation can be significantly reduced by solving each block separately. Moreover, these blocks can be solved in parallel. This is because the permuted $\boldsymbol{A_1}$ matrix, $\boldsymbol{P}\boldsymbol{A_1}\boldsymbol{R}^T$, can be rearranged into a block diagonal matrix, which allows the non-linear system of equations in Eq. (5.3) be partitioned into n_b smaller independent non-linear systems.

5.5.1 Update of the demands and nodal heads

The demands for each block are only required to be updated once before every evaluation and the head for each unknown-head node is only required to be updated once after the solution of each block is found. As stated previously, each block row of the matrix $\boldsymbol{A_C}$ has only one non-zero block entry below its block diagonal. The matrix $\boldsymbol{B_{ij}}$ only has one column entry that is non-zero. This column entry is the $\boldsymbol{A_2}$ matrix for that block, which is the node-arc incidence matrix representing the connection between the pseudo-source and the pipes in the child block.

LEMMA 2. *Suppose $\boldsymbol{v} \in \mathbb{R}^{n_j \times 1}$ is a column vector of all ones $\boldsymbol{A_1} \in \mathbb{R}^{n_p \times n_j}$, is an unknown-head node-arc incidence matrix and $\boldsymbol{A_2} \in \mathbb{R}^{n_p \times 1}$ is a fixed-head node-arc incidence matrix for one of the WDS's blocks that is not the root block. Then*

$$-\boldsymbol{A_1 v} = \mathbf{A_2} \tag{5.18}$$

Proof. Denote by p_1, a set of indices for the pipes that are not connected to a water source; by p_2, a set of indices for the pipes that are connected to a water source. Let $\boldsymbol{A_1} = \begin{pmatrix} \boldsymbol{a_1}^T & \boldsymbol{a_2}^T & \dots & \boldsymbol{a_{n_p}}^T \end{pmatrix}^T$. The i-th row of the matrix $\boldsymbol{A_1}$ has two non-zero entries, 1 and -1, and the i-th row of the matrix $\boldsymbol{A_2}$ is zero if $i \in p_1$. It is evident that the inner product of $\boldsymbol{a_i}$ and $\boldsymbol{v}$ becomes 0. The j-th row of the matrix $\boldsymbol{A_1}$ has only one entry, -1, and the j-th row of the matrix $\boldsymbol{A_2}$ has only one entry, 1, if $j \in p_2$. It is evident that the inner product of $\boldsymbol{a_j}$ and $\boldsymbol{v}^T$ is -1. End of LEMMA 2□

The relationship shown in Eq. (5.18) can be used to calculate term $\boldsymbol{A_c h}_B$ in Eq. (5.16). The relationship between the unknown head node-arc incidence matrix, $\boldsymbol{B_{ii}}$, and the fixed head node-arc incidence matrix, $\boldsymbol{B_{ij}}$, is

$$\boldsymbol{B_{ij}} = -\boldsymbol{B_{ii} v}, \tag{5.19}$$

the transpose of which is $\boldsymbol{B_{ij}}^T = -\boldsymbol{v}^T \boldsymbol{B_{ii}}^T$ and multiplying both sides by $\boldsymbol{q_{B_j}}$, the flows in block j , we get $\boldsymbol{B_{ij}}^T \boldsymbol{q_{B_j}} = -\boldsymbol{v}^T \boldsymbol{B_{ii}}^T \boldsymbol{q_{B_j}}$. Therefore,

$$\boldsymbol{B_{ij}}^T \boldsymbol{q_{B_j}} = -\boldsymbol{v}^T \boldsymbol{d_{B_j}}, \tag{5.20}$$

which is in fact the sum of the demands in the child block to the cut-vertex in the parent block. Eq. (5.20) is used repeatedly from the end block to the root block until the $\boldsymbol{A_c}^T \boldsymbol{q}_B$ in Eq. (5.16) has been replaced. This process is performed only once before the iterative phase.

Multiplying both sides of the Eq. (5.19) by the unknown head at cut-vertex c_j, $\boldsymbol{h}_{c_j}$, we get

$$\boldsymbol{B_{ij} h}_{c_j} = -\boldsymbol{B_{ii} v h}_{c_j}, \tag{5.21}$$

which is used to move $\boldsymbol{A_c h}_b$ from the left-hand-side of the equation to the right-hand-side of the equation so that each block can be solved in parallel. The heads need only be computed just once after the iterations for all blocks have been completed.

5.5.2 The properties of the system of equations after bridge-block partitioning

In the BBPA, a full WDS network is partitioned into n_b smaller independent non-linear systems by permuting the original full system of equations using two orthogonal permutations $\boldsymbol{P}$ and $\boldsymbol{R}$. One of the main contributions of this paper is to show that the use of the BBPA can significantly reduce the computational loads and improve the numerical reliability of the results.

The BBPA can be used to improve the reliability of solution of the looped component in the final WDS solution. This is because the condition number, the ratio between the largest to the smallest singular value of a matrix, can be used to estimate the loss of reliable digits in solving a linear system with that matrix. The orthogonal permutations of the BBPA shuffle the n_j singular values of the Schur Complement into their corresponding blocks. This is because pre-and-post-multiplying a matrix by orthogonal matrices preserves the singular values. The upper bound of the largest singular value of all blocks is the largest singular value of the full system and the lower bound for the smallest singular value of all blocks is the smallest singular value for the full system. Therefore, the condition number of each block at the solution is bounded above by the condition number of the full system of equations but in most cases will be smaller. Moreover, the only occasions when one of the blocks has the same condition number as the full system is where both the highest and lowest singular values are present in the *same* block. Even in this particular case the other blocks in the system will have lower condition numbers than the full system.

Furthermore, the use of the BBPA can minimize the need to use regularization methods for handling zero-flows. In the FCPA paper (Simpson et al. 2012), the authors pointed out that it is common for zero flows to occur at the ends of trees with zero demands. Similarly, it is also possible for all nodes in the end blocks to have zero demands. The GGA fails catastrophically at these blocks when the head loss is modelled by the Hazen-William head loss model. One side-effect of identifying these end blocks with zero nodal demands is zero flows can be assigned to all pipes in these blocks and the head of pseudo-source can be assigned to all nodes in these blocks. When zero flows occur in other blocks, regularization is needed only for the blocks with the presence of zero flows instead of the full system.

In addition to the improvement of the numerical reliability of the final result, the use of the BBPA can significantly reduce computational loads. This reduction in computational loads is achieved through: (1) the bridge component being solved by a linear process, the removal of which reduces the number of non-zeroes in Schur component, (2) the probable reduction in the number iterations required by each block as shown in Appendix in section 5.13, and (3) the non-linear system of equations for each block is independent of other blocks which allows each block to be solved in parallel.

5.6 Bridge-Block Partitioning Algorithm

The steps of the BBPA are now described. The BBPA starts with a forest search algorithm to identify the forest component as distinct from the core. This is followed by identifying all the blocks and bridges in the core, and updating the demands for the cut-vertices by using Stage 3 as given below, a variation of the algorithm detailed by Hopcroft and Tarjan (1973). Note that this algorithm is based on the depth-first search and runs in linear time. There are two ways to solve the core of the network: in parallel or serially.

Parallel: It can be more efficient to solve all the blocks in parallel when the solution of the entire system is needed, such as in a design setting. After the network has been permuted, each block is then individually solved by using Stage 4 in no particular order. Once the solutions for all blocks are found, the heads for the core nodes are recovered by using Stage 5 from the root block to the end blocks. Finally, the heads for the forest nodes are solved.

Stage 3: Bridge block partitioning determination & bottom-up demand adjustment

```
/* Serial determination of network block from the end blocks to the root
   blocks and bottom-up cut-vertex demand accumulation                  */
input  : Adjacency List and d
output : The system of equations of all blocks
Procedure DFS(currentNode,d,dm)
    visited[currentNode] = true;
    d=d+1;
    depth[currentNode] = d;
    low[currentNode] = d;
    foreach (nextNode,nextPipe)∈ adjList(currentNode) do
        if nextNode is not a Forest node then
            if nextNode is not visited then
                stack.push_back(adjList[currentNode]);
                parent[nextnode]=nextpipe;
                DFS(nextnode,d,dm);
                if low[currentNode]≥ depth[currentNode] then
                    BlockSource[N_B].push_back(currentNode);
                    do
                        temp←stack->pop_back();
                        if (temp.first<np) then
                            BlockPipe[N_B].insert(temp.first);
                        end if
                        if (temp.second!=currentNode) then
                            if (temp.second<nj) then
                                if (BlockNode[N_B].insert(temp.second).second==true)
                                then
                                    (*dm)[currentNode]+=(*dm)[temp.second];
                                end if
                            else if (temp.second>=nj) then
                                BlockSource[N_B].insert(temp.second);
                        end if
                    while (temp.first!=nextPipe);
                    N_B = N_B + 1;
                end if
                low[currentNode]=min(low[nextNode],low[currentNode]);
            else if (parent[currentNode]≠nextpipe&&depth[nextnode]<depth[currentNode])
            then
                stack.push_back(adjList[currentNode]);
                low[currentNode]=min(low[currentNode],depth[nextnode]);
        end if
    end foreach
Algorithm BBPA()
    for currentNode← n_j to n_j + n_f do
        DFS(currentNode,depth,dm);
    end for
    Initialize the system of equations for each block using Eq. (5.16);
```

Stage 4: Serial or parallel block solution

```
/* Nonlinear solution for the blocks can either be found serially or in
   parallel                                                          */
```

input : The system of equations for a block
output : The solution of the flows in the input block and heads that need to be updated

```
1  foreach Block do
2      if The size of the block =1 then
3          This block is a bridge and assign the demand of the only node to the flow of
           the only pipe;
4      else
5          if Sum of the demands in this block=0 then
6              assign the flows of the pipes=0;
7              continue
8          endif
9          Using a WDS solution method to solve the nonlinear system for the flows and
           interim heads.;
10     endif
11 end foreach
```

Stage 5: Top-down head correction

```
/* Top-down determinations of corrected heads from the relative heads.  Actual
   heads in any block can only be found when the flows and interim heads of its
   ancestor blocks have been found                                          */
```

input : The unrecovered heads of a block
input : The head of the pseudo-source from the parent block of the current block
output : The recovered heads of the input block

```
1  foreach Block do
2      if The input block is not the root bock then
3          Recover the actual heads of the input block from the interim heads by using
           Eq. (5.21).
4      endif
5  end foreach
```

Serial: Alternatively, each of the blocks can be solved in sequence. After the network has been permuted, each block is then separately solved by using Stage 4 and Stage 5 from the root block to the end blocks. Note that it is possible to only solve for a part of the system which includes the blocks of interest and their ancestor blocks. Solving the system this way reduces the computational time in a management setting because (1) the flows in the blocks with unchanged nodal demands do not need to be solved again and the heads in the corresponding block only need to be adjusted a posteriori and (2) a different priority can be assigned to a different block which allows blocks with different priorities to be updated in a different time interval. Finally, the heads for the forest nodes are solved.

5.7 Example

In this section, the use of the BBPA is demonstrated by applying it to the example network shown in Fig. 5.2. The system of equations for each block are displayed. This network has eight pipes, six nodes with unknown heads, and one water source. The solution for this example is demonstrated below in two steps: (1) network permutation and (2) network solution.

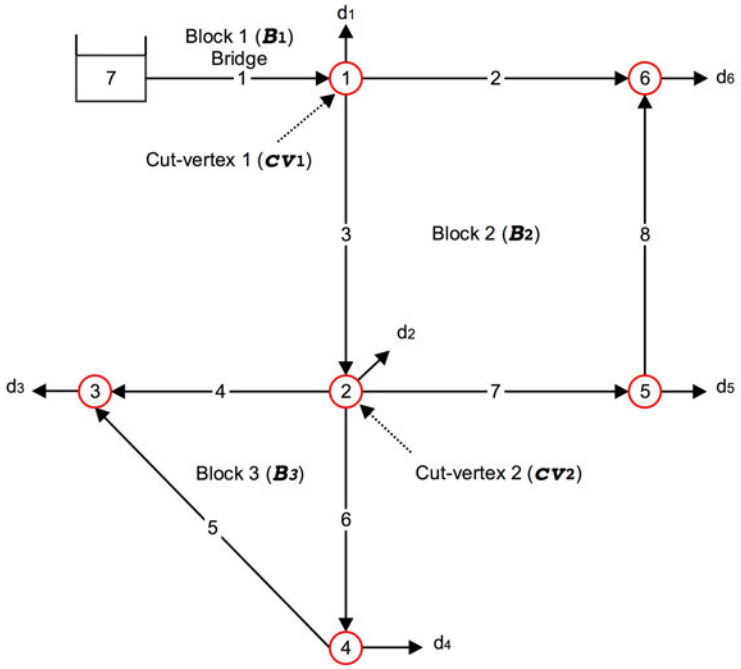


Fig. 5.2. A simple example network that is made up of three blocks, and two cut-vertices. Block 1 is referred to as $\boldsymbol{B_1}$, Block 2 is referred to as $\boldsymbol{B_2}$, and Block 3 is referred to as $\boldsymbol{B_3}$. Cut-vertex 1 is referred to as $\boldsymbol{cv_1}$ and Cut-vertex 2 is referred to as $\boldsymbol{cv_2}$.

5.7.1 Permutation for example network

The unknown-head node-arc incidence matrix, $\boldsymbol{A_1}$, and the fixed-head node-arc incidence matrix, $\boldsymbol{A_2}$ for this example network are

$$\boldsymbol{A_1} = \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix}, \boldsymbol{A_2} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

The system of pipe head loss and nodal continuity equations for the example network is

$$\begin{pmatrix} G_1 & & & & & & & & 1 & 0 & 0 & 0 & 0 & 0 \\ & G_2 & & & & & & & -1 & 0 & 0 & 0 & 0 & 1 \\ & & G_3 & & & & & & -1 & 1 & 0 & 0 & 0 & 0 \\ & & & G_4 & & & & & 0 & -1 & 1 & 0 & 0 & 0 \\ & & & & G_5 & & & & 0 & 0 & 1 & -1 & 0 & 0 \\ & & & & & G_6 & & & 0 & -1 & 0 & 1 & 0 & 0 \\ & & & & & & G_7 & & 0 & -1 & 0 & 0 & 1 & 0 \\ & & & & & & & G_8 & 0 & 0 & 0 & 0 & -1 & 1 \\ 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & & & & & & \\ 0 & 0 & 1 & -1 & 0 & -1 & -1 & 0 & & & & & & \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & & & & & & \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & & & & & & \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & & & & & & \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \\ q_7 \\ q_8 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \end{pmatrix} = \begin{pmatrix} e_{l_7} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{pmatrix}. \tag{5.22}$$

By permuting the rows (pipes) in the ordering given by $p = \{1; 2; 3; 7; 8; 4; 5; 6\}$ and the columns (nodes) in the ordering given by $v = \{1; 6; 2; 5; 3; 4\}$, the system of equations in Eq. (5.22) can be rearranged into the following block structure:

$$\begin{array}{cc|cccc|cccc|cccc|cc|cc} & & \multicolumn{8}{c|}{\boldsymbol{Pipes}} & \multicolumn{6}{c|}{\boldsymbol{Nodes}} & & \\ & \textbf{Block} & \multicolumn{2}{c}{\boldsymbol{B_1}} & \multicolumn{3}{c}{\boldsymbol{B_2}} & \multicolumn{3}{c|}{\boldsymbol{B_3}} & \multicolumn{2}{c}{\boldsymbol{B_1}} & \multicolumn{2}{c}{\boldsymbol{B_2}} & \multicolumn{2}{c|}{\boldsymbol{B_3}} & & \\ \multirow{3}{*}{\rotatebox{90}{\boldsymbol{Pipes}}} & \boldsymbol{B_1} & G_1 & & & & & & & & \mathbf{1} & 0 & \mathbf{0} & 0 & 0 & 0 & q_1 & e_{l_7} \\ & & & G_2 & & & & & & & \mathbf{-1} & 1 & \mathbf{0} & 0 & 0 & 0 & q_2 & 0 \\ & \boldsymbol{B_2} & & & G_3 & & & & & & \mathbf{-1} & 0 & \mathbf{1} & 0 & 0 & 0 & q_3 & 0 \\ & & & & & G_7 & & & & & \mathbf{0} & 0 & \mathbf{-1} & 1 & 0 & 0 & q_7 & 0 \\ & & & & & & G_8 & & & & \mathbf{0} & 1 & \mathbf{0} & -1 & 0 & 0 & q_8 & 0 \\ & & & & & & & G_4 & & & \mathbf{0} & 0 & \mathbf{-1} & 0 & 1 & 0 & q_4 & 0 \\ & \boldsymbol{B_3} & & & & & & & G_5 & & \mathbf{0} & 0 & \mathbf{0} & 0 & 1 & -1 & q_5 & 0 \\ & & & & & & & & & G_6 & \mathbf{0} & 0 & \mathbf{-1} & 0 & 0 & 1 & q_6 & 0 \\ \multirow{3}{*}{\rotatebox{90}{\boldsymbol{Nodes}}} & \boldsymbol{B_1} & \mathbf{1} & \mathbf{-1} & \mathbf{-1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & & & & & & & h_1 & d_1 \\ & \boldsymbol{B_2} & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & & & & & & & h_6 & d_6 \\ & & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{-1} & \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{-1} & & & & & & & h_2 & d_2 \\ & & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & & & & & & & h_5 & d_5 \\ & \boldsymbol{B_3} & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & & & & & & & h_3 & d_3 \\ & & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & & & & & & & h_4 & d_4 \end{array} \tag{5.23}$$

*the bold numbers in the matrix represent the cut-vertices

Eq. (5.23) has three graph blocks as shown in Fig. 5.2 include Block 1 (a bridge), Block 2, and Block 3. Note that, for cross-referencing purposes, this equation has been labeled with the block numbers (affiliated with pipes and nodes) corresponding to each entity in the example network. The cut-vertices ($\boldsymbol{cv_1}$ and $\boldsymbol{cv_2}$ in Fig. 5.2) are highlighted in bold in their corresponding matrix blocks. In the equation, it is evident that the permuted $\boldsymbol{A_1}$ matrix is a block three by three, lower block triangular matrix which represents a WDS with the three graph blocks ($\boldsymbol{B_1}$, $\boldsymbol{B_2}$, and $\boldsymbol{B_3}$).

The end block ($\boldsymbol{B_3}$ in Fig. 5.2) is a sub-network consisting of three pipes {4; 5; 6}, two nodes {3; 4}, and a pseudo-source at node {2}. The nodal demands of this block do not need to be updated because this is the end block. The head of the node 2 ($\boldsymbol{cv_2}$) , which is the cut-vertex behaving as the pseudo-source for this block, can be moved to the right-hand-side of system of equations using Eq. (5.16). The solution of block $\boldsymbol{B_3}$ can be found separately after the head of the pseudo-source at node {2} is found.

The second block diagonal row ($\boldsymbol{B_2}$ in Fig. 5.2) is a sub-network consisting of four pipes {2; 3; 7; 8}, three nodes {2; 5; 6}, and one pseudo-source at node {1}. This is an intermediate block so that the demand at the node 2 ($\boldsymbol{cv_2}$), a cut-vertex that is not a pseudo-source, needs to be updated by increasing its demand by the sum of demands at all nodes of its child block ($\boldsymbol{B_3}$) as follows: $\widehat{d_2} = d_2 + d_3 + d_4$ using Eq. (5.20). Node 1 ($\boldsymbol{cv_1}$),

which is the cut-vertex behaving as the pseudo-source for this block, $\boldsymbol{B_2}$, can be moved to the right-hand-side of system of equations using Eq. (5.16). The solution of block $\boldsymbol{B_2}$ can be found separately after the head of the pseudo-source at node $\{1\}$ is found.

Finally, the root block ($\boldsymbol{B_1}$ in Fig. 5.2) is a sub-network consisting of pipe $\{1\}$, node $\{1\}$, and source $\{7\}$. Block B_1 is a bridge component. The bridge component can be solved by using a linear process. The demand for the node 1 in Fig. 5.2 ($\boldsymbol{cv_1}$), a cut vertex in the root block, is updated by increasing its demand by the sum of demands at all nodes of its child block ($\boldsymbol{B_2}$) as follows: $\widehat{d_1} = d_1 + d_2 + d_3 + d_4 + d_5 + d_6$ and the elevation head for the source stays the same. After updating the demands and heads, the system of equations in Eq. (5.23) becomes:

$$
\begin{array}{cc|cccccccc|cccccc|cc|c}
 & & \multicolumn{8}{c|}{\textit{Pipes}} & \multicolumn{6}{c|}{\textit{Nodes}} & & & \\
 & \textbf{Block} & \boldsymbol{B_1} & \multicolumn{3}{c}{\boldsymbol{B_2}} & & \multicolumn{3}{c|}{\boldsymbol{B_3}} & \boldsymbol{B_1} & \multicolumn{2}{c}{\boldsymbol{B_2}} & & \multicolumn{2}{c|}{\boldsymbol{B_3}} & & & \\
\multirow{3}{*}{\textit{Pipes}} & \boldsymbol{B_1} & G_1 & & & & & & & & \mathbf{1} & 0 & \mathbf{0} & 0 & 0 & 0 & q_1 & = & e_{l_7} \\
 & \multirow{2}{*}{\boldsymbol{B_2}} & & G_2 & & & & & & & \mathbf{0} & 1 & \mathbf{0} & 0 & 0 & 0 & q_2 & & h_1 \\
 & & & & G_3 & & & & & & \mathbf{0} & 0 & \mathbf{1} & 0 & 0 & 0 & q_3 & & h_1 \\
 & & & & & G_7 & & & & & \mathbf{0} & 0 & \mathbf{-1} & 1 & 0 & 0 & q_7 & & 0 \\
 & & & & & & G_8 & & & & \mathbf{0} & 1 & \mathbf{0} & -1 & 0 & 0 & q_8 & & 0 \\
 & \multirow{2}{*}{\boldsymbol{B_3}} & & & & & & G_4 & & & \mathbf{0} & 0 & \mathbf{0} & 0 & 1 & 0 & q_4 & & h_2 \\
 & & & & & & & & G_5 & & \mathbf{0} & 0 & \mathbf{0} & 0 & 1 & -1 & q_5 & & 0 \\
 & & & & & & & & & G_6 & \mathbf{0} & 0 & \mathbf{0} & 0 & 0 & 1 & q_6 & & h_2 \\
\hline
\multirow{3}{*}{\textit{Nodes}} & \boldsymbol{B_1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & & & & & & & h_1 & & d_1+d_2+d_3+d_4+d_5+d_6 \\
\hline
 & \multirow{2}{*}{\boldsymbol{B_2}} & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & & & & & & & h_6 & & d_6 \\
 & & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{-1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & & & & & & & h_2 & & d_2+d_3+d_4 \\
 & & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & & & & & & & h_5 & & d_5 \\
\hline
 & \boldsymbol{B_3} & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & & & & & & & h_3 & & d_3 \\
 & & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & & & & & & & h_4 & & d_4
\end{array}
\tag{5.24}
$$

Note that the system of equations obtained in Eq. (5.24) is equivalent to performing block Gauss-Jordan elimination on Eq. (5.23). Solving the system of equations in this way requires solving each block in a particular sequence, from the root block ($\boldsymbol{B_1}$) to the end block ($\boldsymbol{B_3}$). The sequence that is required in the example network in Fig. 5.2 is: (1) to find the solution of block $\boldsymbol{B_1}$, the root block; (2) to find the solution of block $\boldsymbol{B_2}$ using the head of the node one, $\boldsymbol{cv_1}$, in block $\boldsymbol{B_1}$; and (3) to find the solution of block $\boldsymbol{B_3}$, the end block, using the head of the node two, $\boldsymbol{cv_2}$, in block $\boldsymbol{B_2}$.

Furthermore, the second pipe head-loss block equation or the second block equation ($\boldsymbol{B_2}$) in Eq. (5.24) is:

$$\boldsymbol{G_{b_2} q_{b_2} - B_{22} h_{b_2} = B_{21} h_{b_1}},$$

which expands to:

$$
\begin{pmatrix} G_2 & & & \\ & G_3 & & \\ & & G_7 & \\ & & & G_8 \end{pmatrix}
\begin{pmatrix} q_2 \\ q_3 \\ q_7 \\ q_8 \end{pmatrix}
+
\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{pmatrix}
\begin{pmatrix} h_6 \\ h_2 \\ h_5 \end{pmatrix}
=
\begin{pmatrix} h_1 \\ h_1 \\ 0 \\ 0 \end{pmatrix},
\tag{5.25}
$$

the right-hand-side of which can be rewritten as:

$$\boldsymbol{B_{21} h_{b_1} = -B_{22} [v_3 h_1]},
\tag{5.26}$$

which expands to:

$$
\begin{pmatrix} h_1 \\ h_1 \\ 0 \\ 0 \end{pmatrix}
=
\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{pmatrix}
\begin{pmatrix} h_1 \\ h_1 \\ h_1 \end{pmatrix}
$$

using Eq. (5.21). Substituting it back into Eq. (5.25), we get:

$$\boldsymbol{G_{b_2} q_{b_2}} - \boldsymbol{B_{22} h_{b_2}} = -\boldsymbol{B_{22}}[\boldsymbol{v}_3 h_1],$$

which expands to:

$$\begin{pmatrix} G_2 & & & \\ & G_3 & & \\ & & G_7 & \\ & & & G_8 \end{pmatrix} \begin{pmatrix} q_2 \\ q_3 \\ q_7 \\ q_8 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} h_6 \\ h_2 \\ h_5 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} h_1 \\ h_1 \\ h_1 \end{pmatrix},$$

which can further simplified into:

$$\boldsymbol{G_{b_2} q_{b_2}} - \boldsymbol{B_{22}}[\boldsymbol{h_{b_2}} + \boldsymbol{v}_3 h_1] = \boldsymbol{O},$$

which expands to:

$$\begin{pmatrix} G_2 & & & \\ & G_3 & & \\ & & G_7 & \\ & & & G_8 \end{pmatrix} \begin{pmatrix} q_2 \\ q_3 \\ q_7 \\ q_8 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} h_6 - h_1 \\ h_2 - h_1 \\ h_5 - h_1 \end{pmatrix} = \boldsymbol{O}.$$

The third pipe head-loss block equation or the third block equation ($\boldsymbol{B_3}$) in Eq. (5.24) is:

$$\boldsymbol{G_{b_3} q_{b_3}} - \boldsymbol{B_{33} h_{b_3}} = \boldsymbol{B_{32} h_{b_2}},$$

which expands to:

$$\begin{pmatrix} G_4 & & \\ & G_5 & \\ & & G_6 \end{pmatrix} \begin{pmatrix} q_4 \\ q_5 \\ q_6 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} h_3 \\ h_4 \end{pmatrix} = \begin{pmatrix} h_2 \\ 0 \\ h_2 \end{pmatrix}. \tag{5.27}$$

Eq. (5.27) can be further simplified to

$$\begin{pmatrix} G_4 & & \\ & G_5 & \\ & & G_6 \end{pmatrix} \begin{pmatrix} q_4 \\ q_5 \\ q_6 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} h_3 - h_2 \\ h_4 - h_2 \end{pmatrix} = \boldsymbol{O}$$

using a similar manipulation as for Block 2 above.

Finally, the system of equations in Eq. (5.24) may be rewritten as:

		Pipes								***Nodes***							
	Block	$\boldsymbol{B_1}$	$\boldsymbol{B_2}$				$\boldsymbol{B_3}$			$\boldsymbol{B_1}$	$\boldsymbol{B_2}$			$\boldsymbol{B_3}$			
Pipes	$\boldsymbol{B_1}$	G_1								**1**	0	**0**	0	0	0	q_1	e_{l_7}
			G_2							**0**	1	**0**	0	0	0	q_2	0
	$\boldsymbol{B_2}$			G_3						**0**	0	**1**	0	0	0	q_3	0
					G_7					**0**	0	**−1**	1	0	0	q_7	0
						G_8				**0**	1	**0**	−1	0	0	q_8	0
	$\boldsymbol{B_3}$						G_4			**0**	0	**0**	0	1	0	q_4	0
								G_5		**0**	0	**0**	0	1	−1	q_5	0
									G_6	**0**	0	**0**	0	0	1	q_6	0
Nodes	$\boldsymbol{B_1}$	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**							h_1	$d_1 + d_2 + d_3 + d_4 + d_5 + d_6$
	$\boldsymbol{B_2}$	0	1	0	0	1	0	0	0							$h_6 - h_1$	d_6
		0	**0**	**1**	**−1**	**0**	**0**	**0**	**0**							$h_2 - h_1$	$d_2 + d_3 + d_4$
		0	0	0	1	−1	0	0	0							$h_5 - h_1$	d_5
	$\boldsymbol{B_3}$	0	0	0	0	0	1	1	0							$h_3 - h_2$	d_3
		0	0	0	0	0	0	−1	1							$h_4 - h_2$	d_4

(5.28)

5.7.2 Solving the example network

Consider the network shown in Fig. 5.2 and its permuted system of equations, Eq. (5.28). Each block becomes an independent system and can be solved sequentially from the root block to the end block. The system of equations for the root block, $\boldsymbol{B_1}$ (Block 1 in Fig. 5.2), which also represents a bridge, is:

$$\begin{pmatrix} G_1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} q_1 \\ h_1 \end{pmatrix} = \begin{pmatrix} e_{l_7} \\ d_1 + d_2 + d_3 + d_4 + d_5 + d_6 \end{pmatrix}, \tag{5.29}$$

the solution of which can be used to solve its child block, block $\boldsymbol{B_2}$ (Block 2 in Fig. 5.2) by using:

$$\begin{pmatrix} G_2 & & & & 1 & 0 & 0 \\ & G_3 & & & 0 & 1 & 0 \\ & & G_7 & & 0 & -1 & 1 \\ & & & G_8 & 1 & 0 & -1 \\ 1 & 0 & 0 & 1 & & & \\ 0 & 1 & -1 & 0 & & & \\ 0 & 0 & 1 & -1 & & & \end{pmatrix} \begin{pmatrix} q_8 \\ q_7 \\ q_3 \\ q_2 \\ h_6 - h_1 \\ h_2 - h_1 \\ h_5 - h_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ d_6 \\ d_2 + d_3 + d_4 \\ d_5 \end{pmatrix}, \tag{5.30}$$

and finally, the end block, block $\boldsymbol{B_3}$ Block 3 in Fig. 5.2) can be solved by using:

$$\begin{pmatrix} G_4 & & & 1 & 0 \\ & G_5 & & 1 & -1 \\ & & G_6 & 0 & 1 \\ 1 & 1 & 0 & & \\ 0 & -1 & 1 & & \end{pmatrix} \begin{pmatrix} q_6 \\ q_5 \\ q_3 \\ h_3 - h_2 \\ h_4 - h_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ d_3 \\ d_4 \end{pmatrix}. \tag{5.31}$$

The systems of equations for each of the three blocks can also be solved in parallel.

Note that, when using BBPA, if the head loss of the example network shown in Fig. 5.2 is modeled by the Hazen-William formula and the nodal demands at nodes three and four are zero, this does not cause a failure of the method due to singularity of the Schur complement, unlike the GGA and RCTM on the same network (Elhay and Simpson 2011). In addition, the block with zero total demand can be solved (1) prior to the iterative phase by assigning zero flows to all applicable pipes and (2) by assigning the heads of the source to all nodes in this block after the iterative phase.

5.8 Relation of BBPA to other solution methods

The BBPA can be described as a pre-and-post-processing method for the following reasons: (1) it finds the blocks and bridges of a WDS, (2) the bridges can be solved by using a linear process similar to the forest component, and (3) then uses any WDS solution method, for example GGA, RCTM, or GMPA, to, independently, solve each block.

The BBPA can also be used to identify the forest component of the network. However, the use of the FCPA requires less overhead than the BBPA.

The same topological properties exploited by FCPA and BBPA are partly responsible for the savings achieved by partial-update (Abraham and Stoianov 2015). The forest and bridge components - being linear - converge after just one iteration of application of a non linear solver. The partial update scheme is able to exploit this by checking for convergence

every iteration. Once the convergence test for a pipe has been met, the head-loss of the converged component does not need to be re-computed, whereas the linear solver for the full system is required until the convergence tests for all pipes have been met. In contrast, FCPA and BBPA have the advantage of identifying these components in advance and removing them from non-linear solution process. BBPA also has the additional advantage of being able to exploit earlier convergence of different blocks in the core network and removing them from the problem once they have converged. As a result, the authors recommend that it is inefficient to implement the partial update for a full WDS system before applying the FCPA and the BBPA. The usefulness of applying the partial update to each block requires further investigation.

5.9 Case Studies

A comparison of the GGA with or without BBPA on eight case study networks has been carried out in order to support the above discussion. Note that the first step each method is to use FCPA to remove the forest component from the case study networks, to ensure a fair comparison.

The efficiency and reliability of the BBPA in a once-off simulation setting, in which the steady-state heads and flows are computed just once with the given WDS parameters, was benchmarked against an efficient GGA implementation. As a baseline, timings of the solution process for the benchmark networks using EPANET2 were also recorded. The benchmark tests were performed on a Intel(R) Core(TM) CPU i5-4590 running at 3.30 GHz with 4 cores in C++ under IEEE-standard double precision floating point arithmetic with machine epsilon $\epsilon_{mach} = 2.22 \times 10^{-16}$. The number of cores allocated to each test was limited to one. Each timing test, measuring wall-clock time, was repeated 15 times on each benchmark network.

It is shown that the use of an efficiently implemented BBPA can provide a significant runtime reduction and improvement in the reliability of the solution. The BBPA with the GGA and the standalone GGA were each applied to eight case studies with between 932 and 19,651 pipes and between 848 and 17,977 nodes with no pumps and no valves.

5.10 Results and Discussion

The basic details of the case study networks considered in this study are described in columns 2 to 4 in Table 5.1 and more information can be found in Simpson et al. (2012). The size of the core component for each of the eight case studies is shown in the columns 5 and 6, the number of blocks in column 7, with the number of blocks with no nodal demands in the brackets, and the number of bridges in column 8. Table 5.2 shows the detailed profile of the size of each block in each of the eight case study networks. The size of the largest block, smallest block, and median block and the number in brackets is the percentage of the corresponding block size as a proportion of the core component of the network

Table 5.3 shows the summary statistics of the 15 repetitions of each solution method applied to the eight benchmark networks. The GGA benefits from the use of the BBPA by between 33% and 70%. It has been established in Elhay et al. (2014) and Abraham and Stoianov (2015) that the number of non-zeros can be used as a surrogate to approximate the runtime of the non-linear system. The saving in runtime is partially achieved through the reduction in the number of non-zeros by the removal of the bridge components.

Table 5.1. Benchmark networks summary, their core network size, the number of blocks and the number of bridges

	Full Network			Core network		BBPA	
Network	n_p	n_j	n_s	n_{j_c}	n_{p_c}	The number of blocks	The number of bridges
N_1	934	848	8	573	487	33(1)*	118
N_2	1118	1039	2	797	718	10(2)	45
N_3	1975	1770	4	1152	947	7	6
N_4	2465	1890	3	2036	1461	47(3)	62
N_5	2509	2443	2	1087	1741	8(1)	45
N_6	8585	8392	2	6735	6542	7(2)	58
N_7	14830	12523	7	11898	9591	487(19)	895
N_8	19647	17971	15	15232	13557	17(2)	59

*Numbers in the brackets refers to the number of blocks with no nodal demands

Table 5.2. The profile of blocks in each of the eight case study networks: size of the largest, the smallest and the median blocks

	Largest size block		Smallest size block		Median size block	
Network	n_p	n_j	n_p	n_j	n_p	n_j
N_1	81(18)*	62(17)	3(0.7)	2(0.5)	7(1.6)	5(1.4)
N_2	684(91.9)	615(92.2)	2(0.3)	1 (0.1)	9.5 (1.3)	8(1.2)
N_3	953 (83.1)	78(33.1)	6(0.5)	5(2.1)	31 (2.7)	25.5(10.8)
N_4	1549(78.7)	1100(78.8)	2 (0.1)	1 (0.1)	7(0.4)	5(0.4)
N_5	1061(60.3)	1026(60.5)	2(0.1)	1(0.1)	53(3.0)	52(3.1)
N_6	5578(83.7)	5418(83.7)	2(0.03)	1(0.02)	51(0.8)	50(0.8)
N_7	8418(77.00)	6970(88.52)	2(0.02)	1(0.01)	4(0.04)	1(0.01)
N_8	14961(98.78)	13309(98.79)	3(0.02)	2(0.01)	12(0.08)	11(0.08)

*Numbers in the brackets refers to the percentage of the corresponding block size in the core component of the network

Another important factor of the algorithm efficiency is the number of iterations required to satisfy the stopping test. Figs. (5.3) shows that the number of iterations required by each block of networkN_1 and the number of iteration required by the full system to satisfy the stopping test. The horizontal axis shows the number of junctions and the vertical axis shows the number of iterations, and the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test. For example, in network N_1 there are six blocks that have two nodes, three of which require six iterations to satisfy the stopping test; one of which requires five iterations to satisfy the stopping test; and four of which require four iterations to satisfy the stopping test. The number of iterations that is required by each block of N_1 is bounded above by that which is required by the full network of N_1. The bubble plots for networks N_2 to N_8 can be found in the supplemental data.

On another note, the BBPA can also be used to improved reliability of the solution. Fig. (5.4) shows that the condition number at the solution and the condition number for

Table 5.3. The mean time of once-off simulation runs averaged over 15 once-off simulations for each of the two solution methods applied eight case study networks (milliseconds±standard error) and the "% diff." refers to relative difference compared to the GGA mean time

	EPANET	GGA with FCPA	GGA with BBPA	
	Mean time	Mean time	Mean time	%diff.
N_1	9.09	4.66 ± 0.07	2.32 ± 0.05	-50%
N_2	16.75	6.61 ± 0.07	2.86 ± 0.04	-56%
N_3	21.46	8.72 ± 0.09	3.64 ± 0.05	-58%
N_4	26.45	22.76 ± 0.53	6.64 ± 0.11	-70%
N_5	28.46	12.19 ± 0.13	5.97 ± 0.12	-51%
N_6	172.84	44.79 ± 0.18	28.53 ± 0.12	-36%
N_7	307.17	63.06 ± 0.65	42.35 ± 0.67	-33%
N_8	600.08	131.82 ± 3.99	59.08±0.6	-55%

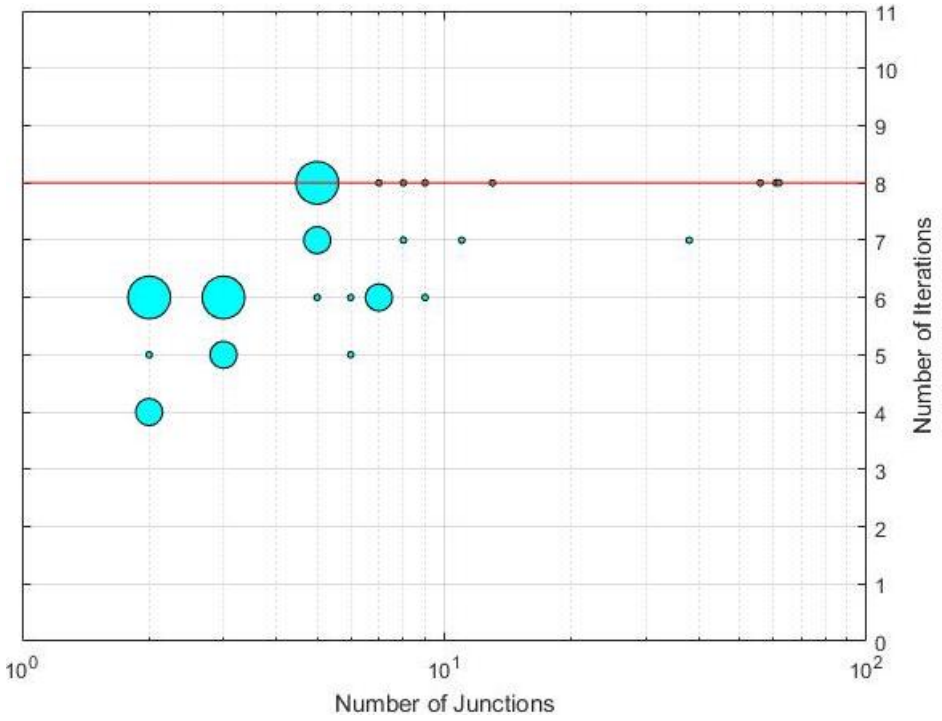


Fig. 5.3. The number of iterations for each block of network N_1 against the number of junctions (the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test)

the full system. The following observation can be made from the Fig. (5.4) that (i) the condition number for each block is bounded above by the condition number for the full matrix, (ii) each of networks N_2, N_5, N_6, and N_8 has one block with the same condition number as the full system.

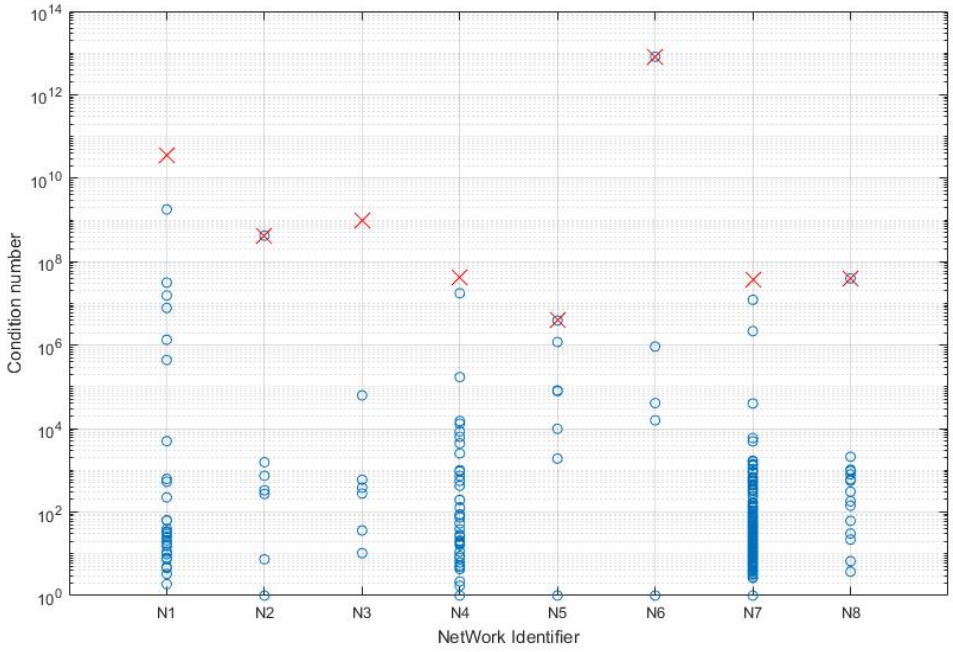


Fig. 5.4. The condition number of the Schur complement at the solution for each block (scatter point) and the condition number of the Schur complement for the full system (red line)

5.11 Conclusions

In this paper, the bridge-block partitioning algorithm is introduced. The BBPA is a pre-processing and post-processing algorithm that (1) first partitions the network into bridge components and block components, (2) then solves for the flows in the bridge components by a linear process, (3) after that it separately solves for the flows and the estimated heads for each independent block by using any WDS solver, and (4) finally the heads are recovered by a linear process at the end. This partitioning of the network can be used to speed-up the solution process of the steady state demand-driven hydraulic simulation and to improve the reliability of the results if the core component of the WDS graph is one-connected. The speed-up of the solution process is achieved by (1) solving the bridge component in the BBPA by a linear process similar to that of solving for the forest in the FCPA, which reduces the number of non-zeroes in the Schur complement (2) solving each block by using the minimum number of iterations that is required by that block. Moreover, the BBPA improves the reliability of the results because the condition number of the Schur Complement for each block is bounded above by the condition number for the Schur Complement of the full system.

The usefulness of the BBPA has also been demonstrated by applying it to eight benchmark networks with between 934 and 19,647 pipes and between 848 and 17,971 nodes. The total savings in wall clock time after applying the BBPA to the GGA are between 33% and 70%. It is shown that, the number of iterations and the condition number required by each block are bounded by the number of iterations and the condition number required by the full system, respectively. The use of the BBPA can also minimize the need to regularize the zero flows when the head loss is modelled by the Hazen-William head loss equation. This is because in real life systems, such as the case study networks used

in this paper, can have blocks, such that the nodes in these block all have zero demands, which can be handled by use of BBPA. Moreover, when regularization is needed, it is only required to be applied at the corresponding block instead of the full system of equations.

5.12 References

References are included in bibliography Chapter 7. In addition the submitted paper is given in Appendix C.

5.13 Appendix: Why the number of iterations required by each block is bounded above by that of the full system

The BBPA is derived to partition the WDS network into a number of blocks to improve the efficiency and reliability of the WDS solution process. The number of iterations that is required by each block is bounded above by the number of iterations that is required by the full system. The permuted system of equations shown in Eq. (5.15) can be rewritten as

$$\begin{pmatrix} \boldsymbol{F}_B^{(m)} & -\boldsymbol{A_B} \\ -\boldsymbol{A_B}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_B^{(m+1)} \\ \boldsymbol{h}_B^{(m+1)} \end{pmatrix} - \begin{pmatrix} 0 & \boldsymbol{A_C} \\ \boldsymbol{A_C}^T & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_B^{(m+1)} \\ \boldsymbol{h}_B^{(m+1)} \end{pmatrix} = - \begin{pmatrix} (\boldsymbol{G}_B^{(m)} - \boldsymbol{F}_B^{(m)})\boldsymbol{q}_B^{(m)} - \boldsymbol{a}_B \\ -\boldsymbol{d}_B. \end{pmatrix}. \tag{5.32}$$

Note that each block row of the matrix $\boldsymbol{A_C}$, that represents a root block, is entirely zero. As a result, the system of equations for the root block B_i is

$$\begin{pmatrix} \boldsymbol{F}_{b_i}^{(m)} & -\boldsymbol{B_{ii}} \\ -\boldsymbol{B_{ii}}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_{b_i}^{(m+1)} \\ \boldsymbol{h}_{b_i}^{(m+1)} \end{pmatrix} = - \begin{pmatrix} (\boldsymbol{G}_{\boldsymbol{b_i}}^{(\boldsymbol{m})} - \boldsymbol{F}_{\boldsymbol{b_i}}^{(\boldsymbol{m})})\boldsymbol{q}_{\boldsymbol{b_i}}^{(\boldsymbol{m})} - \boldsymbol{a}_{\boldsymbol{b_i}} \\ -\widehat{\boldsymbol{d}}_{b_i}. \end{pmatrix}. \tag{5.33}$$

Also note that each block row of the matrix $\boldsymbol{A_C}$, that does not represent a root block, has exactly one non-zero block and each of these blocks has exactly one non-zero column. As a result, the system of equations for a block, B_j, that is not a root block is

$$\begin{pmatrix} \boldsymbol{F}_{b_j}^{(m)} & -\boldsymbol{B_{kj}} & -\boldsymbol{B_{jj}} \\ -\boldsymbol{B_{jj}}^T & \boldsymbol{O} & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_{b_j}^{(m+1)} \\ \boldsymbol{h}_{b_k}^{(m+1)} \\ \boldsymbol{h}_{b_j}^{(m+1)} \end{pmatrix} = - \begin{pmatrix} (\boldsymbol{G}_{\boldsymbol{b_j}}^{(\boldsymbol{m})} - \boldsymbol{F}_{\boldsymbol{b_j}}^{(\boldsymbol{m})})\boldsymbol{q}_{\boldsymbol{b_j}}^{(\boldsymbol{m})} - \boldsymbol{a}_{\boldsymbol{b_j}} \\ -\widehat{\boldsymbol{d}}_{b_j} \end{pmatrix}, \tag{5.34}$$

where the non-zero block row entry at block row j, $\boldsymbol{B_{kj}} = \boldsymbol{P}_{e_{b_j}} \boldsymbol{A_1} \boldsymbol{R}_{v_{b_k}}^T \in \mathbb{R}^{n_{p_{b_j}} \times n_{j_{b_j}}}$, which represents the connection between the current block j and its parent block k, has only one non-zero column entry, $\boldsymbol{A}_{2_{b_j}} = \boldsymbol{P}_{e_{b_j}} \boldsymbol{A_1} \boldsymbol{R}_{cv_{b_k}}^T \in \mathbb{R}^{n_{p_{b_j}} \times 1}$. This non-zero column entry is the unknown-head node-arc incidence matrix for block b_k and cv_{b_k} is the cut-vertex that behaves as the pseudo-source in block b_k. We can write the term $\boldsymbol{B_{kj}} \boldsymbol{h}_{b_k}^{(m+1)}$ as $[\boldsymbol{P}_{e_{b_j}} \boldsymbol{A_1} \boldsymbol{R}_{cv_{b_k}}^T][\boldsymbol{R}_{cv_{b_k}} \boldsymbol{h}]$, which is $\boldsymbol{A}_{2_{b_j}} \boldsymbol{h}_{cv_{b_j}}$ (see (5.25) and (5.27)).

In addition, the combination of matrices $\boldsymbol{B_{jj}}$ and $\boldsymbol{A}_{2_{b_j}}$ is the Laplacian matrix of the graph of block B_j. Every row of a Laplacian matrix has exactly two non-zero entries:

1 and -1. Therefore, $\boldsymbol{B_{jj}}\boldsymbol{v_{n_{j_{b_j}}}} + \boldsymbol{A_{2_{b_j}}}\boldsymbol{v_{n_{f_{b_j}}}} = \boldsymbol{o}$. We also know that $n_{f_{b_j}} = 1$ which is equivalent to $\boldsymbol{B_{jj}}\boldsymbol{v_{n_{j_{b_j}}}} = -\boldsymbol{A_{2_{b_j}}}$ as shown in Lemma 2.

Thus, the left-hand-side of the first block equation of Eq. (5.34) is:

$$\boldsymbol{F}_{b_j}^{(m)}\boldsymbol{q}_{b_j}^{(m+1)} - \boldsymbol{B_{kj}}\boldsymbol{h}_{b_k}^{(m+1)} - \boldsymbol{B_{jj}}\boldsymbol{h}_{b_j}^{(m+1)}$$

and can be rewritten as

$$\boldsymbol{F}_{b_j}^{(m)}\boldsymbol{q}_{b_j}^{(m+1)} - \boldsymbol{B_{jj}}\boldsymbol{v_{n_{j_{b_j}}}}\boldsymbol{h}_{b_k}^{(m+1)} - \boldsymbol{B_{jj}}\boldsymbol{h}_{b_j}^{(m+1)}$$

and finally, denoting $\widehat{\boldsymbol{q}}^{(m+1)} = \boldsymbol{q}_{b_j}^{(m+1)}$ and $\widehat{\boldsymbol{h}}^{(m+1)} = \boldsymbol{h}_{b_j}^{(m+1)} + \boldsymbol{v_{n_{j_{b_j}}}}\boldsymbol{h}_{b_k}^{(m+1)}$, gives

$$\begin{pmatrix} \boldsymbol{F}_{b_j}^{(m)} & -\boldsymbol{B_{jj}} \\ -\boldsymbol{B_{jj}}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \widehat{\boldsymbol{q}}^{(m+1)} \\ \widehat{\boldsymbol{h}}^{(m+1)} \end{pmatrix} = - \begin{pmatrix} (\boldsymbol{G_{b_j}^{(m)}} - \boldsymbol{F_{b_j}^{(m)}})\boldsymbol{q_{b_j}^{(m)}} - \boldsymbol{a_{b_j}} \\ -\widehat{\boldsymbol{d}}_{b_j} \end{pmatrix}. \tag{5.35}$$

The matrices on the left-hand-side of Eq. (5.33) and Eq. (5.35) are identical and invertible and the right-hand-side of both equations are also identical. Therefore

$$\begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{h} \end{pmatrix} = \begin{pmatrix} \widehat{\boldsymbol{q}} \\ \widehat{\boldsymbol{h}} \end{pmatrix}$$

The Newton equation shown in Eq. (5.32), which is the GGA solution of an orthogonal permutation of the original system of equations, has the same flows and heads iterates as the GGA solution of Eq. (5.5). Moreover, it is shown above that the Newton equation in Eq. (5.33) has the same flow and head iterates as the Newton equation in Eq. (5.32). At the same time, the Newton equation in Eq. (5.35) has the same flow iterates as the Newton equation in Eq. (5.32) and the actual heads can be recovered from the interim heads a posteriori. Thus, solving each block individually produces the same flow iterates as solving the unpartitioned WDS network. The number of iterations for each block to satisfy the stopping test, $\frac{||\boldsymbol{q}^{m+1} - \boldsymbol{q}^m||_\infty}{||\boldsymbol{q}^{m+1}||_\infty}$, is bounded above by the number of iterations required by the whole system.

5.14 Supplementary Data

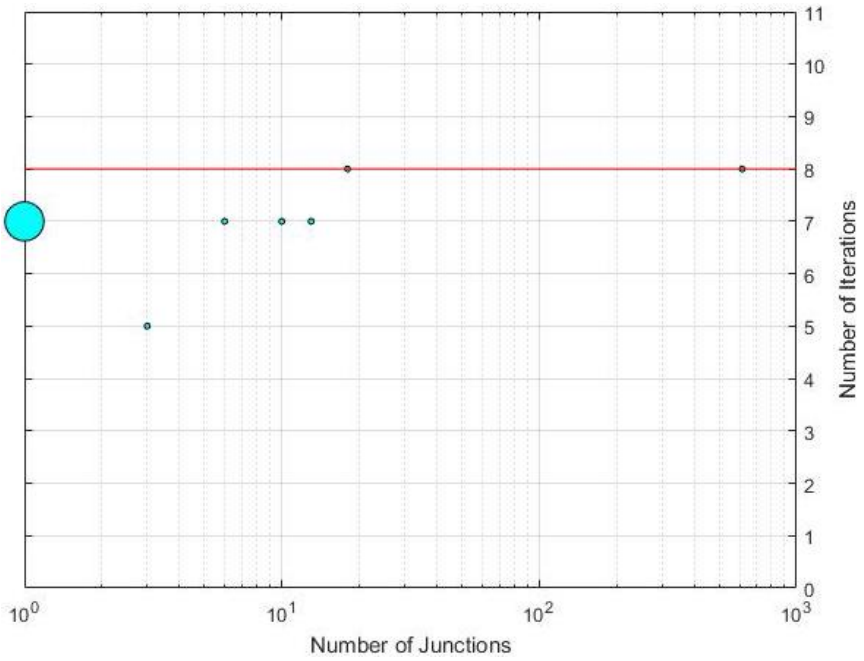


Fig. 5.5. The number of iterations for each block of network N_2 against the number of junctions (the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test)

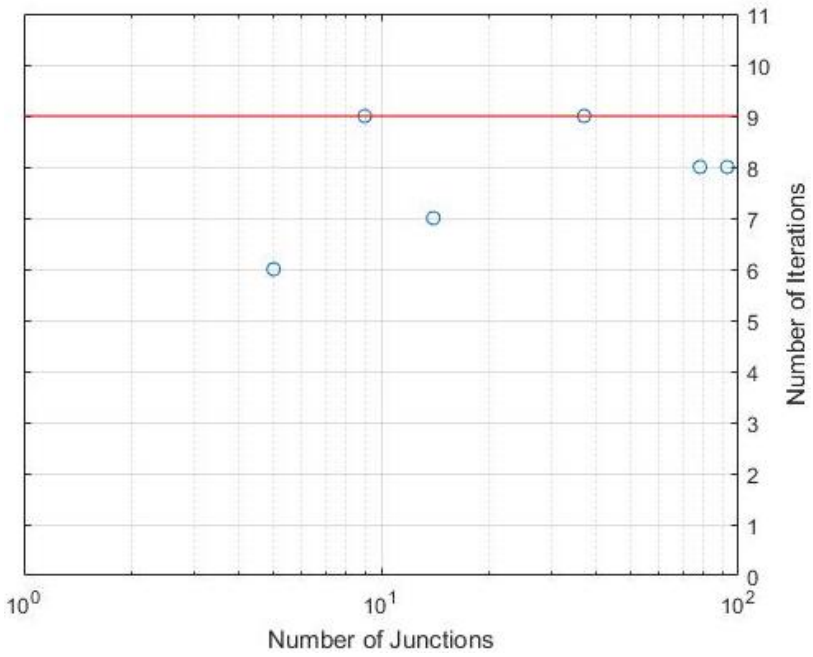


Fig. 5.6. The number of iterations for each block of network N_3 against the number of junctions (the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test)

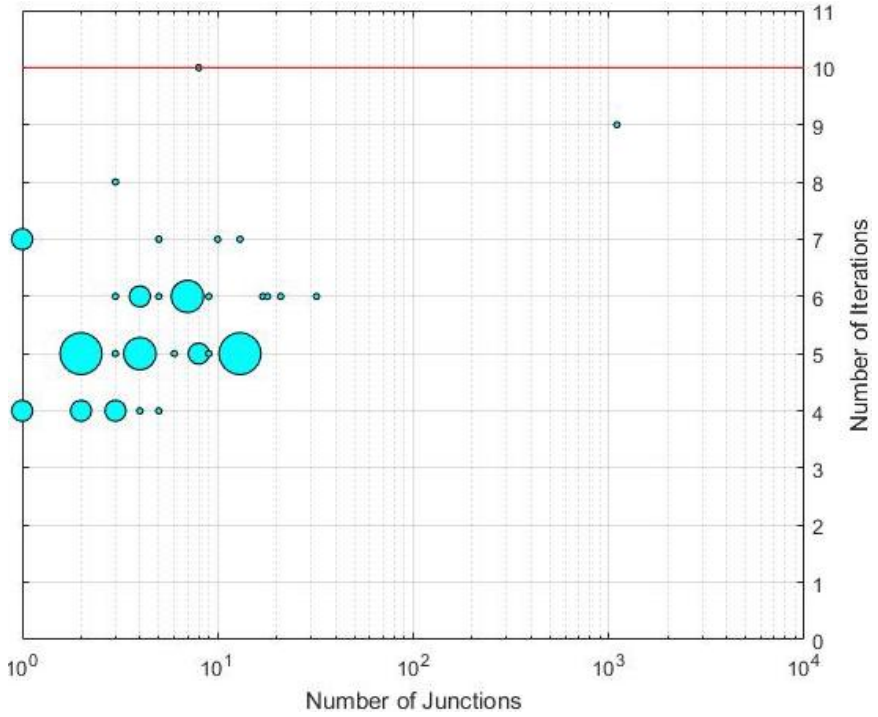


Fig. 5.7. The number of iterations for each block of network N_4 against the number of junctions (the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test)

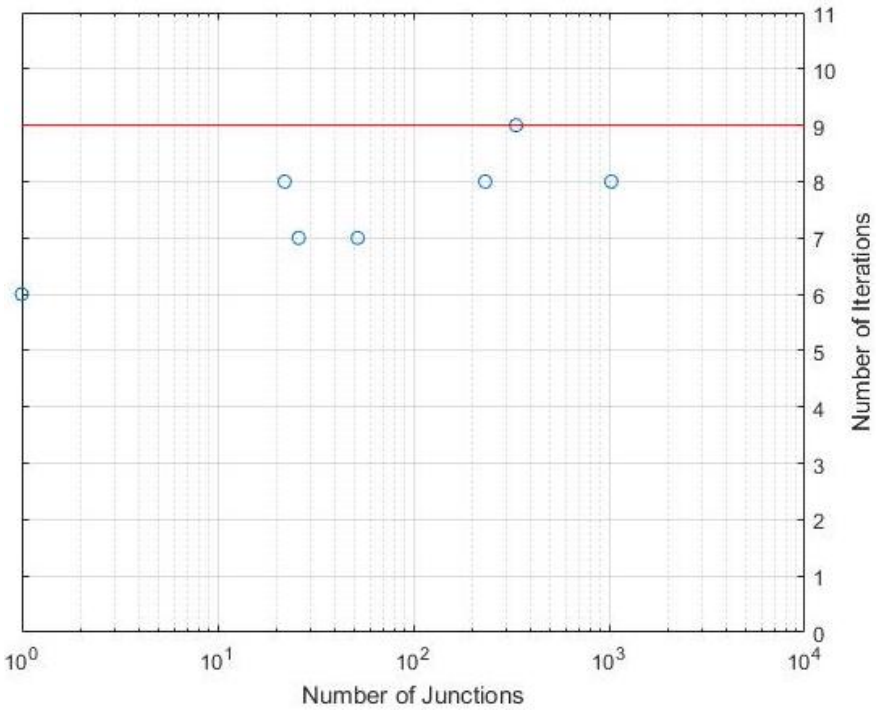


Fig. 5.8. The number of iterations for each block of network N_5 against the number of junctions (the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test)

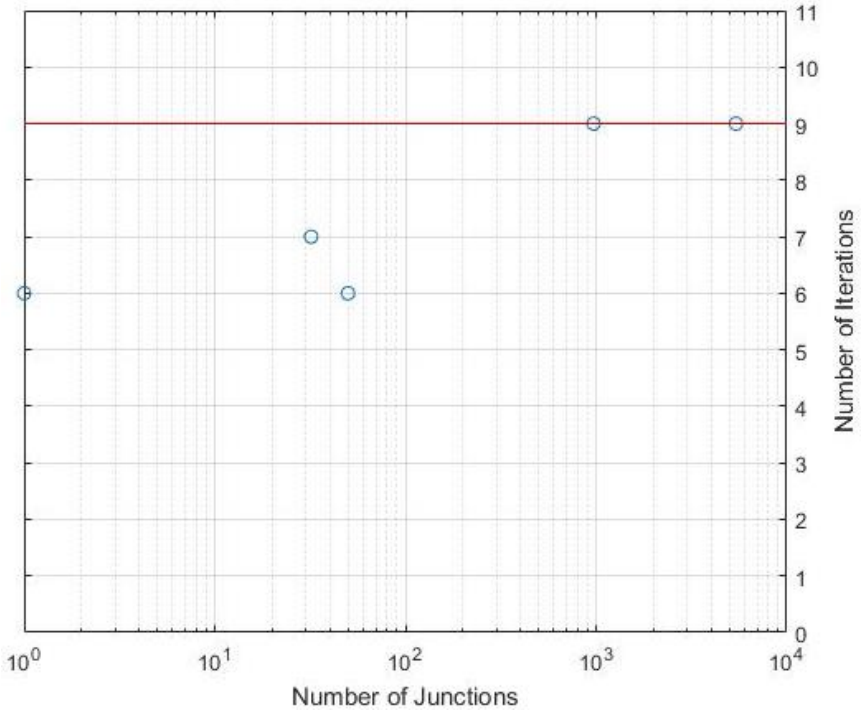


Fig. 5.9. The number of iterations for each block of network N_6 against the number of junctions (the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test)

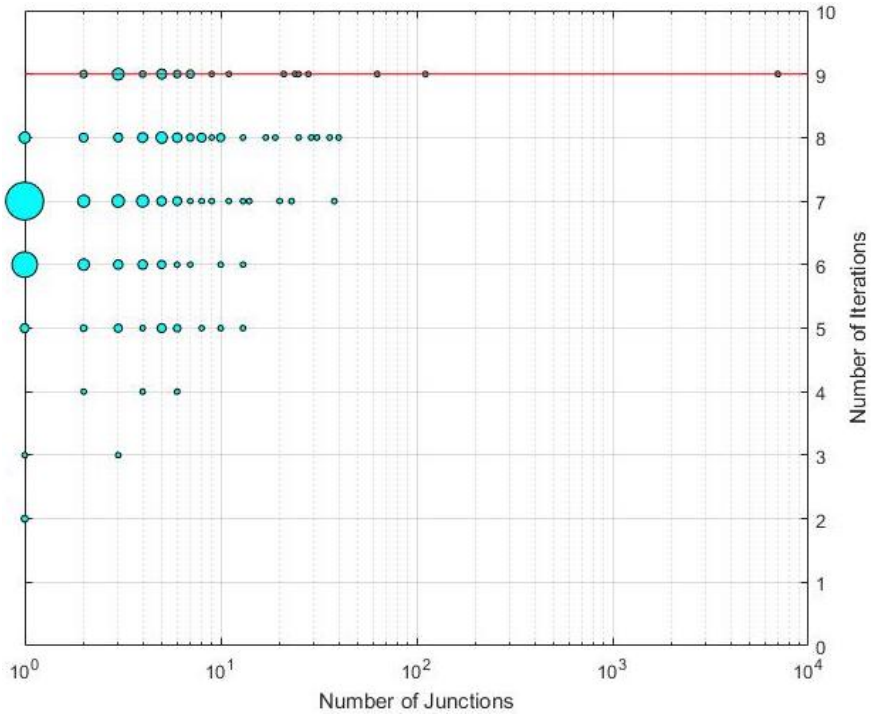


Fig. 5.10. The number of iterations for each block of network N_7 against the number of junctions (the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test)

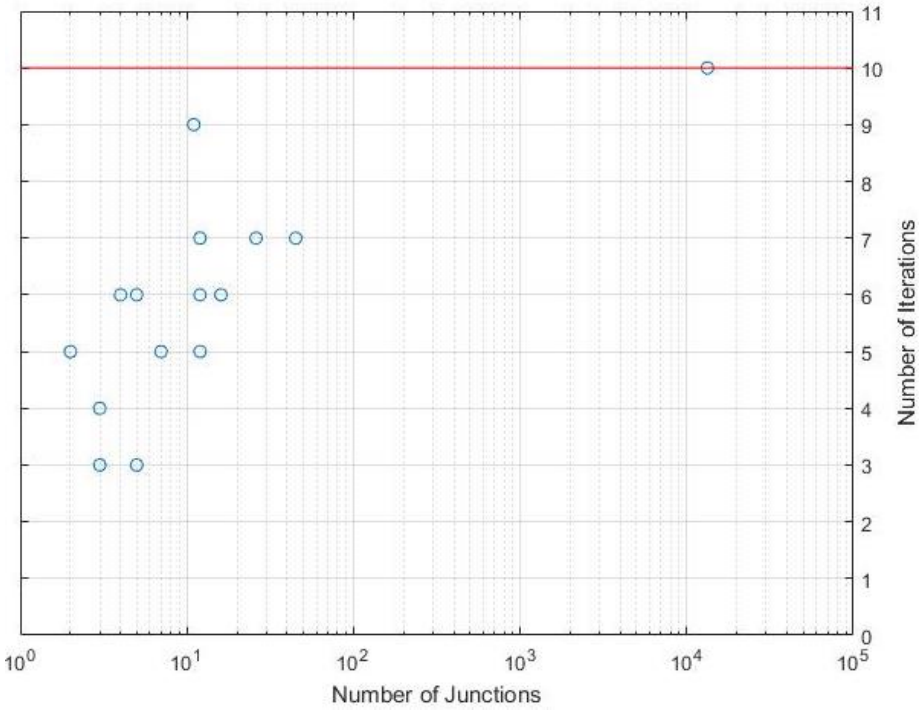


Fig. 5.11. The number of iterations for each block of network N_8 against the number of junctions (the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test)

This page is intentionally left blank

Chapter 6

Conclusions and Recommendations for Future Study

6.1 Conclusions

The research presented here has been mainly concerned with the existing steady-state demand-driven water distribution system (WDS) solution methods and with the development of a new graph theory based partitioning method. WDS solution methods are a key component that is used repeatedly to solve WDS models in every WDS design, management and operation problem. There are a number of special graph properties of the WDS graph that can be exploited to improve the efficiency of the WDS solution process. This thesis demonstrates the usefulness of using graph properties in terms of the computational speed and numerical reliability.

6.2 Research Contributions

The key contributions of this research are summarised as follows.

The development of a testbed for water distribution system solution methods. A software package, that can be used (1) to efficiently implement a number of WDS solution methods; (2) to incorporate newly developed WDS solution methods; (3) to compare different solution methods; (4) to focus the research on the most time-consuming parts of a solution method; and (5) to guide the choice of solution method when multiple simulation runs are required, is an essential tool to achieve the aims that have been set forth for this research. Chapter 3 introduced WDSLib, a library for steady-state hydraulic simulation of WDS networks. WDSLib provides (1) a fast simulation platform for both once-off and multi-run simulations and (2) a testbed for comparing different solution methods in different settings for the same network topologies. As such, WDSLib is designed with a pluggable architecture which can extended to efficiently incorporate new solution methods as they are created. This will enhance the capability of the research community to demonstrate the efficacy of new methods without having to re-engineer the content of shared WDSLib functions and data representations.

Propose a framework for benchmarking water distribution system solution methods In Chapter 4, a framework for comparing different WDS solution methods is proposed.

The proposed framework significantly reduces the computation load of each of the solution methods that are implemented in WDSLib. This is achieved by categorising each of the functions that are used in each of the solution methods into three categories: (1) the functions that will only have to be executed once are called level one (L_1) functions. L_1 functions relate to network topology, which is invariant for the whole simulation; (2) in a multi-simulation setting, certain functions will need to be run once for every hydraulic-phase. These, once-per-assessment functions, are called level two (L_2) functions; and (3) for every hydraulic assessment, there is a non-linear iterative phase in the solution process. The functions in this phase run many times for each hydraulic assessment until the stopping test has been satisfied. These iterative-phase functions are called level three (L_3) functions. Equipped with such a framework, it is possible (1) to conduct a fair comparison between different solution methods; and (2) to allow each function to be run the minimum number of times determined by its simulation setting.

Use the proposed framework to conduct a benchmark study on four different WDS solution methods The proposed framework is then used in Chapter 4 to benchmark the performance of four solution methods, the global gradient algorithm (GGA), the GGA with the forest-core partitioning algorithm (FCPA), the reformulated co-tree flows method, and the RCTM with the FCPA, against each other. Each of the four solution methods is applied to eight case study networks.

Propose a new partitioning algorithm to improve the existing WDS solution methods In Chapter 5, a new graph partitioning algorithm, bridge-block partitioning algorithm (BBPA), is proposed. The BBPA is a pre-and-post-processing algorithm that partitions the WDS graph into a number of bridge components and a number of block components. Each of the bridge components can be solved using a linear process similar to the FCPA and each of the block components can be separately solved by using any WDS solution method, the GGA, RCTM, or GMPA. There is a number of advantages to using the BBPA: (1) the number of iterations required by each block is bounded above by that required by the unpartitioned system – solving the flows and heads in each block separately significantly reduces the overall computational time for the non-linear solver in almost all cases; (2) the condition number of the Schur complement of each block is bounded above by that of the unpartitioned system. In most cases, the condition numbers for all the individual blocks will be smaller than the condition number of the full matrix; (3) the solution of each block can be found in parallel in a demand-driven model because the flows and heads in one block component are independent from those of the other block components.

6.3 Recommendations for WDS demand-driven solution methods

The performance of any water distribution system solution method is very problem dependent. To date, there has been no reliable method that accurately predicts the performance of a given algorithm on a particular network a priori. This is reflected in the performance differences reported in this thesis.

The network topology is the most influential factor in the performance of different solution methods (matrix density, the distribution of non-zero elements after bandwidth reduction, etc.). Recommendations are given as follows:

1. There often are a significant number of forest pipes in most real-life water distribution systems. It is recommended to use the FCPA as the first step of every demand-driven steady-state WDS simulation.
2. Use the BBPA, after the FCPA, to identify the linear bridge components and the independent block components. The flows in the bridge components can be solved just once every iterative phase. The solution of each block can be found independently. Finally, the heads in the system can be solved just once after the iterative phase.
3. The performance of the RCTM is dependent upon the choice of spanning tree. It is difficult, if not impossible, to determine the optimal choice of spanning that minimises the number of non-zeros in the Schur complement. As a result, an arbitrary choice of spanning tree is normally used. A trial run is recommended before a multi-run simulation, to identify the relative performance and thereby determine which WDS solution method or combination of WDS solution methods to use.

6.4 Scope for future work

Suggestions for possible future work include:

1. The water distribution system solution methods that are evaluated and developed in this research have been applied to a number of real-life WDS networks with between 934 and 19,647 pipes and between 848 and 17,971 nodes. However, these real-life WDS networks only consist of pipes, nodes, tanks, and reservoirs. Extension of the solution methods to incorporate the use of control devices and pumps will be a valuable contribution.
2. The performance of different WDS solution methods are dependent upon the network topology. It would be an interesting study to determine a priori the best method to use for a given WDS network.
3. This work has only considered the steady-state solution of a WDS under the demand-driven model. The usefulness of the WDS solution methods, that are investigated in the current research, under the pressure driven setting should also be investigated in the future
4. The performance of the RCTM is determined by on the choice of spanning tree. Different choices of spanning tree will produce a different number of non-zeros in the Schur complement and a different set of initial guesses of flows. A study on how to choose an optimal or near optimal spanning tree may speed up the RCTM significantly.

This page is intentionally left blank

Chapter 7

Bibliography

Abraham, E & Stoianov, I 2015, 'Sparse null space algorithms for hydraulic analysis of large-scale water supply networks', *Journal of Hydraulic Engineering*, p. 04015058.

Alperovits, E & Shamir, U 1977, 'Design of optimal water distribution systems', *Water Resources Research*, vol. 13, no. 6, pp. 885–900.

Alvarruiz, F, Martínez-Alzamora, F & Vidal, AM 2015, 'Improving the efficiency of the loop method for the simulation of water distribution systems', *Journal of Water Resources Planning and Management*, vol. 141, no. 10, p. 04015019.

Anderson, EJ & Al-Jamal, KH 1995, 'Hydraulic-network simplification', *Journal of Water Resources Planning and Management*, vol. 121, no. 3, pp. 235–240.

Baños, R, Gil, C, Agulleiro, JI & Reca, J 2007, 'A memetic algorithm for water distribution network design', *Soft Computing in Industrial Applications*, Springer, pp. 279–289.

Benzi, M, Golub, GH & Liesen, J 2005, 'Numerical solution of saddle point problems', *Acta. Num.* Pp. 1–137.

Beyer, H & Schwefel, H 2002, 'Evolution strategies–A comprehensive introduction', *Natural Computing*, vol. 1, no. 1, pp. 3–52.

Cheung, PB, Van Zyl, JE & Reis, LFR 2005, 'Extension of EPANET for pressure driven demand modeling in water distribution system', *Computing and Control for the Water Industry*, vol. 1, pp. 311–316.

Creaco, E & Franchini, M 2013, 'Comparison of Newton-Raphson global and loop algorithms for water distribution network resolution', *Journal of Hydraulic Engineering*, vol. 140, no. 3, pp. 313–321.

Cross, H 1936, 'Analysis of flow in networks of conduits or conductors', *University of Illinois. Engineering Experiment Station. Bulletin; no. 286*,

Cunha, MC & Sousa, J 1999, 'Water distribution network design optimization: simulated annealing approach', *Journal of Water Resources Planning and Management*, vol. 125, no. 4, pp. 215–221.

Dandy, GC, Simpson, AR & Murphy, LJ 1996, 'An improved genetic algorithm for pipe network optimization', *Water Resources Research*, vol. 32, no. 2, pp. 449–458.

Davis, T, Hager, WW & Duff, IS 2013, 'Suitesparse', *URL< http://www. cise. ufl. edu/research/sparse/SuiteSparse*,

Deuerlein, J 2008, 'Decomposition model of a general water supply network graph', *Journal of Hydraulic Engineering*, vol. 134, no. 6, pp. 822–832.

Deuerlein, J, Elhay, S & Simpson, AR 2015, 'Fast Graph Matrix Partitioning Algorithm for Solving the Water Distribution System Equations', *Journal of Water Resources Planning and Management*, vol. 142, no. 1, pp. 04015037–1–04015037–11.

Diestel, R 2005, 'Graph Theory', *Graduate Texts in Mathematics*, vol. 173,

Dijkstra, EW 1959, 'A note on two problems in connexion with graphs', *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271.

Duff, IS, Grimes, RG & Lewis, JG 1989, 'Sparse matrix test problems', *ACM Transactions on Mathematical Software (TOMS)*, vol. 15, no. 1, pp. 1–14.

Dunlop, E 1991, 'WADI users manual', *Local Government Computer Services Board, Dublin, Ireland*,

Elhay, S & Simpson, AR 2011, 'Dealing with zero flows in solving the nonlinear equations for water distribution systems', *Journal of Hydraulic Engineering*, vol. 137, no. 10, pp. 1216–1224.

Elhay, S, Simpson, AR, Deuerlein, J, Alexander, B & Schilders, WH 2014, 'Reformulated co-tree flows method competitive with the global gradient algorithm for solving water distribution system equations', *Journal of Water Resources Planning and Management*, vol. 140, no. 12, pp. 04014040–1–04014040–10.

Elhay, S, Deuerlein, J, Piller, O & Simpson, AR 2018, 'Graph Partitioning in the Analysis of Pressure Dependent Water Distribution Systems', *Journal of Water Resources Planning and Management*, vol. 144, no. 4, p. 04018011, DOI: 10.1061/(ASCE)WR.1943-5452.0000896.

Epp, R & Fowler, AG 1970, 'Efficient code for steady-state flows in networks', *Journal of the Hydraulics Division*, vol. 96, no. 1, pp. 43–56.

Eusuff, MM & Lansey, KE 2003, 'Optimization of water distribution network design using the shuffled frog leaping algorithm', *Journal of Water Resources Planning and Management*, vol. 129, no. 3, pp. 210–225.

Geem, ZW 2006, 'Optimal cost design of water distribution networks using harmony search', *Engineering Optimization*, vol. 38, no. 03, pp. 259–277.

Geem, ZW, Kim, JH & Loganathan, GV 2002, 'Harmony search optimization: application to pipe network design', *International Journal of Modelling and Simulation*, vol. 22, no. 2, pp. 125–133.

Gorev, NB, Kodzhespirov, IF, Kovalenko, Y, Prokhorov, E & Trapaga, G 2012, 'Method to Cope with Zero Flows in Newton Solvers for Water Distribution Systems', *Journal of Hydraulic Engineering*, vol. 139, no. 4, pp. 456–459.

Guidolin, M, Burovskiy, P, Kapelan, Z & Savic, DA 2010, 'CWSNet: An object-oriented toolkit for water distribution system simulations', *Proceedings of the 12th Annual Water Distribution Systems Analysis Conference, WDSA*, pp. 12–15.

Hopcroft, J & Tarjan, R 1973, 'Algorithm 447: Efficient Algorithms for Graph Manipulation', *Commun. ACM*, vol. 16, no. 6, pp. 372–378, ISSN: 0001-0782, DOI: 10.1145/362248.362272, URL: http://doi.acm.org/10.1145/362248.362272.

Jun, L & Guoping, Y 2012, 'Iterative methodology of pressure-dependent demand based on EPANET for pressure-deficient water distribution analysis', *Journal of Water Resources Planning and Management*, vol. 139, no. 1, pp. 34–44.

Lippai, I, Heaney, JP & Laguna, M 1999, 'Robust water system design with commercial intelligent search optimizers', *Journal of Computing in Civil Engineering*, vol. 13, no. 3, pp. 135–143.

Loganathan, GV, Greene, JJ & Ahn, TJ 1995, 'Design heuristic for globally minimum cost water-distribution systems', *Journal of Water Resources Planning and Management*, vol. 121, no. 2, pp. 182–192.

Maier, HR, Simpson, AR, Zecchin, AC, Foong, WK, Phang, KY, Seah, HY & Tan, CL 2003, 'Ant colony optimization for design of water distribution systems', *Journal of Water Resources Planning and Management*, vol. 129, no. 3, pp. 200–209.

Montalvo, I, Izquierdo, J, Pérez, R & Tung, MM 2008, 'Particle swarm optimization applied to the design of water supply systems', *Computers & Mathematics with Applications*, vol. 56, no. 3, pp. 769–776.

Morley, MS & Tricarico, C 2008, *Pressure-Driven Demand Extension for EPANET (EPANETpdd)*, tech. rep., University of Exeter.

Morley, MS, Atkinson, RM, Savic, DA & Walters, GA 2000, 'OpenNet: An applicationindependent framework for hydraulic network representation, manipulation & dissemination', *Hydroinformatics 2000 Conference, University of Iowa, Iowa City, USA*, pp. 23–27.

Murphy, LJ, Simpson, AR & Dandy, GC 1993, *Pipe network optimization using an improved genetic algorithm*, Department of Civil and Environmental Engineering, University of Adelaide Adelaide, Australia.

Nethercote, N & Seward, J 2007, 'Valgrind: a framework for heavyweight dynamic binary instrumentation', *ACM Sigplan Notices*, vol. 42, no. 6, ACM, pp. 89–100.

Nielsen, HB 1989, 'Methods for analyzing pipe networks', *Journal of Hydraulic Engineering*, vol. 115, no. 2, pp. 139–157.

van Zyl, JE, Borthwick, J & Hardy, A 2003, 'Ooten: An object-oriented programmers toolkit for epanet', *Advances in Water Supply Management (CCWI 2003)*,

Ostfeld, A & Tubaltzev, A 2008, 'Ant colony optimization for least-cost design and operation of pumping water distribution systems', *Journal of Water Resources Planning and Management*, vol. 134, no. 2, pp. 107–118.

Perelman, L & Ostfeld, A 2011, 'Topological clustering for water distribution systems analysis', *Environmental Modelling & Software*, vol. 26, no. 7, pp. 969–972.

Qiu, M, Simpson, AR, Elhay, S & Alexander, B 2018, 'A Benchmarking Study of Water Distribution System Solution Methods', Manuscript submitted for publication to Journal of Water Resources Planning and Management.

Quindry, GE, Liebman, JC & Brill, ED 1981, 'Optimization of looped water distribution systems', *Journal of the Environmental Engineering Division*, vol. 107, no. 4, pp. 665–679.

Rahal, H 1995, 'A co-tree flows formulation for steady state in water distribution networks', *Advances in Engineering Software*, vol. 22, no. 3, pp. 169–178.

Rossman, LA 1994, 'Integrated Computer Applications in Water Supply', ed. by Bryan Coulbeck, vol. 2, Chichester, UK: John Wiley and Sons Ltd., chap. The EPANET Water Quality Model, pp. 79–93, ISBN: 0-471-94233-2, URL: `http://dl.acm.org/citation.cfm?id=190101.190107`.

— 2000, 'Epanet 2 users manual, us environmental protection agency', *Water Supply and Water Resources Division, National Risk Management Research Laboratory, Cincinnati, OH*, vol. 45268,

Saldarriaga, JG, Ochoa, S, Rodriguez, D & Arbeláez, J 2008, 'Water distribution network skeletonization using the resilience concept', *Water Distribution Systems Analysis 2008*, pp. 1–13.

Savic, DA & Walters, GA 1997, 'Genetic algorithms for least-cost design of water distribution networks', *Journal of Water Resources Planning and Management*, vol. 123, no. 2, pp. 67–77.

Schilders, WH 2009, 'Solution of indefinite linear systems using an LQ decomposition for the linear constraints', *Linear Algebra and its Applications*, vol. 431, no. 3, pp. 381–395.

Shamir, U & Salomons, E 2008, 'Optimal real-time operation of urban water distribution systems using reduced models', *Journal of Water Resources Planning and Management*, vol. 134, no. 2, pp. 181–185.

Siew, C & Tanyimboh, TT 2012, 'Pressure-dependent EPANET extension', *Water Resources Management*, vol. 26, no. 6, pp. 1477–1498.

Simpson, AR & Elhay, S 2010, 'Jacobian matrix for solving water distribution system equations with the Darcy-Weisbach head-loss model', *Journal of Hydraulic Engineering*, vol. 137, no. 6, pp. 696–700.

Simpson, AR, Dandy, GC & Murphy, LJ 1994, 'Genetic algorithms compared to other techniques for pipe optimization', *Journal of Water Resources Planning and Management*, vol. 120, no. 4, pp. 423–443.

Simpson, AR, Elhay, S & Alexander, B 2012, 'Forest-core partitioning algorithm for speeding up analysis of water distribution systems', *Journal of Water Resources Planning and Management*, vol. 140, no. 4, pp. 435–443.

Suribabu, CR 2010, 'Differential evolution algorithm for optimal design of water distribution networks', *Journal of Hydroinformatics*, vol. 12, no. 1, pp. 66–82.

Suribabu, CR & Neelakantan, TR 2006, 'Design of water distribution networks using particle swarm optimization', *Urban Water Journal*, vol. 3, no. 2, pp. 111–120.

Todini, E & Pilati, S 1988, 'A gradient algorithm for the analysis of pipe networks', *Computer Applications in Water Supply: vol. 1—systems analysis and simulation*, Research Studies Press Ltd., pp. 1–20.

Vasan, A & Simonovic, SP 2010, 'Optimization of water distribution network design using differential evolution', *Journal of Water Resources Planning and Management*, vol. 136, no. 2, pp. 279–287.

Vassiljev, A & Koppel, T 2015, 'Estimation of real-time demands on the basis of pressure measurements by different optimization methods', *Advances in Engineering Software*, vol. 80, pp. 67–71.

Zecchin, AC, Simpson, AR, Maier, HR, Leonard, M, Roberts, AJ & Berrisford, MJ 2006, 'Application of two ant colony optimisation algorithms to water distribution system optimisation', *Mathematical and Computer Modelling*, vol. 44, no. 5-6, pp. 451–468.

Zecchin, AC, Maier, HR, Simpson, AR, Leonard, M & Nixon, JB 2007, 'Ant colony optimization applied to water distribution system design: comparative study of five algorithms', *Journal of Water Resources Planning and Management*, vol. 133, no. 1, pp. 87–92.

Zheng, F, Zecchin, AC & Simpson, AR 2012, 'Self-adaptive differential evolution algorithm applied to water distribution system optimization', *Journal of Computing in Civil Engineering*, vol. 27, no. 2, pp. 148–158.

Appendix A

Submitted version of Publication 1: WDSLib: A Water Distribution System Simulation Test Bed

WDSLib: A Water Distribution System Simulation Test Bed

Mengning Qiu* Bradley Alexander Angus R. Simpson
Sylvan Elhay
University of Adelaide, South Australia, 5005

Highlights

- A library for the steady-state analysis of a water distribution system (WDS)
- An open-source C++ software implementation of a number of WDS solution methods.
- A fast simulation platform for both once-off and multi-run simulations
- A timing model to parameterize multiple simulation times is introduced.
- Several improvements to the existing solution methods have been made.

Abstract

WDSLib is an extensible simulation toolkit for the steady-state analysis of a water distribution system. It includes a range of solution methods: the forest-core partitioning algorithm, the global gradient algorithm, the reformulated co-tree flow method, and also combinations of these methods. WDSLib has been created using a modularized object-oriented design and implemented in the C++ programming language, and has been validated against a reference MATLAB implementation. WDSLib has been designed: (i) to avoid unnecessary computations by hoisting each

*Corresponding Author
Email: mengning.qiu@adelaide.edu.au

of the modules to its appropriate level of repetition, (ii) to perform the computations independently of measurement units using scaled variables, (iii) to accurately report the execution time of all the modules in that it is possible to produce a timing model to parameterize multiple simulation times (such as in an optimization using a genetic algorithm) from a series of sampling simulation runs and (iv) to guard against numerical failures. Two example applications, a once-off simulation and a network optimization design application simulation, are presented. This toolkit can be used (i) to implement, test and compare different solution methods, (ii) to focus the research on the most time-consuming parts of a solution method and (iii) to guide the choice of solution method when multiple simulation runs are required.

Keywords: Water Distribution System; C++ toolkit; Object-Oriented design; Forest-Core Partitioning Algorithm; Reformulated Co-tree Flows Method; Global Gradient Algorithm; open source software

Software availability

Name of the Software: WDSLib

Version: 1.0

Available from: https://github.com/a1184182/WDSLib

Language: C++

Year first available: 2018

1 Introduction

Hydraulic simulation has been used to model water distribution systems (WDSs) for several decades and is an essential tool for the design, operation, and management of WDSs in industry and research. Hydraulic simulation allows users (1) to optimize WDS network parameters, such as pipe diameters, in a design setting, (2) to calibrate network parameters, such as demand patterns, in a conventional operational setting, (3) to conduct

real-time monitoring and calibration of the network elements in a supervisory control and data acquisition (SCADA) operational setting, and (4) to adjust control devices, such as valves, in a management setting. In the design setting and both the above operational settings, repeated hydraulic assessment is required on a network with fixed topology. In the management setting, repeated hydraulic assessment is required on a network with flexible network parameter settings. With ever-increasing network sizes and the need for real-time management using a SCADA system, it is important to have a robust simulation package which can be configured to be maximally efficient whatever the setting.

In the field of hydraulic simulation, the system of equations can be formulated as a large and sparse non-linear saddle point problem. There are several well-known iteration methods for solving the non-linear saddle point problem. These include: range space methods (Global Gradient Algorithm (Todini and Pilati 1988)), Null space methods (Co-Tree flow formulation variations (Rahal 1995; Elhay et al. 2014)), loop-based methods (Loop flow correction (Cross 1936)), and pre-and-post-processing methods (forest-core partitioning algorithm (Simpson et al. 2014), domain decomposition (Diao et al. 2014), network clustering (Perelman and Ostfeld 2011)). Their relative performance in terms of speed, rate-of-convergence, and accuracy depends among other things on the topology of the target network: size of the forest component, the number of network loops, and the density of these network loops. It is difficult to evaluate the impact of these topology factors by only examining the incidence matrix that describes the pipe network connectivity. As a result, the best method to use for a particular network cannot be easily determined a priori. Moreover, extra complexity is introduced when a multi-run hydraulic assessment is required. During a multi-run hydraulic simulation, the elapsed computation time of each method can be broken down into two parts: the components that are only required to be performed once at the very beginning for the same network, called the overhead, and the components that are required to be carried out repeatedly for each separate run until the required number of iterations has been met, called the hydraulic-phase. It is desirable to have a simulation platform, given the different levels of repetition, to implement these alternative algorithms efficiently. Equipped with such a platform a user

would be able to easily benchmark the performance of alternative methods on a small number of evaluations for a given network and use that performance to inform the choice of algorithm to use for either a once-off simulation setting or for a multiple simulation setting (such as for an evolutionary algorithm (EA)).

This work describes an extensible WDS simulation platform called WDSLib. WDSLib is a numerically robust, efficient and accurate C++ library that implements many WDS simulation methods. WDSLib is written using a modular object-oriented design which allows users to easily mix and interchange solution components, thereby enabling users to avoid redundant computations. It has been optimized to use sparse data structures which are oriented to the pattern of access required for each solution method. WDSLib has been validated for accuracy on a range of realistic benchmark water distribution networks against reference implementations and tested for speed. The program accepts the input file formats of the industry standard EPANET2 (Rossman 2000) toolkit and its performance is faster than EPANET2 in all tested settings and benchmarks.

The remainder of this paper is structured as follows. The next section describes related methodologies and implementations. A general description of the WDS demand-driven steady-state problem is given in the next section. Section 3 presents a mathematical formulation of the network and the solution methods that are used in WDSLib. The tool-kit structure is then given in section 4. This is followed, in section 5, by the toolkit implementation details. Section 6 provides some examples of how the toolkit can be utilized in a simulation work flow. The results are discussed in Section 7. Finally, section 8 summarizes the results of this paper and describes future extensions to the toolkit.

2 Background

This section describes related water distribution system network solution methods and implementations. The first sub-section describes solution methods, including those used by WDSLib. This is followed by a description of currently available implementations and

compares these with WDSLib.

2.1 Related Methods

This research considers a water distribution model made up of energy conservation equations and the demand driven model continuity equations. The Hardy Cross method (Cross 1936), also known as the loop flow corrections method, is one of the oldest methods and uses successive approximations, solving for each loop flow correction independently. It is a method that was widely used for its simplicity at the time when it was introduced. More than three decades later, Epp and Fowler (1970) developed a computer version of Cross's method and replaced the numerical solver with the Newton method, which solves for all loop flow corrections simultaneously. However, this method has not been widely used because of the need (i) to identify the network loops, (ii) to find initial flows that satisfy continuity and (iii) to use pseudo-loops.

The GGA is a range space method that solves for both flows and heads. It was the first algorithm, in the field of hydraulics, to exploit the block structure of the Jacobian matrix to reduce the size of the key matrix in the linearization of the Newton method. The GGA has gained popularity through its rapid convergence rate for a wide range of starting values. This is the result of using the Newton method on an optimizations problem that has a quadratic surface. However, it was reported by Elhay and Simpson (2011) that the GGA fails catastrophically in the presence of zero flows in a WDS when the head loss is modeled by the Hazen-Williams formula. Regularization methods have been proposed by both Elhay and Simpson (2011) and Gorev et al. (2012) to deal with zero flows when the head loss is modeled by the Hazen-Williams formula.

The GGA as it was first proposed, applied only for the WDSs in which the head loss is modeled by the Hazen-Williams formula, where the resistance factor was independent of flow. Rossman (2000) extended the GGA to allow the use of the Darcy-Weisbach formula. It has been pointed out in Simpson and Elhay (2010), however, that Rossman incorrectly treated the Darcy-Weisbach resistance factor as independent of the flow. They introduced the correct Jacobian matrix to deal with this. It has been demonstrated that

once the correct Jacobian matrix is used, the quadratic convergence rate of the Newton method is restored. Furthermore, Elhay and Simpson (2011) reported that the GGA does not fail in the presence of zero flows when the derivatives of the Darcy-Weisbach Jacobian matrix are correctly computed for laminar flows.

The co-trees flow method (CTM) (Rahal 1995) is a null space method that solves for the co-tree flows and spanning tree flows separately. The CTM, unlike the loop flow corrections method, does not require the initial flows to satisfy continuity. However, it does require: (i) the identification of the associated circulating graph; (ii) the determination of the demands that are to be carried by tree branches; (iii) finding the associated chain of branches closing a circuit for each co-tree chord; (iv) computing pseudo-link head losses. The reformulated co-trees flow method (RCTM) (Elhay et al. 2014) is also a null space method that solves for co-tree flows and spanning trees flows separately. It represents a significant improvement on the CTM by removing requirements (i) to (iv) above. It uses the Schilders' factorization (Schilders 2009) to permute the node-arc incidence matrix into an invertible spanning tree block and a co-tree block. This permutation reduces the size of the Jacobian matrix from the number of junctions (as in the GGA) to approximately the number of loops in the network.

Abraham and Stoianov (2015) proposed a novel idea to speed-up the solution process when using a null space method to solve a WDS network. Their idea exploits the fact that a significant proportion of run-time is spent computing the head losses. At the same time, flows within some pipes exhibit negligible changes after a few iterations. As a result, there is no point in wasting computer resources to re-compute the pipe head losses for the pipes that have little or no change in flows. This partial update can be used to economize the computational complexity of the GGA, the RCTM and their variations.

The forest-core partitioning algorithm (FCPA) (Simpson et al. 2014) speeds up the solution process in the case where the network has a significant forest component. This algorithm permutes the system equations to partition the linear component of the problem, which is the forest of the WDS, from the non-linear component, which is the core of the WDS. It can be viewed as a method that simplifies the problem by solving for

the flows and the heads in the forest just once instead of at every iteration. The FCPA reduces the number of pipes, number of junctions, and the dimension of the Jacobian matrix in the core by the number of forest pipes (or nodes).

The graph matrix partitioning algorithm(GMPA) (Deuerlein et al. 2015) exploited the linear relationships between flows of the internal trees within the core and the flows of the corresponding super-links after the forest of the network has been removed. This was a major breakthrough. The GMPA permutes the node-arc incidence matrix in such a way that all of the nodes with degree two in the core can be treated as a group. By partitioning the network this way, the network can be solved by a global step, which solves for the nodes with degree greater than two (super nodes) and the pipes which connect to them (path chords), and a local step, which solves for the nodes with degree two (interior path nodes) and pipes connected to them (path-tree links).

2.2 Related Implementations

EPANET 2 (Rossman 2000) is a widely used WDS simulation package. EPANET 2 implemented the GGA to provide a demand-driven steady-state solution of a WDS. The code for EPANET 2 is in the public domain, allowing many extensions to be developed. Currently available extensions include: the implementation of a pressure-dependent model (Cheung et al. 2005; Morley and Tricarico 2008; Siew and Tanyimboh 2012; Jun and Guoping 2012) and a real-time simulation capability (Vassiljev and Koppel 2015).

The EPANET 2 implementation is not explicitly designed to necessarily be easy to understand or accommodate alternative solution methods (Guidolin et al. 2010). The elements that are used in EPANET 2 are stored by the variables that describe their graph properties. For example, (1) junctions, reservoirs, and tanks are stored as a C struct called *Node* and (2) all valves, pipes, and pumps are stored as a C struct called *Link*. The abundant use of global variables limits the reusability and the possibility of the thread-safe design (Guidolin et al. 2010).

Consequently, it is difficult to cleanly incorporate new solution methods into EPANET 2 in a manner that allows a fair comparison of performance between these methods.

Moreover, because there are no clearly defined interfaces for the incorporation of third-party code components in EPANET 2, there is no guarantee that independently authored extensions will be easy to combine with each other.

In the absence of a popular easy-to-modify WDS simulation platform there is currently no straightforward means for comparing different solution methods. To date, when new solution methods have been developed they have been compared using different research systems, on different platforms with different implementation languages. This leads to difficulty in comparing methods, limits the reusability of code, and creates a barrier for researchers to reproduce and replicate results. To address these issues, an extensible framework is required that allows implementation of new methodologies to be easily incorporated without an adverse impact on the performance of the rest of the system.

To this end, a number of attempts have been made to implement an object-oriented wrapper to encapsulate the EPANET 2 solver (openNet (Morley et al. 2000) and OOTEN(van Zyl et al. 2003)). However, these two systems were focused on providing more flexibility in the processing of input to the core EPANET solver. They did not address any issues relating to the solution process. CWSnet, a C++ implementation in object-oriented style, was produced by Guidolin et al. (2010) as an alternative to EPANET 2.0. In CWSnet, more attention has been given to the hydraulic elements of the WDS network. In addition, CWSNet provides a pressure driven model, and takes advantage of the computing power of the computer's Graphics Processing Unit (GPU). However, in CSWnet the data structures representing the network are specialized to the solution methods that it uses. These data structures are not easily adapted to work efficiently with the different traversal orders, and graph algorithms used by newly developed solution methods. However, CWSnet still uses the same hydraulic solver and the same linear solver techniques implemented in EPANET 2 (Guidolin et al. 2010).

To accommodate the deficiencies referred to above, this paper presents a new hydraulic simulation toolkit WDSlib. WDSlib is coded in C++, and incorporates a number of recently published techniques. This toolkit offers users the ability to: (i) choose from, or modify, different approaches and implementations of different WDS model analyses,

and (ii) extend the toolkit to include new developments. These features have been implemented using fast and modularized code. A focus of attention in this research has been program correctness, robustness and code efficiency. The correctness of the toolkit has been validated against a reference MATLAB implementation. The differences between all results (intermediate and final) produced by the C++ toolkit and the MATLAB implementation were shown to be smaller than 10^{-10}. In the interest of toolkit robustness, special attention has been paid to numerical processes to guard against avoidable failures, such as loss of significance through subtractive cancellation, and numerical errors, such as division by zero. The data structures and code libraries in WDSLib are shared and all implementations have been carefully designed to ensure fairness of performance comparisons between algorithms. WDSLib uses a pluggable architecture where solution-methods, and their accompanying pre-processing and post-processing code are easily substituted. In addition, different numerical linear algebra techniques can be incorporated using a well-defined interface. This concludes the discussion of related work. The mathematical formulations of the solution methods used in WDSLib are presented in the next section.

3 General WDS Demand-Driven Steady-State Problem

This section describes the general WDS demand-driven steady-state problem. The following starts with the basic definitions and notation, followed by the system equations. Finally, the relevant equations are shown for each of the different solution methods that are implemented in WDSLib. All variables are described in the nomenclature section in Appendix E.

3.1 Definitions and Notation

Consider a water distribution system that contains n_p pipes, n_j junctions, n_r fixed head nodes and n_f forest pipes and nodes. The $j-th$ pipe of the network can be characterized by its diameter D_j, length L_j, resistance factor r_j. The $i-th$ node of the network has two properties: its nodal demand d_i and its elevation z_i.

Let $\boldsymbol{q} = (q_1, q_2, ...q_{n_p})^T$ denote the vector of unknown flows, $\boldsymbol{h} = (h_1, h_2, ...h_{n_j})^T$ denote the vector of unknown heads, $\boldsymbol{r} = (r_1, r_2, ...r_{n_p})^T$ denote the vector of resistance factors, $\boldsymbol{d} = (d_1, d_2, ...d_{n_j})^T$ denote the vector of nodal demands, $\boldsymbol{e_l} = (e_{l_1}, e_{l_2}....e_{l_{n_f}})^T$ denote the vector of fixed head elevations.

The head loss exponent n is assumed to be dependent only on the head loss model: $n = 2$ for the Darcy-Weisbach head loss model and $n = 1.852$ for Hazen-Williams head loss model. The head loss within the pipe j, which connects the node i and the node k, is modelled by $h_i - h_k = r_j q_j |q_j|^{n-1}$. Denote by $\boldsymbol{G}(\boldsymbol{q}) \in \mathbb{R}^{n_p \times n_p}$, a diagonal square matrix with element $[\boldsymbol{G}]_{jj} = r_j |q_j|^{n-1}$ for $j = 1, 2, ...n_p$. Denote by $\boldsymbol{F}(\boldsymbol{q}) \in \mathbb{R}^{n_p \times n_p}$, a diagonal square matrix where the j-th element on its diagonal $[\boldsymbol{F}]_{jj} = \frac{d}{dq_j}[\boldsymbol{G}]_{jj} q_j$. $\boldsymbol{A_1}$ is the full rank, unknown head, node-arc incidence matrix, where $[\boldsymbol{A_1}]_{ji}$ is used to represent the relationship between pipe j and node i; $[\boldsymbol{A_1}]_{ji} = -1$ if pipe j enters node i, $[\boldsymbol{A_1}]_{ji} = 1$ if pipe j leaves node i, and $[\boldsymbol{A_1}]_{ji} = 0$ if pipe j is not connected to node i. $\boldsymbol{A_2}$ is the fixed-head node-arc incidence matrix, where $[\boldsymbol{A_2}]_{ji}$ is used to represent the relationship between pipe j and fixed head node i, $[\boldsymbol{A_2}]_{ji} = -1$ if pipe j enters fixed head node i, $[\boldsymbol{A_2}]_{ji} = 1$ if pipe j leaves fixed head node i, and $[\boldsymbol{A_2}]_{ji} = 0$ if pipe j is not connected to fixed head node i.

3.2 System of Equations

The steady-state flows and heads in the WDS system are modeled by the demand-driven model (DDM) continuity equations (1) and the energy conservation equations (2):

$$-\boldsymbol{A_1}^T \boldsymbol{q} - \boldsymbol{d} = \boldsymbol{O} \tag{1}$$

$$\boldsymbol{G}(\boldsymbol{q})\boldsymbol{q} - \boldsymbol{A_1}\boldsymbol{h} - \boldsymbol{A_2}\boldsymbol{e_l} = \boldsymbol{O}, \tag{2}$$

which can be expressed as

$$\begin{pmatrix} \boldsymbol{G}(\boldsymbol{q}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{h} \end{pmatrix} - \begin{pmatrix} \boldsymbol{A_2}\boldsymbol{e_l} \\ \boldsymbol{d} \end{pmatrix} = 0, \tag{3}$$

where its Jacobian matrix is

$$\boldsymbol{J} = \begin{pmatrix} \boldsymbol{F}(\boldsymbol{q}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \tag{4}$$

and it is sometimes referred to as a nonlinear saddle point problem (Benzi et al. 2005).

This non-linear system is normally solved by the Newton method, in which $\boldsymbol{q}^{m+1}$ and $\boldsymbol{h}^{m+1}$ are repeatedly computed from $\boldsymbol{q}^m$ and $\boldsymbol{h}^m$ by

$$\begin{pmatrix} \boldsymbol{F}^{(m)}(\boldsymbol{q}^m) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}^{(m+1)} - \boldsymbol{q}^{(m)} \\ \boldsymbol{h}^{(m+1)} - \boldsymbol{h}^{(m)} \end{pmatrix} = - \begin{pmatrix} \boldsymbol{G}^{(m)}\boldsymbol{q}^{(m)} - \boldsymbol{A_1}\boldsymbol{h}^{(m)} - \boldsymbol{A_2}\boldsymbol{e_l} \\ -\boldsymbol{A_1}^T\boldsymbol{q}^{(m)} - \boldsymbol{d}, \end{pmatrix} \tag{5}$$

until the relative differences $\frac{||\boldsymbol{q}^{(m+1)} - \boldsymbol{q}^{(m)}||}{||\boldsymbol{q}^{(m+1)}||}$ and $\frac{||\boldsymbol{h}^{(m+1)} - \boldsymbol{h}^{(m)}||}{||\boldsymbol{h}^{(m+1)}||}$ are sufficiently small.

3.3 Global Gradient Algorithm

Todini and Pilati (1988) applied block elimination to Eq. (5) to yield a two-step Hazen-William solver: Eq. (6) for the heads and Eq. (7) for the flows.

$$\boldsymbol{h}^{(m+1)} = \boldsymbol{U}^{-1}\left\{-n\boldsymbol{d} + \boldsymbol{A_1}^T[(1-n)\boldsymbol{q}^{(m)} - \boldsymbol{G}^{-1}\boldsymbol{A_2}\boldsymbol{e_l}]\right\} \tag{6}$$

where $\boldsymbol{U} = \boldsymbol{A_1}^T\boldsymbol{G}^{-1}\boldsymbol{A_1}$

$$\boldsymbol{q}^{(m+1)} = \frac{1}{n}\left\{(n-1)\boldsymbol{q}^{(m)} + \boldsymbol{G}^{-1}(\boldsymbol{A_2}\boldsymbol{e_l} + \boldsymbol{A_1}\boldsymbol{h})\right\} \tag{7}$$

Later, Simpson and Elhay (2010) proposed

$$\boldsymbol{V}\boldsymbol{h}^{(m+1)} = -\boldsymbol{d} + \boldsymbol{A_1}^T\boldsymbol{F}^{-1}\left[(\boldsymbol{G} - \boldsymbol{F})\,\boldsymbol{q}^{(m)} - \boldsymbol{A_2}\boldsymbol{e_l}\right] \tag{8}$$

where $\boldsymbol{V} = \boldsymbol{A_1}^T\boldsymbol{F}^{-1}\boldsymbol{A_1}$

$$\boldsymbol{q}^{(m+1)} = \boldsymbol{q}^{(m)} + \boldsymbol{F}^{-1}\boldsymbol{A_1}\boldsymbol{h}^{(m+1)} - \boldsymbol{F}^{-1}\left[\boldsymbol{G}\boldsymbol{q}^{(m)} - \boldsymbol{A_2}\boldsymbol{e_l}\right] \tag{9}$$

as the generalized equations that can be applied when the head-loss is modeled by the Hazen-Williams equation or the Darcy-Weisbach equation. The correct Jacobian matrix with the formula for $\boldsymbol{F}$, when head loss is modeled by Darcy-Weisbach equation, can be found in Simpson and Elhay (2010). They showed that the use of the correct Jacobian matrix restores the quadratic rate of convergence.

It is important to note that the GGA, as it was originally proposed, solves the entire network by a non-linear solver, and this can include some unnecessary computations which can be avoided by exploiting the structural properties of the WDS graph composition. The methods described below exploit these structural properties to potentially improve the speed of the solution process.

3.4 Forest-Core Partitioning

Associated with a WDS is a graph $G = (V, E)$, where the elements of V are the nodes (vertices) of the graph G and elements of E are the pipes (links) of the graph G. The graph G can be partitioned into smaller subgraphs with special properties. The special properties that are exploited in WDSLib and their formulations are described in this subsection. The concept of partitioning the WDS network was proposed by Deuerlein (2008) in order to simplify the WDS solution process. Simpson et al. (2014) extended the idea of the network partitioning of Deuerlein (2008) and introduced the forest-core partitioning algorithm (FCPA), which partitions the network into a treed component and a looped or core component. The FCPA starts with a searching algorithm which identifies the forest subgraph, $G_f = (V_f, E_f)$, in which $\boldsymbol{S} \in \mathbb{N}^{n_f \times n_p}$ is the permutation matrix which identifies the pipes in the forest, E_f, as distinct from the pipes in the core , E_c, and $\boldsymbol{T} \in \mathbb{N}^{n_f \times n_j}$ is the permutation matrix which identifies the nodes in the forest, V_f, as distinct from the nodes in the core, V_c, as distinct from the core subgraph, $G_c = (V_c, E_c)$, in which $\boldsymbol{P} \in \mathbb{N}^{n_{p_c} \times n_p}$ is the permutation matrix for E_c and $\boldsymbol{C} \in \mathbb{N}^{n_{j_c} \times n_j}$ is the permutation matrix for V_c.

The flows of the pipes in the forest, $\boldsymbol{Sq}$, can be found directly from

$$\boldsymbol{Sq} = -\left(\boldsymbol{T}{\boldsymbol{A_1}}^T\boldsymbol{S}^T\right)^{-1}\boldsymbol{Td}. \tag{10}$$

The system for the reduced non-linear problem (for the core heads and flows) can be expressed as

$$\begin{pmatrix} \boldsymbol{PGP}^T & -\boldsymbol{PA_1C}^T \\ -\boldsymbol{C}{\boldsymbol{A_1}}^T\boldsymbol{P} & \boldsymbol{O} \end{pmatrix}\begin{pmatrix} \boldsymbol{Pq} \\ \boldsymbol{Ch} \end{pmatrix} = \begin{pmatrix} \boldsymbol{PA_2e_l} \\ \boldsymbol{Cd} + \boldsymbol{C}{\boldsymbol{A_1}}^T\boldsymbol{S}^T\boldsymbol{Sq} \end{pmatrix}, \tag{11}$$

and then the Newton iterative method is applied to Eq. (11).

Finally, once the iterative solution process for the core has stopped, the forest heads can be found by solving a linear system:

$$\boldsymbol{Th} = \left(-\boldsymbol{SA_1T}^T\right)^{-1}\left(\boldsymbol{SA_2e_l} - \boldsymbol{SGS}^T\boldsymbol{Sq} + \boldsymbol{SA_1C}^T\boldsymbol{Ch}\right). \tag{12}$$

The system for the reduced non-linear problem (for the core heads and flows) in Eq. 11 can be expressed as:

$$\begin{pmatrix} \widehat{\boldsymbol{G}} & -\widehat{\boldsymbol{A}}_{\boldsymbol{1}} \\ -\widehat{\boldsymbol{A}}_{\boldsymbol{1}}{}^T & \boldsymbol{O} \end{pmatrix}\begin{pmatrix} \widehat{\boldsymbol{q}} \\ \widehat{\boldsymbol{h}} \end{pmatrix} = \begin{pmatrix} \widehat{\boldsymbol{A}}_{\boldsymbol{2}}\boldsymbol{e_l} \\ \widehat{\boldsymbol{d}} \end{pmatrix} \tag{13}$$

where $\widehat{\boldsymbol{G}} = \boldsymbol{PGP}^T, \widehat{\boldsymbol{A}}_{\boldsymbol{1}} = \boldsymbol{PA_1C}^T, \widehat{\boldsymbol{q}} = \boldsymbol{Pq}, \widehat{\boldsymbol{h}} = \boldsymbol{Ch}, \widehat{\boldsymbol{A}}_{\boldsymbol{2}} = \boldsymbol{PA_2}$, and $\widehat{\boldsymbol{d}} = \boldsymbol{Cd} + \boldsymbol{C}{\boldsymbol{A_1}}^T\boldsymbol{S}^T\boldsymbol{Sq}$.

The FCPA simplifies the problem by identifying the linear part of the problem and solving it separately from the core to avoid unnecessary computation in the iterative process.

3.5 Reformulated Co-Tree flows method

A graph, with or without forest, can be partitioned into two sub-graphs: a spanning tree subgraph and a complementary co-tree subgraph. The reformulated co-tree flow method (RCTM) (Elhay et al. 2014) exploited the relationship between the spanning tree

pipes and the co-tree pipes. The RCTM starts with a spanning tree search algorithm which identifies a spanning tree subgraph, $G_{st} = (V, E_{st})$, in which $\boldsymbol{K_1} \in \mathbb{N}^{n_{p_{st}} \times n_p}$ is the permutation matrix that identifies the pipes in the spanning tree, E_{st}, as distinct from the pipes in the co-tree, E_{ct}. $\boldsymbol{R}$ is the permutation matrix for the nodes which traverse the same sequence as the corresponding spanning tree pipes, E_{st}, and $\boldsymbol{K_2} \in \mathbb{N}^{n_{p_{ct}} \times n_p}$ is the permutation matrix for the pipes in the complementary co-tree edges, E_{ct}. It is important to note that there are many choices of spanning tree for any cyclic graph. The choice of spanning tree and co-tree combination does not affect the correctness of the method.

By exploiting the relationship between the spanning tree and cotree, Elhay et al. (2014) proposed the following equations to solve the WDS for the flows: first for the spanning tree flows $\boldsymbol{q_1^{(m+1)}}$,

$$\boldsymbol{q}_1^{(m+1)} = \boldsymbol{L_{21}}^T \boldsymbol{q}_2^{(m)} - \boldsymbol{R}_1^{-T} \widehat{\boldsymbol{d}} \tag{14}$$

and second for the co-tree flows $\boldsymbol{q_2^{(m+1)}}$:

$$\boldsymbol{W}^{(m+1)} \boldsymbol{q}_2^{(m+1)} = \boldsymbol{L_{21}} \left(\boldsymbol{F}_1^{(m+1)} - \boldsymbol{G}_1^{(m+1)} \right) \boldsymbol{q}_1^{(m+1)} + \left(\boldsymbol{F}_2^{(m)} - \boldsymbol{G}_2^{(m)} \right) \boldsymbol{q}_2^{(m)} + \boldsymbol{a_2} \tag{15}$$

where: $\boldsymbol{R}_1 = \boldsymbol{K_1} \hat{\boldsymbol{A}}_1 \boldsymbol{R}^T$; $\boldsymbol{R}_2 = \boldsymbol{K_2} \hat{\boldsymbol{A}}_1 \boldsymbol{R}^T$; $\boldsymbol{L_{21}} = -\boldsymbol{R}_2 \boldsymbol{R}_1^{-T}$; $\boldsymbol{F}_1^{(m)} = \boldsymbol{K_1} \widehat{\boldsymbol{F}^{(m)}} \boldsymbol{K_1}^T$; $\boldsymbol{F}_2^{(m)} = \boldsymbol{K_2} \widehat{\boldsymbol{F}^{(m)}} \boldsymbol{K_2}^T$ $\boldsymbol{G}_1^{(m)} = \boldsymbol{K_1} \widehat{\boldsymbol{G}^{(m)}} \boldsymbol{K_1}^T$; $\boldsymbol{G}_2^{(m)} = \boldsymbol{K_2} \widehat{\boldsymbol{G}^{(m)}} \boldsymbol{K_2}^T$; $\boldsymbol{W}^{(m)} = \boldsymbol{L_{21}} (\boldsymbol{F}_1^{(m)})^{-1} \boldsymbol{L_{21}}^T + (\boldsymbol{F}_2^{(m)})^{-1}$; $\boldsymbol{a_1} = \boldsymbol{K_1} \hat{\boldsymbol{A}}_2 \boldsymbol{e_l}$; $\boldsymbol{a_2} = \boldsymbol{L_{21}} \boldsymbol{K_1} \hat{\boldsymbol{A}}_2 \boldsymbol{e_l} + \boldsymbol{K_2} \hat{\boldsymbol{A}}_2 \boldsymbol{e_l}$.

Note that in Eq. (14), an initial set of the co-tree flows $\boldsymbol{q_2^{(0)}}$ is needed to commence the solution process.

The heads are found after the iterative process of the RCTM has been completed by using a linear solution process:

$$\boldsymbol{R}_1 \boldsymbol{h} = \boldsymbol{F}_1 \boldsymbol{q}_1^{(m+1)} - (\boldsymbol{F}_1 - \boldsymbol{G}_1) \boldsymbol{q}_1^{(m)} - \boldsymbol{a_1} \tag{16}$$

This partitioning of the network equations reduces the size of the non-linear compo-

nent of the solver to $n_p - n_j$ (the number of co-tree elements in the network). It has been proven by Elhay et al. (2014) that the RCTM and the GGA have identical iterative results and solutions if the same starting values are used. However, for the RCTM, the user only needs to set the initial flow estimates for the co-tree pipes, $\boldsymbol{q_2^{(0)}}$, in contrast to GGA where initial flow estimates are required for all pipes. The flows in the complementary spanning pipes are generated by Eq.(14) in the RCTM.

4 WDSLib Structure

WDSLib is a WDS simulation toolkit consisting of a set of C++ member functions, which henceforth will be referred to just as *functions*, that can be composed to solve for the steady state solution of a WDS. WDSLib can be used for a once-off simulation or a multi-run simulation. Pre-packaged driver code is provided to perform once-off simulations using a choice of solver methods. For a multi-simulation setting, where the use-cases are very diverse, the user is able to select the desired components of WDSLib to compose and compile their own driver.

Individual functions in WDSLib are classified according to their role in the simulation workflow. In any simulation workflow, there will be functions that will only have to be executed once. For example, functions to read the input file or partition the network will only have to execute once at the start of the simulation (or of all simulations). Likewise, code to reverse the network partitioning and write simulation results will only have to execute once at the end of the simulation. In this work, these functions that are only required to be run once are called level one (L_1) functions. L_1 functions relate to network topology, which is invariant for the whole simulation. In a multi-simulation setting, certain functions will need to be run once for every hydraulic-phase. An example of such a module is the module making the initial guesses of pipe flow rates for the updated network configuration. In this work, these, once-per-assessment functions, are called level two (L_2) functions.

Finally, for every hydraulic assessment there is a non-linear iterative phase in the so-

lution process. The functions in this phase run many times for each hydraulic assessment until the stopping test has been satisfied. Examples of these include the functions to calculate the $\boldsymbol{G}$ and $\boldsymbol{F}$ matrices (see Eqs. (3) and (4)) and running the Cholesky solver. These iterative-phase functions are called level three (L_3) functions.

Fig. 1 illustrates the global structure of WDSLib under a once-off simulation setting and a multi-run simulation setting. The modular setup of WDSLib allows each module to be run the minimum number of times determined by its simulation setting. Under the module structure described above a once-off simulation setting can be viewed as a special case where the L_1 functions and L_2 functions are both run once. Note that after running the initial L_1 functions it is possible to run hydraulic assessments of the network in parallel. This mode of execution might be used in a design setting such as using a genetic algorithm (GA) to optimize pipe diameter sizes.

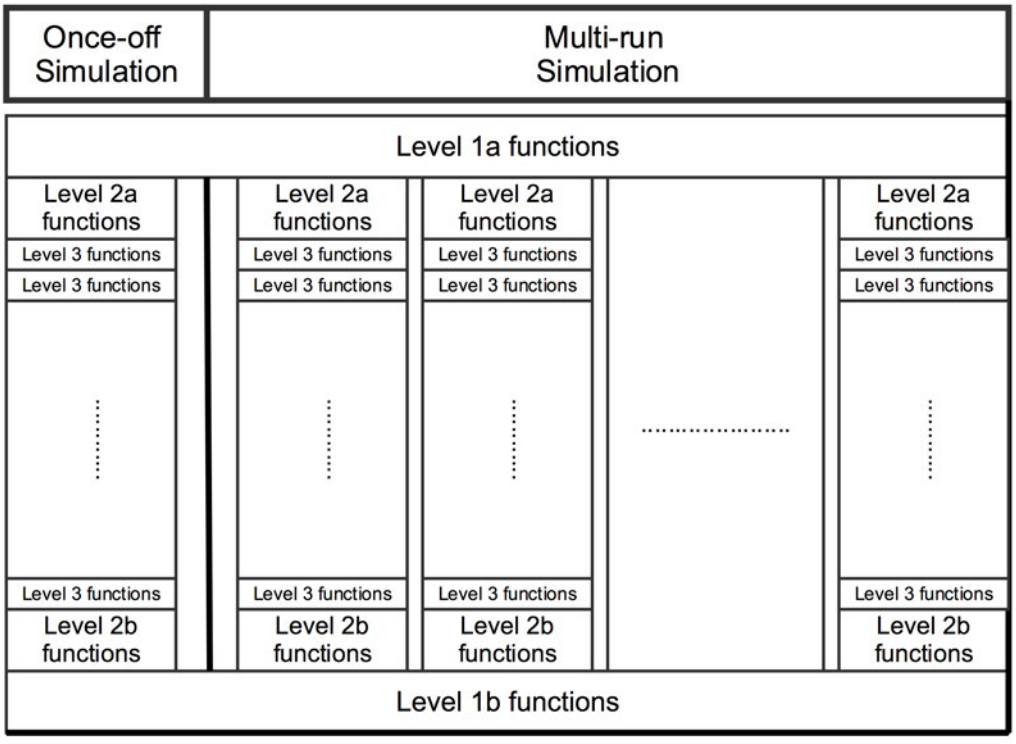


Figure 1: Global structure of WDSLib for both simulation settings

L_1 and L_2 functions are classified into parts a and b according to whether they run before or after the lower level processing that they embed. These functions are detailed in Fig. 2. The L_1 functions that run at the start of the simulation are called L_{1a} func-

tions. These include the module to read the configuration file and the EPANET *.inp* file; partition the network; and solve the linear part(s) of the network. The corresponding L_{1b} functions are run at the end of the simulation. These include tasks such as reversing the network partitioning. Note that certain L_{1a} functions require their corresponding L_{1b} functions to be used. For example the forest search module needs to be paired with the reverse FCPA permutation. There is a similar structure for L_2 functions. L_{2a} functions are run at the start of each hydraulic assessment and L_{2b} functions run at the end. The functions that must be included for the FCPA method are denoted with single asterisks. Likewise the functions that must be included with the RCTM method are denoted with double asterisks. For these methods to work correctly all affiliated functions must be included in the simulation workflow. Note that it is also possible to run both the RCTM and FCPA in the same workflow. Also note that the user cannot run both GGA and RCTM in the same workflow – the user must choose between these solution methods.

Table 1 provides a mapping from the function descriptions in Fig. 2 to the function names in WDSLib. In addition, the dependencies between functions for each solution method are shown in Table 1a, Table 1b, Table 1c and Table 1d. The columns in each table list, respectively, the description of the function, its name in WDSLib, the C++ class in which it appears, its input parameters, and its output values. Note, that *void* is used in these latter two columns to denote that the function interacts with the class variables rather than through its parameters and return value. Examples of how these functions can be coded are presented in section 6. The key data-access functions in WDSLib are described next.

Getter and Setter methods Each class in WDSLib has various methods available for setting the network parameters and retrieving the results of the WDS network. These methods allow the user to reconfigure the network before and during simulation runs. The names of the setter methods all start with a prefix *set* and the names of the getter methods all start with a prefix *get*. For example, a user can set (write-to) the diameter of pipe *index* to *value* by calling `pipe->setD(index,value)` and get (read-from) the head of node *index* by calling `h[index]=result->gethFinal(index)`. A summary of

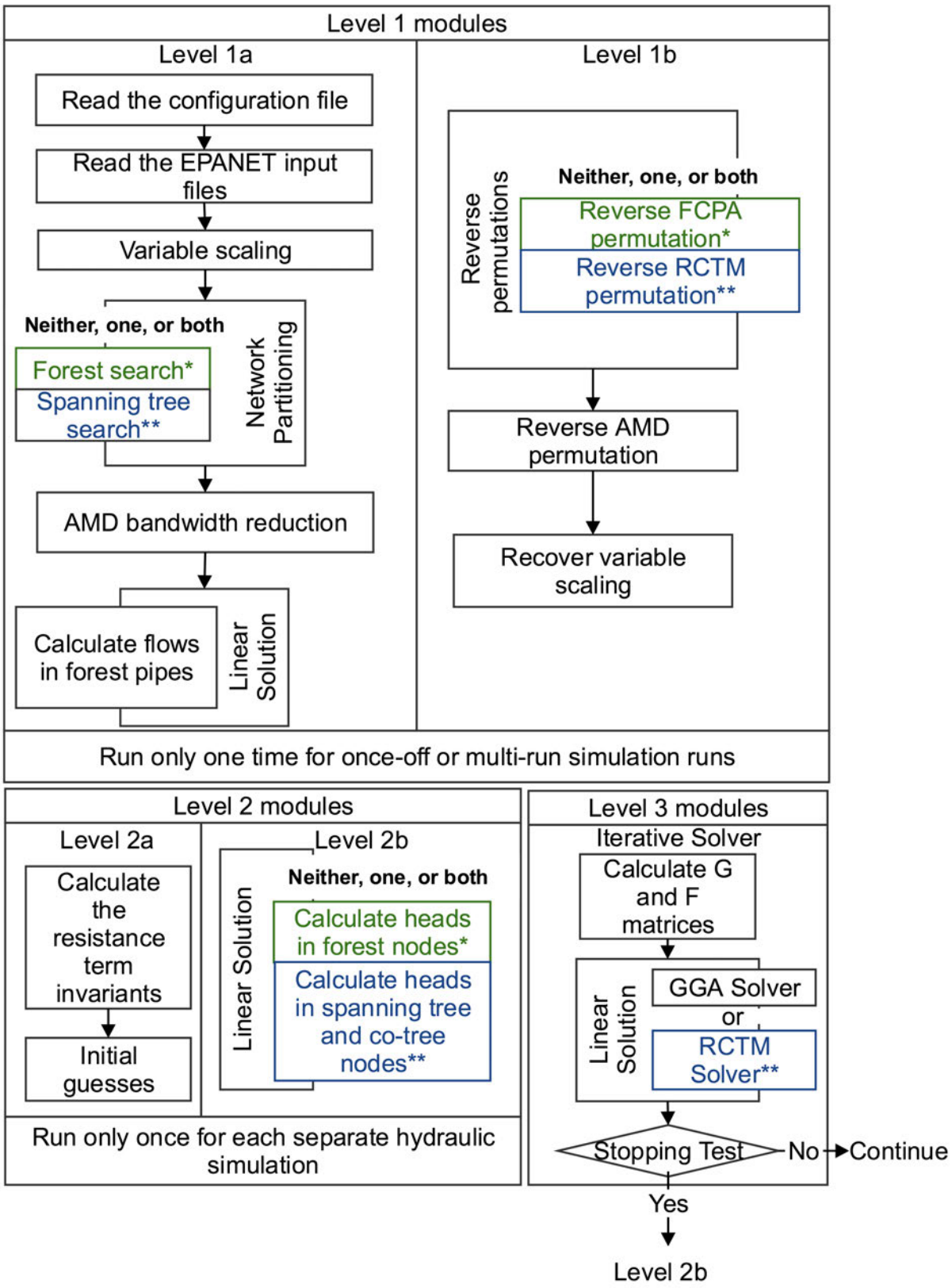


Figure 2: Function classification in WDSLib. The functions marked with single asterisks must be used for the FCPA method. The functions marked with double asterisks must be used for the RCTM method. Note that it is possible to use both methods at the same time.

Table 1: Key function descriptions, names, their classes, inputs and outputs. The affiliated functions are shown in sub-tables (1a) (1b) (1c) (1d).

(a) Shared Modules

Description	Module name	Class	Input	Output
Read the configuration file	readConfig	runManager	config file name	void
Read EPANET input file	getInputData	Input	EPANET *.inp* file	EPANET err code
Variables scaling	scale	Solver	void	void
AMD bandwidth reduction	AMD	Suitesparse	void	void
Calculate the resistance constants	getRf	Solver	net	resistance constant
Generate initial guesses of flows	init	Solver	diameter	flow rate
Calculate the head loss coefficients	getGF	Solver	net, resistance constant	void
Stopping test	stopTest	Solver	result	norm
Recover scaled variables	rScale	Solver	void	void

(b) Global gradient algorithm (GGA)

Description	Module name	Class	Input	Output
GGA Solver	runH	GGASolver(Solver)	void	void

(c) Forest-core algorithm (FCPA)

Description	Module name	Class	Input	Output
Forest search	forestSearch	topology	SN, EN	void
Calculate flows in forest	forestFlow	solver	demands	flows in forest pipes
Calculate heads in forest	forestHead	solv	result	heads in forest pipes
Reverse FCPA permutation	rFCPA	Solver	void	void

(d) Reformuated cotree flows method (RCTM)

Description	Module name	Class	Input	Output
Spanning tree search	STSearch	topology	SN, EN	void
RCTM solver	runH	RCTMsolver (Solver)	void	void
Calculate heads in ST and CT	RCTMHead	RCTMsolver	flows in ST and CT	void
Reverse RCTM permutation	rRCTM	RCTMsolver	void	void

Table 2: The getter and setter functions of each class and the variables they access

Class Name	Description	Read-Access	Write-Access
`Net`	Basic network properties, & `Pipe and Node`	`Node,Pipe,` n_p, n_j, n_s	
`Node`	Node properties	$\boldsymbol{d}$, $\boldsymbol{z_s}$, $\boldsymbol{z_u}$	
`Pipe`	Pipe properties	$\boldsymbol{SN}$, $\boldsymbol{EN}$, $\boldsymbol{L}$, $\boldsymbol{D}$, $\boldsymbol{R}$, pipe ID	
`Flag`	Flag information	getFlag("flagN",flagV)	setFlag("flagN")
`Parameter`	Parameter information	getPara("paraN",paraV)	setPara("paraN")
`Simulation`	Manage hydraulic simulation	-	-
`Solver`	Parent class of solution methods	-	-
`GGASolver`	GGA solution method	`Result`	-
`RCTMSolver`	RCTM solution method	`Result`	-
`Topology`	Network topology information	getCore, getForest	
`Result`	Results of the simulations	qIter, hIter, GIter , FIter, numIter, CresIter, EresIter, Time	

the variables that can be read-from (read-access through getter methods) and written-to (write-access through setter methods for each key classes is specified in Table 2. This concludes the discussion of the the broad structure of the WDSLib package. The next section describes key aspects of the implementation of the package.

5 WDSLIB: Toolkit Implementation

This section outlines key implementation details of WDSLib. As previously mentioned, the overall aim of WDSLib is to provide a clearly-structured, flexible and extensible hydraulic simulation toolkit that allows testing, evaluation, and use, in production settings, of both existing and new WDS solution techniques. These aims require WDSLib be implemented so that it is fast to execute, flexible to configure, robust to challenging input data cases, and easy to understand and modify. The following describes aspects of the implementation of WDSLib that enable it to meet these requirements. The next subsection describes the general considerations that informed the design of the whole toolkit. This general discussion is followed by a summary of key improvements to the solution processes encoded in forest searching and spanning tree searching in the WDSLib package.

5.1 General capabilities and properties

This sub-section describes design aspects underpinning the utility and performance of WDSLib. In-turn, the following outlines measures taken to: (1) maximize code clarity and modularity; (2) increase the efficiency of memory access and storage; (3) maximize numerical robustness; (4) facilitate accurate timing of code execution; and (5) maximize simulation speed for different settings.

Design Considerations 1: Modularity

The modular design of WDSLib is central to the evaluation and testing of different WDS solution methods. All methods have been defined to perform a single, well–defined, function and each class can be compiled, used and tested independently. These features allow users to assemble the methods of interest from independently developed components to create a customized WDS solution method in a reliable way. WDSLib's modular design also allows the users to profile the computation time of each individual component of an algorithm. Functions communicate through well-defined interfaces and the function code has been factored to minimize development and testing cost. This architecture allows customized simulation applications (i) to combine the functions of interest and (ii) to implement new solution algorithms to extend the functionalities of WDSLib.

Design Considerations 2: Memory Considerations

Care was taken to minimize the memory footprint of executing code (in order to reduce memory requirements and prevent memory leaks) in the interest of the toolkit efficiency and toolkit robustness. Reducing memory requirements allows the solution of larger WDS problems for a given memory capacity. In WDSLib, memory reduction was achieved through both, using sparse matrix representations and the systematic allocation and deallocation of working structures in the C++ code. The matrices used in WDS simulation are often sparse, with the density of the full node-arc incidence matrix being only $2/n_j$. Consequently, it is more efficient to store these matrices using sparse storage schemes which store only the non-zero elements of the matrix and pointers to their

428 locations (Davis et al. 2014). It is important to note that the choice of a sparse ma-
429 trix representation is made based on (1) the storage requirements of the matrix and (2)
430 common search orders to column elements and row elements. This latter factor means
431 that the best format for sparse matrix representation varies with the preponderant or-
432 ders of search, (row-wise, column-wise, or both), employed by each method. There is
433 a number of common storage formats for sparse matrices (Compressed column storage
434 (CCS) of Duff et al. (1989)), Compressed row storage (CRS), Block Compressed column
435 storage (BCCS), Block Compressed row storage (BCRS), and Adjacency lists). As will
436 be described shortly, WDSLib, uses a modified adjacency-list representation.

437 Other implementations use a variety of storage schemes. In EPANET 2, the $\boldsymbol{A_1}$
438 matrix is stored as two arrays of node indices, which represent start nodes (SN) and the
439 end nodes (EN) of each pipe. The $i-th$ entry of the SN and EN arrays represent the
440 start node and end node of $i-th$ pipe of the network. This storage format minimizes
441 the memory required to store the $\boldsymbol{A_1}$ matrix because only the indices are required to
442 be stored because $[\boldsymbol{A_1}]_{(i,SN_i)} = -1$ and $[\boldsymbol{A_1}]_{(i,EN_i)} = 1$. As shown in Table 4, searching
443 through rows (pipes) of matrices that are stored in this format is efficient. However,
444 searching though the columns (nodes) is relatively inefficient. This storage format is also
445 used in CWSnet.

446 Both CCS and CRS are used in the FCPA implementation reported in Simpson
447 et al. (2014), and the RCTM implementation reported in Elhay et al. (2014). The
448 partial update null space method (Abraham and Stoianov 2015) used CCS. The memory
449 requirement for storing the $\boldsymbol{A_1}$ matrix in CCS is $2 \times nnz + n_j + 1$ as shown in Table 4.
450 This storage scheme is fast for searching through columns (nodes) of matrices that are
451 stored in CCS and slow for searching though rows (pipes).

452 In WDSLib, a modified adjacency list, described in Table 3, tailored for WDS hy-
453 draulic simulation, is used. An adjacency list for an undirected and unweighted graph
454 consists of n_j unordered lists for each vertex n_i, which contains all the vertices to which
455 vertex n_i is adjacent. The network that is shown in the Fig. 3 has one source, three
456 nodes, and four pipes. The adjacency list for this network can be described by four lists

Table 3: The adjacency-list matrix presentation

Node Index	adjacent to	Size
1	$\{(v_i, e_j) \mid v_i \in N(v_1)\ e_j$ connects v_1 and $v_i\}$	$\mathrm{Deg}(v_1)$
2	$\{(v_i, e_j) \mid v_i \in N(v_2)\ e_j$ connects v_2 and $v_i\}$	$\mathrm{Deg}(v_2)$
3	$\{(v_i, e_j) \mid v_i \in N(v_3)\ e_j$ connects v_3 and $v_i\}$	$\mathrm{Deg}(v_3)$
$\vdots$	$\vdots$	$\vdots$
n_j	$\{(v_i, e_j) \mid v_i \in N(v_{n_j})\ e_j$ connects v_{n_j} and $v_i\}$	$\mathrm{Deg}(v_{n_j})$

$\{\{2,3\},\{1,4\},\{1,4\},\{2,3\}\}$. Each list describes the set of adjacent vertices of a vertex in the graph. For example, the first list, $\{2,3\}$, represents that the vertex 1 is adjacent to the vertex 2 and vertex 3.

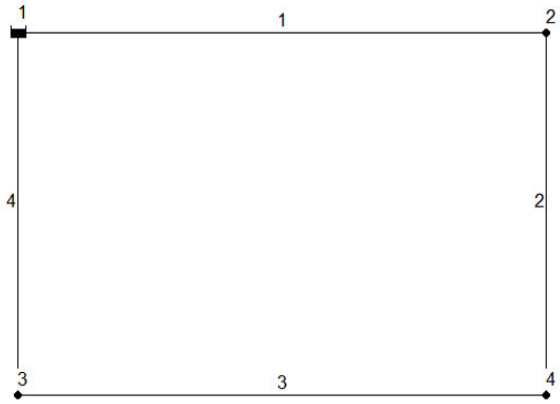


Figure 3: A simple sample network. Numbers denote junction and pipe indices in the network.

The adjacency list is modified to include a directed and weighted graph for WDSLib. This modified adjacency list for a directed and weighted WDS graph consists of n_j unordered lists for each vertex n_i. This list contains all the vertex and edge pairs to which vertex n_i is adjacent. For example, the adjacency list for the same network that is shown in the Fig. 3 can be described by four lists $\{\{(2,1),(3,4)\},\{(1,1),(4,2)\},\{((1,4),(4,3)\},\{(2,2),(3,3)\}\}$. Each list represents the set of adjacent vertex and edge pair of a vertex in the graph. For example, the first list, $\{(2,1),(3,4)\}$, describes that the vertex 1 is adjacent to the vertex 2 by edge 1 and the vertex 3 by edge 4. It is fast to search through both the rows and columns of the $\boldsymbol{A_1}$ matrices that are stored in this format.

In addition to these optimized encodings, both $\boldsymbol{G}$ and $\boldsymbol{F}$ are diagonal square matrices,

Table 4: Different sparse representations for $\boldsymbol{A_1}$

Types	size($\boldsymbol{A_1}$)	size($\boldsymbol{A_2}$)	size($[\boldsymbol{A_1}\ \boldsymbol{A_2}]$)	Column Search	Row Search
CCS	$2 \times nnz + n_j + 1$	$2 \times nnz + n_f + 1$	$4 \times n_p + n_n + 2$	$O(n)$	$O((n_j)n)$
CRS	$2 \times nnz + n_p + 1$	$2 \times nnz + n_p + 1$	$6 \times n_p + 2$	$O((n_p)n)$	$O(n)$
EPANET	-	-	$2 \times n_p$	$O(n)$	$O((n_j)n)$
WDSlib	-	-	$4 \times n_p$	$O(n)$	$O(n)$

which require less storage when stored as vectors than in sparse matrix form. The storage methods used for the variables in WDSLib and their associated memory usage are given in Table 5.

As a final note, to offer further assurance of the correctness of memory management in WDSLib, Valgrind (Nethercote and Seward 2007), a programming debugging tool, was deployed during testing to detect any memory leaks, memory corruption, and double-freeing.

Table 5: Vectors and matrices in WDSLib

variables	type	size	storage method	memory requirements
$\boldsymbol{q}$, $\boldsymbol{L}$, $\boldsymbol{D}$, $\boldsymbol{r}$	vector	$n_p \times 1$	vector	$n_p\times$ double
$\boldsymbol{h}$, $\boldsymbol{d}$	vector	$n_j \times 1$	vector	$n_j\times$ double
$\boldsymbol{G}$, $\boldsymbol{F}$	matrix	$n_p \times n_p$	vector	$n_p\times$ double
$\boldsymbol{A_1}$, $\boldsymbol{A_2}$	matrix	$n_p \times n_j$	sparse matrix	$(2 \times n_p)\times$ integer
$\boldsymbol{L_{21}}$	matrix	$(n_p - n_j) \times n_j$	sparse matrix	$\leq (n_p - n_j) \times n_j \times integer$

Design Considerations 3: Numerical Considerations

The calculations in WDSLib are performed in C++ under IEEE-standard double precision floating point arithmetic with machine epsilon $\epsilon_{mach} = 2.22 \times 10^{-16}$. Invariant terms and parameters in every equation were evaluated in advance and replaced by full 20-decimal digit accuracy constants. Intermediate results of calculations, (which are not easily accessible in EPANET), can be output at the user's request. The stopping tolerance and stopping test can be set by the user either through the configuration file or by the relevant setter method in the `Parameter` class.

Table 6: WDS variables and units

Variables	SI unit	US Customary unit	Scaling factor
Length	m	ft	$L_0 = max\,(\boldsymbol{L})$
Diameter	m	ft	$D_0 = max\,(\boldsymbol{D})$
Nodal head	m	ft	$h_0 = max\,(\boldsymbol{e_l})$
Source elevation	m	ft	$e_{l_0} = max\,(\boldsymbol{e_l})$
flow	m^3/s	ft^3/s	$q_0 = max\,(\boldsymbol{d})$
demand	m^3/s	ft^3/s	$d_0 = max\,(\boldsymbol{d})$
G, F	s/m^2	s/ft^2	$G_0 = \frac{L_0}{D_0^P}\lvert q_0\rvert^{n-1}$

In the construction of any numerical solver, there are two primary dangers that are associated with floating point arithmetic that cannot be ignored: (i) subtractive cancellation and (ii) overflow and underflow. To avoid problems associated with these, all input variables are scaled to a similar range to minimize the risk of avoidable computational inaccuracy or failure in floating point arithmetic. It is important to note that unscaled or poorly scaled variables can unnecessarily confound a computation. These scaled input variables are physically dimensionless, which allows computation which is independent of the system of measurement units.

The variables, that are provided in EPANET input file for the package, and their corresponding units in US customary and SI units are shown in Table 6. As with the input variables, the system equations were modified to use dimensionless variables. Once the stopping test is satisfied, the original variables are then recovered by reversing the initial scaling. Details of the scaling are shown in Appendix A.

To help ensure that WDSLib solution methods are both fast and reliable. The sparse matrix operations are implemented using SuiteSparse (Davis et al. 2014). SuiteSparse is a state-of-the-art sparse arithmetic suite with exceptional performance, from which the approximate minimum degree permutation (AMD) and the sparse Cholesky decomposition routines have been used.

Design Considerations 4: Timing Considerations

When executing WDSLib, each function reports the time spent in it by sampling wall clock time at the start and end of its execution. Although the overhead for sampling

wall clock time is small, there are at least two special considerations involved in the interpretation of these timings: (i) the operating system, at its own discretion, may launch background processes (for example anti-virus software), which will distort the timings and (ii) extrapolating the timing for multiple hydraulic simulations from a single analysis (as may be required, for example, in a genetic algorithm or other evolutionary algorithm run) must be done with care because the relationship between the different settings is not linear.

This concludes the discussion of the main considerations concerning the global design of WDSLib. In the following, key details of the implementation of selected parts of the solution processes are described.

5.2 Key Improvements to Solution Processes

The WDSLib implementation makes several improvements to extant solution processes. This section focuses on the improvement of the network partitioning processes in FCPA and RCTM.

Key Optimization 1: Improvements to partitioning in Forest Search

The forest-core partitioning algorithm (FCPA) in this paper is a substantial improvement over the algorithm of the original paper (Simpson et al. 2014). Specifically the original FCPA algorithm almost always required many sweeps of the columns (nodes) of the $\boldsymbol{A_1}$ matrix in order to reduce the forest component down to the core component. The refined algorithm exploits the adjacency list representation of the $\boldsymbol{A_1}$ matrix so that the partitioning process is achieved in a single sweep. This improves the speed of the partitioning process from being $O(an_p)$ to $O(n_p + n_f)$ where a is the depth of the deepest tree-component in the forest. This can lead to substantial time savings in the case when a is relatively large.

The pseudo-code for this refined forest search algorithm is shown in Appendix B This algorithm traverses each tree component in turn from its leaf nodes, which maximizes the locality of operations with respect to the graph representation. In this algorithm, a node

is identified as a leaf node when its node degree is one. Every time a leaf node, node k, is identified, the node pointer is moved to its adjacent node, node k, and the node degree of node k is reduced by one. This process repeats if the adjusted node degree of node k is one. Otherwise, node k is the root node for this tree and the algorithm progresses to the next tree in the forest.

Key Optimization 2: Improvements to Spanning Tree Search

The reformulated co-tree flows method (RCTM) in this paper is also a substantial improvement over the algorithm of the original paper (Elhay et al. 2014). The original spanning tree search algorithm sweeps the rows of the $\boldsymbol{A_1}$ matrix (pipes) in order to identify the singleton rows and their corresponding columns. The spanning tree search in the original RCTM required a sweep of of the $\boldsymbol{A_1}$ matrix to identify the next pipe in the spanning tree. This algorithm is $O(n_p n_j)$, which is relatively inefficient.

The pseudo-code for the refined spanning tree search algorithm is shown in Appendix C This improved algorithm takes as input the adjacency list describing the network and the pipe indexes of the core component of the network from the Algorithm 1 (if the FCPA is used). In this algorithm, all water sources are the starting point of the search process, SN, and marked as *visited*. The nodes in SN are then used as to identify a spanning tree within the WDS. This is achieved by repeatedly finding all adjacent pairs, node t and pipe s, of and removing the first node in SN by using the adjacency list. If the adjacent node t is not visited then node t is inserted into the spanning-tree node vector, STN, and search node vector, SN, and node t is marked as visited and pipe s to the spanning-tree pipe vector, STP, and pipe s is marked as visited. If the adjacent node t is visited and the pipe s is not visited then the pipe s is inserted into the co-tree pipe vector, CTP and mark pipe s as visited. This process is repeated until SN is empty. The overall time-complexity of this algorithm is $O(n_p + n_j)$ (compared to O($n_p n_j$) as mentioned above) is the same as the best asymptotic complexity of breadth-first search on a graph.

6 Example Applications

WDSLib consists of a collection of functions which can be used either as a standalone application for fast one-off simulations or as a library of software components that can be integrated into a user's own WDS solution processes. This section presents two example applications. The first application is the setup for a basic one-off simulation of a WDS. The second application (described in subsection 6.1) presents an example using WDSLib to implement a simple 1+1 Evolutionary Strategy (Beyer and Schwefel 2002) (1+1-ES or, more commonly, 1+1EA) for sizing pipes in a WDS.

Example 1 - Once-off Simulation

The setup for WDSLib as a standalone application is straightforward. The user provides a configuration text file that specifies input and output filenames; the name of the solver; the desired output variables; and simulation parameters. These values have sensible defaults so the user can set up the solver by using a minimal configuration such as that shown in Fig. 4. By using this config file, WDSLib is configured to run a single hydraulic analysis of the network that is stored as say "hanoi.inp", an EPANET- formatted input file, under "Network/" sub-directory, using the reformulated co-tree flows method with the forest-core partitioning algorithm. The full set of configuration parameters for once

```
[InputFile]
<directory>      %the input file directory
Network/
<file>           %the input file name
hanoi.inp
<end>
%-------------------------------------------------%
[controlFlag]
<SolverFlag>     %1 for GGA 2 for RCTM
2
<FCPAFlag>
1
<end>
%-------------------------------------------------%
```

Figure 4: A minimal configuration file to run the GGA in WDSLib

off simulations is shown in Fig. 10 in Appendix D.

6.1 Example 2 - A Simple Network Design Application

As a minimalist example of the application of WDSLib to a WDS network design problem, the following example uses 1+1EA for optimally sizing pipe diameters. This algorithm takes an existing network with randomly generated pipe diameters and optimizes the network to minimize cost, subject to given pressure head constraints. A 1+1EA is a very simple evolutionary strategy (Beyer and Schwefel 2002) which starts with a randomly generated individual (in this case a WDS diameter configuration). This 1+1EA then progresses by applying a mutation to a random pipe diameter size, and then evaluating the new individual. If the new individual is better it replaces the old network. This process continues in a loop until a given number of evaluations is reached.

The C++ code for this example is shown in Figs. 5, 6, 7, and 8. If the name of the file containing this code is: `simpEA.cc` then the simplest command to compile this code is:

```
g++ simpEA.cc -o simpEA -Llib -lWDSLib
```

To run this code the user would type:

```
./simpEA config.txt
```

where `config.txt` contains the same configuration text as for the previous example.

Starting with the `main` function in Fig. 5, line 15 points to the config file specified by the command line. The next two lines initialize the `result` and the `simulation` according to the configuration file. This is followed by the L_{1a} module to perform the user selected L_{1a} functions. Line 19 generates the initial pipe diameters of the network and line 20 initializes the workspace for the mutated string. Line 23 sets the pipe diameters of the network. Line 24 evaluates the current network configuration. The permutation and scaling for the current individual is reversed by L_{1b} in line 25 of Fig. 5. Line 26 calculates the fitness of the current network configuration by using the `evaluate` function in Fig. 8. This function applies a penalty for pressure head constraint violations and pipe material costs. The body of the 1+1EA is contained in the selection operator and mutation operator that follow. Lines 27 to 31 compare the string in the current generation with

```
#include "Simulation.h"
#include "result.h"
using namespace std;
#define GTN 100000                                  //the total number of
    generations
/*available pipe diameters (inches)*/
vector<int> ADiameter
    ={36,48,60,72,84,96,108,120,132,144,156,168,180,192,204};
/*dollar per unit length*/
vector<int>unitCost
    ={93.6,133.7,176.3,221,267.6,315.8,365.4,416.5,468.7,
                522.1,576.6,632.1,688.5,745.1,804.1};
vector<int>  generateinitialDiameters(int);      // see fig. 6
vector<int>  mutate(vector<int>);                // see fig. 7
double evaluate(Net*, Result*);                  // see fig. 8
int  main(int argc, char *argv[]){
  srand (1);
  char *config=argv[1];
  Result *result=new Result();
  Simulation *simulation1=new Simulation();      //initialize the
      simulation class
  Net *net=simulation1->L1a(result,config);      //perform the L1a
      functions
  vector<int> p1=generateinitialDiameter(net->getNp());//initial
      guesses of diameter
  vector<int> p2;                                //work space for
      current best
  double cost,currentbest;
  for(int i=0;i<GTN;i++){
    simulation1->setD(&p1,&ADiameter,net->getPIPE()->Diascale());
    simulation1->solve(result);                  //perform the L2 and L3
        functions
    simulation1->L1b(result);                    //reverse the
        permutation
    cost=evaluate(result,net,&p1);               //evaluate the network
        cost
    if(cost<currentbest||i==0){                  //selection operator
      currentbest=cost;
      p2=p1;
      cout<<i<<"\t"<<currentbest<<"\t"<<penaltyCost<<endl;
    }
    p1=mutate(&p2);                              //mutation operator
  }
  simulation1->dispResult(result);
  delete simulation1;
  delete result;
  return 0;
 }
```

Figure 5: Example code for 1+1EA for Pipe Sizing

```
vector<int> generateinitialDiameters(int np)
{
  vector<int> Diameter = vector<int>(np);
  for (int i=0;i<np;i++)
    Diameter[i]=rand()%ADiameter.size();  //set the index for pipe
        diameters
  return Diameter;
}
```

Figure 6: Implementation code for pipe size initialization

```
vector<int>  mutate(vector<int>* string){
  vector<int> string1;
  string1=*string;
  int aa=rand()%(string->size()); //choose which pipe to mutate
  int a=rand()%ADiameter.size();  //choose a pipe diameter after the
      mutation
  (string1)[aa]=a;                //set the pipe index
  return string1;
}
```

Figure 7: Implementation code for the mutation operator

```
double evaluate (Result* result,Net* net,vector<int> *p1) {
  PIPE* pipe =net->getPIPE();
  double np=net->getNp();
  double nj=net->getNj();
  double P=1e7;
  double cost1=0;
  penaltyCost=0;
  vector<double> hsol=result->getHsol();       //get the vector of nodal
      heads
  vector<double>* L=pipe->getL();              //get the vector of pipe
      lengths
  vector<double>* zu=net->getNODE()->getZU(); //get the vector of nodal
      elevations
  double Lscale=pipe->Lenscale();              //get the scaling factor
      for length
  for (int i = 0; i<np; i++)
    cost1+=(*L)[i]*Lscale*unitCost[(*p1)[i]]; //calculate the material
        cost
  for (int i = 0; i<nj; i++)
    if ((hsol)[i]-(*zu)[i]<minP)
      penaltyCost+=(minP-(hsol)[i]+(*zu)[i])*scaleP;//calculate penalty
          cost
  return cost1+penaltyCost;
}
```

Figure 8: Implementation code for the evaluation function

the current best string if the individual `p1`, as measured by `evaluate` is better than the individual `p2` then `p1` replaces `p2`. Line 32 mutates the current network, `p2`, using `mutate` (see Fig. 7). The `mutate` function changes the diameter of a randomly selected pipe in the network to a randomly selected diameter, chosen from a set of commercially available pipe diameters. The mutated individual, stored in the workplace `p1`, is used as the network configuration for the next iteration. Until the total number of generations is reached, the user selected information about the best individual is outputted by `dispResult` in line 34 of Fig. 5.

It should be noted that the algorithm described above can be used to design a simple WDS but is not optimal in terms of speed of convergence. Other EA's such as genetic algorithms (Simpson et al. 1994) will perform better. However the above example has the advantage of simplicity and contains all the basic elements that a GA would use when interacting with WDSLib.

This concludes the presentation of examples in this work. The next section presents a case study that illustrates the performance of WDSLib in a multi-simulation setting.

7 Case Study

The following presents timing results for WDSLib running the 1+1EA described in the previous section. The results below compare the four different solvers plus EPANET2. Note, that detailed timings for once-off simulations comparing the four methods can be found in Qiu et al. (2018). Three networks were benchmarked in these experiments. These were the N_1, N_3, and N_4 case-study networks used in Simpson et al. (2014). Table 7 summarizes the characteristics of these networks.

Table 7: Benchmark networks summary

	Full Network			Forest & Core Networks			Co-tree Network
Network	n_p	n_j	n_s	$n_f(n_f/n_p^{\#})$	n_{j_c}	n_{p_c}	n_{ct}
N_1	934	848	8	361 (38%)	573	487	84
N_3	1975	1770	4	823 (42%)	1152	947	205
N_4	2465	1890	3	429 (17%)	2036	1461	757

627 Table 8 shows the results of the 1+1EA from Fig. 5 for the GGA, GGA with FCPA,
628 RCTM, RCTM with FCPA and the EPANET2 solvers. For each of the four WDSLib
629 solvers above, the timings are given for running the EA with and without the L1 modules
630 hoisted out the main EA loop. Each experiment evaluates the WDS network 100,000
631 times. And the best performing method for each network is highlighted in bold. It is
632 important to note that 1+1EA using both the GGA and the WDSLib

Table 8: The actual 1+1EA run-time with 100,000 evaluations (min.) for each of the four solution methods applied networks N_1, N_3, N_4

	GGA	GGA with FCPA	RCTM	RCTM with FCPA	EPANET
	min.	min.	min.	min.	min.
N_1	6.73	4.64	4.53	**4.13**	9.81
N_3	15.21	**9.79**	13.75	10.30	26.43
N_4	21.14	**16.29**	23.92	21.93	67.11

633 The results show that the EA runs using WDSLib are substantially faster than the
634 runs using the EPANET2 solver. This is, in part, due to the fact that the EPANET2
635 solver is designed as a standalone solver which does not facilitate lifting out of invariant
636 computations from the EA loop.

637 As a demonstration of how the performance of an EA can be traced Fig. 9 shows the
638 evolution of the fitness values of the N_1 network. These traces were extracted from a file
639 written to in line 30 in Fig. 5. As can be seen, the cost and the pressure head violation
640 terms drop during the EA run. Note that there will be considerable variation between
641 1+1EA runs due to its highly stochastic nature.

642 8 Conclusions

643 This paper has described WDSLib, a library for steady-state hydraulic simulation of WDS
644 networks. WDSLib is fast, modular, and portable with implementation of several stan-
645 dard and recently published hydraulic solution methods. We have outlined the supported
646 solution methodologies, the structure of the package and key aspects of WDSLib's imple-
647 mentation. Two example applications have been presented including a design case study

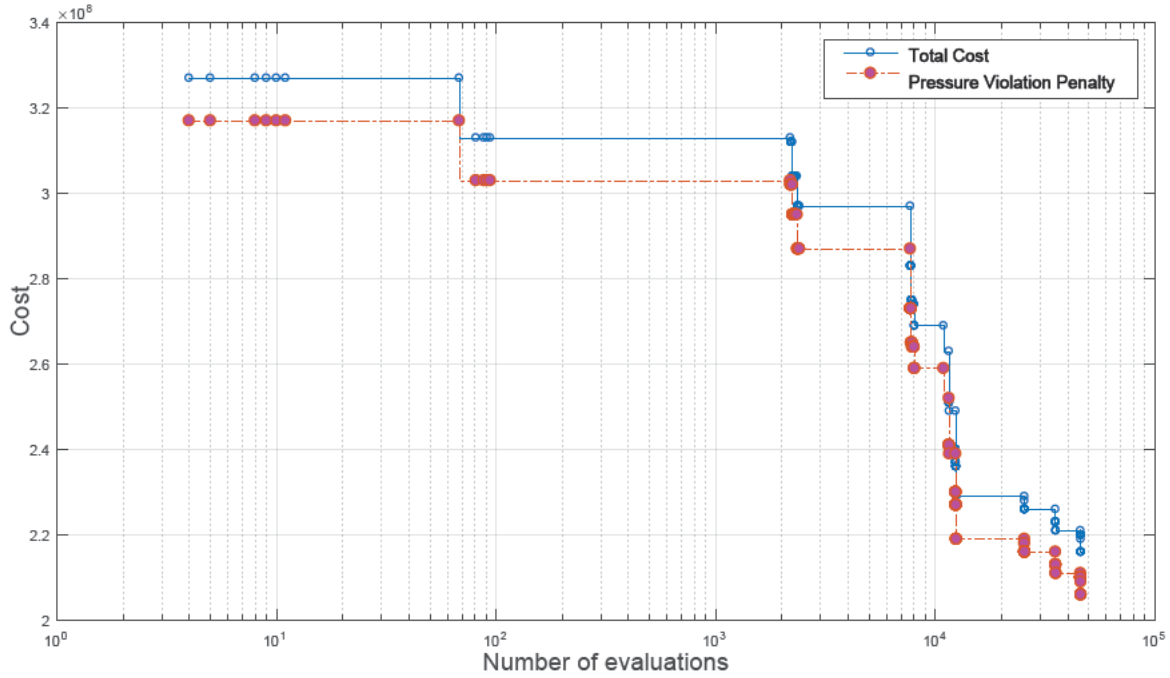


Figure 9: The evaluation of the fitness value of network N_1

using a simple EA. The EA results were benchmarked for different solvers demonstrating a substantial improvement in speed over the industry standard EPANET2 package. These benchmarks also have illustrated how the modular structure of WDSLib could be exploited to speed up execution time in a design setting.

As well as providing a fast simulation platform for both once-off and multi-run simulations, WDSLib also provides a testbed for comparing different solution methods in different settings and network topologies. As such WDSLib is designed with a pluggable architecture which can extended to efficiently incorporate new solution methods as they are created. This will enhance the capability of the research community to demonstrate the efficacy of new methods without having to re-engineer the content of shared WDSLib functions and data representations.

WDSLib accepts the input file format used by EPANET2 which allows for simple deployment. The amount of coding required to use the provided solvers in WDSLib is minimized by the use of a simple and extensible configuration file.

WDSLib is currently limited to finding solutions of demand-driven WDS systems without control devices. The development of software components to simulate control

devices and pressure driven models is future work.

References

Abraham, E. and I. Stoianov (2015). "Sparse null space algorithms for hydraulic analysis of large-scale water supply networks". *Journal of Hydraulic Engineering*, p. 04015058. DOI: 10.1061/(ASCE)HY.1943-7900.0001089.

Benzi, M., G. H. Golub, and J. Liesen (2005). "Numerical solution of saddle point problems". *Acta Numerica* 14, pp. 1–137. DOI: 10.1017/S0962492904000212.

Beyer, H. G. and H. P. Schwefel (2002). "Evolution strategies–A comprehensive introduction". *Natural Computing* 1.(1), pp. 3–52. DOI: 10.1023/A:1015059928466.

Cheung, P. B., J. E. Van Zyl, and L. F. R. Reis (2005). "Extension of EPANET for pressure driven demand modeling in water distribution system". *Computing and Control for the Water Industry* 1, pp. 311–316.

Cross, H. (1936). "Analysis of flow in networks of conduits or conductors". *University of Illinois. Engineering Experiment Station. Bulletin; no. 286*.

Davis, Tim, WW Hager, and IS Duff (2014). "SuiteSparse". *http://faculty.cse.tamu.edu/davis/suitespars*

Deuerlein, J. W. (2008). "Decomposition model of a general water supply network graph". *Journal of Hydraulic Engineering* 134.(6), pp. 822–832. DOI: 10.1061/(ASCE)0733-9429(2008)134:6(822).

Deuerlein, J. W., S. Elhay, and A. R. Simpson (2015). "Fast Graph Matrix Partitioning Algorithm for Solving the Water Distribution System Equations". *Journal of Water Resources Planning and Management* 142.(1), pp. 04015037-1–04015037-11.

Diao, Kegong, Zhengji Wang, Gregor Burger, Chien-Hsun Chen, Wolfgang Rauch, and Yuwen Zhou (2014). "Speedup of water distribution simulation by domain decomposition". *Environmental Modelling & Software* 52, pp. 253–263. ISSN: 1364-8152. DOI: https://doi.org/10.1016/j.envsoft.2013.09.025. URL: http://www.sciencedirect.com/science/article/pii/S1364815213002247.

690 Duff, I. S., R. G. Grimes, and J. G. Lewis (1989). "Sparse matrix test problems". *ACM*
691 *Transactions on Mathematical Software (TOMS)* 15.(1), pp. 1–14.
692 Elhay, S. and A. R. Simpson (2011). "Dealing with zero flows in solving the nonlinear
693 equations for water distribution systems". *Journal of Hydraulic Engineering* 137.(10),
694 pp. 1216–1224. DOI: 10.1061/(ASCE)WR.1943-5452.0000561.
695 Elhay, S., A. R. Simpson, J. Deuerlein, B. Alexander, and W. H. Schilders (2014). "Refor-
696 mulated co-tree flows method competitive with the global gradient algorithm for solv-
697 ing water distribution system equations". *Journal of Water Resources Planning and*
698 *Management* 140.(12), pp. 04014040-1–04014040-10. DOI: 10.1061/(ASCE)WR.1943-
699 5452.0000431.
700 Epp, R. and A. G. Fowler (1970). "Efficient code for steady-state flows in networks".
701 *Journal of the Hydraulics Division* 96.(1), pp. 43–56.
702 Gorev, N. B., I. F. Kodzhespirov, Y. Kovalenko, E. Prokhorov, and G. Trapaga (2012).
703 "Method to Cope with Zero Flows in Newton Solvers for Water Distribution Systems".
704 *Journal of hydraulic engineering* 139.(4), pp. 456–459. DOI: 10.1061/(ASCE)HY.1943-
705 7900.0000694.
706 Guidolin, M., P. Burovskiy, Z. Kapelan, and D. A. Savic (2010). "CWSNet: An object-
707 oriented toolkit for water distribution system simulations". *Proceedings of the 12th*
708 *Annual Water Distribution Systems Analysis Conference, WDSA*, pp. 12–15.
709 Jun, L. and Y. Guoping (2012). "Iterative methodology of pressure-dependent demand
710 based on EPANET for pressure-deficient water distribution analysis". *Journal of Wa-*
711 *ter Resources Planning and Management* 139.(1), pp. 34–44. DOI: 10.1061/(ASCE)
712 WR.1943-5452.0000227.
713 Morley, M. S., R. M. Atkinson, D. A. Savic, and G. A. Walters (2000). "OpenNet: An
714 applicationindependent framework for hydraulic network representation, manipula-
715 tion, and dissemination". *Hydroinformatics 2000 Conference, University of Iowa, Iowa*
716 *City, USA*, pp. 23–27.
717 Morley, M. S. and C. Tricarico (2008). *Pressure-Driven Demand Extension for EPANET*
718 *(EPANETpdd)*. Tech. rep. University of Exeter.

Nethercote, N. and J. Seward (2007). "Valgrind: a framework for heavyweight dynamic binary instrumentation". *ACM Sigplan notices*. Vol. 42. 6. ACM, pp. 89–100. DOI: 10.1145/1273442.1250746.

Perelman, Lina and Avi Ostfeld (2011). "Topological clustering for water distribution systems analysis". *Environmental Modelling & Software* 26.(7), pp. 969–972. ISSN: 1364-8152. DOI: https://doi.org/10.1016/j.envsoft.2011.01.006. URL: http://www.sciencedirect.com/science/article/pii/S1364815211000259.

Qiu, M., S. Elhay, A. R. Simpson, and B. Alexander (2018). "A Benchmarking Study of Water Distribution System Solution Methods". Manuscript submitted for publication to Journal of Water Resources Planning and Management.

Rahal, H. (1995). "A co-tree flows formulation for steady state in water distribution networks". *Advances in Engineering Software* 22.(3), pp. 169–178. DOI: 10.1016/0965-9978(95)00020-W.

Rossman, L. A. (2000). "Epanet 2 users manual, us environmental protection agency". *Water Supply and Water Resources Division, National Risk Management Research Laboratory, Cincinnati, OH* 45268.

Schilders, W. H. (2009). "Solution of indefinite linear systems using an LQ decomposition for the linear constraints". *Linear Algebra and its Applications* 431.(3), pp. 381–395. DOI: 10.1016/j.laa.2009.02.036.

Siew, C. and T. T. Tanyimboh (2012). "Pressure-dependent EPANET extension". *Water resources management* 26.(6), pp. 1477–1498. DOI: 10.1061/41203(425)9.

Simpson, A. R., G. C. Dandy, and L. J. Murphy (1994). "Genetic algorithms compared to other techniques for pipe optimization". *Journal of water resources planning and management* 120.(4), pp. 423–443. DOI: 10.1061/(ASCE)0733-9496(1994)120:4(423).

Simpson, A. R. and S. Elhay (2010). "Jacobian matrix for solving water distribution system equations with the Darcy-Weisbach head-loss model". *Journal of hydraulic engineering* 137.(6), pp. 696–700. DOI: 10.1061/(ASCE)HY.1943-7900.0000341.

747 Simpson, A. R., S. Elhay, and B. Alexander (2014). "Forest-Core Partitioning Algorithm
748 for Speeding Up Analysis of Water Distribution Systems". *Journal of Water Resources*
749 *Planning and Management* 140.(4), pp. 435–443. DOI: 10.1061/(ASCE)WR.1943-
750 5452.0000336.

751 Todini, E. and S. Pilati (1988). "A gradient algorithm for the analysis of pipe networks".
752 *Computer applications in water supply: vol. 1—systems analysis and simulation*. Re-
753 search Studies Press Ltd., pp. 1–20.

754 Vassiljev, A. and T. Koppel (2015). "Estimation of real-time demands on the basis of
755 pressure measurements by different optimization methods". *Advances in Engineering*
756 *Software* 80, pp. 67–71. DOI: 10.1016/j.advengsoft.2014.09.023.

757 van Zyl, J. E., J. Borthwick, and A. Hardy (2003). "Ooten: An object-oriented program-
758 mers toolkit for EPANET". *Advances in Water Supply Management (CCWI 2003)*.

Appendix A Scaling

In WDSlib, all input variables are scaled to a similar range to minimize the risk of avoidable computational inaccuracy or failures in floating point arithmetic.

The variables are scaled as following: $\hat{\boldsymbol{G}} = \boldsymbol{G}/G_0$, $\hat{\boldsymbol{q}} = \boldsymbol{q}/q_0$, $\hat{\boldsymbol{h}} = \boldsymbol{h}/h_0$, $\hat{\boldsymbol{e_l}} = \boldsymbol{e_l}/e_{l0}$, $\hat{\boldsymbol{d}} = \boldsymbol{d}/d_0$ where $\boldsymbol{G}$, $\boldsymbol{h}$, $\boldsymbol{e_l}$, and $\boldsymbol{d}$ are the original input vectors, G_0, h_0, e_{l0}, and d_0 are the scaling factors and $\hat{\boldsymbol{G}}$, $\hat{\boldsymbol{h}}$, $\hat{\boldsymbol{e_l}}$, and $\hat{\boldsymbol{d}}$ are the scaled input vectors.

By substituting $\boldsymbol{G} = \hat{\boldsymbol{G}}G_0$, $\boldsymbol{q} = \hat{\boldsymbol{q}}q_0$, $\boldsymbol{h} = \hat{\boldsymbol{h}}h_0$, $\boldsymbol{e_l} = \hat{\boldsymbol{e_l}}e_{l0}$, $\boldsymbol{d} = \hat{\boldsymbol{d}}d_0$, Eq. (1) and Eq. (2) become:

$$G_0 q_0 \hat{\boldsymbol{G}}\hat{\boldsymbol{q}} - h_0 \boldsymbol{A_1}\hat{\boldsymbol{h}} - e_{l0}\boldsymbol{A_2}\hat{\boldsymbol{e_l}} = 0 \tag{17}$$

$$q_0 \boldsymbol{A_1}^T \hat{\boldsymbol{q}} - d_0 \hat{\boldsymbol{d}} = 0 \tag{18}$$

Eq. (17) and Eq. (18) can be further simplified by introducing the notation: $a = h_0/\left(G_0 * q_0\right)$, $b = e_{l0}/\left(G_0 * q_0\right)$, and $c = d_0/q_0$:

$$\hat{\boldsymbol{G}}\hat{\boldsymbol{q}} - a\boldsymbol{A_1}\hat{\boldsymbol{h}} - b\boldsymbol{A_2}\hat{\boldsymbol{e_l}} = 0 \tag{19}$$

$$\boldsymbol{A_1}^T \hat{\boldsymbol{q}} - c\hat{\boldsymbol{d}} = 0 \tag{20}$$

with the following matrix form:

$$\begin{pmatrix} \hat{\boldsymbol{G}} & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \hat{\boldsymbol{q}} \\ a\hat{\boldsymbol{h}} \end{pmatrix} - \begin{pmatrix} b\boldsymbol{A_2}\hat{\boldsymbol{e}}_l \\ c\hat{\boldsymbol{d}} \end{pmatrix} = 0. \tag{21}$$

Finally, the network can be solved by using Eq. (22) and Eq. (23):

$$\hat{\boldsymbol{h}}^{(m+1)} = \frac{1}{a}\boldsymbol{U}^{-1}[-c\hat{\boldsymbol{d}} - \boldsymbol{A_1}\hat{\boldsymbol{F}}^{-1}[(\hat{\boldsymbol{F}} - \hat{\boldsymbol{G}})\hat{\boldsymbol{q}}^{(m)} + b\boldsymbol{A_2}\hat{\boldsymbol{e}}_l]] \tag{22}$$

$$\hat{\boldsymbol{q}}^{(m+1)} = \hat{\boldsymbol{q}}^{(m)} + a\hat{\boldsymbol{F}}^{-1}\boldsymbol{A_1}\hat{\boldsymbol{h}}^{m+1} - \hat{\boldsymbol{F}}^{-1}(\hat{\boldsymbol{G}}\hat{\boldsymbol{q}}^{(m)} - b\boldsymbol{A_2}\hat{\boldsymbol{e}}_l) \tag{23}$$

Choice of scaling factors

The choice of the scaling factor, despite much research, is not well understood. In this subsection, a choice for each scaling factor, based on the experience of the authors, is recommended. There are two types of variables and parameters that need to be scaled: invariants and variants.

Data sets that have very wide range of values can confound numerical accuracy. As a result, it may be preferable to scale the data to a narrower range. The default scaling factor for each of the input data is chosen to be its maximum absolute value. For example, the scaling factor for demand is $\max(\boldsymbol{d})$, so that its values range from zero to one.

In contrast, it is more difficult to choose a scaling factor a priori for values that vary between iterations (variants). This is because the range of variants can change as the iteration progresses. As a result, the intermediate and the final results might not be within the same range as the initial guesses.

There are two variants that need to be scaled: $\boldsymbol{q}$, $\boldsymbol{h}$. A good choice of the scaling factor for the flow rate is $\frac{\sum \boldsymbol{d}}{n_f}$ because the demand at each node must be satisfied by the water sources in the WDS and it is a reasonable assumption that the all demands are equally carried by each pipe that is directly connected to a water source and a good choice of the scaling factor for nodal head is $max(\boldsymbol{e_l})$ because the maximum nodal head cannot exceed the maximum elevation head of the fixed nodes.

During the process of the computation, the matrices $\boldsymbol{G}$ and $\boldsymbol{F}$ are scaled because their input variables are scaled. For the Darcy-Weisbach head loss model, the diagonal elements of the matrix $\boldsymbol{G}$ are modelled by:

$$[G]_{jj} = diag\left\{(\frac{8}{\pi^2 g})\frac{L_j}{D_j^5} f_j |q_j|\right\} \text{ for } j = 1, \ldots, n_p$$

where the friction factor f is modelled by the Swamee-Jain formula:

$$f_j = \begin{cases} \dfrac{64}{\mathbb{R}_j} & \text{if } \mathbb{R}_j \leq 2000 \\ \sum_{k=0}^{3}(\alpha_k + \beta_k/\theta_j)(\mathbb{R}_j/2000)^k & \text{if } 2000 < \mathbb{R}_i < 4000 \\ \dfrac{0.25}{\log^2(\theta_j)} & \text{if } \mathbb{R}_j \geq 4000 \end{cases} \tag{24}$$

where $\theta_j = \dfrac{\epsilon_j}{3.7D_j} + \dfrac{5.74}{\mathbb{R}_j^{0.9}}$ and $\mathbb{R}_j = \dfrac{4q_j}{\pi \upsilon D_j}$ for $j = 1, \ldots, n_p$. Note that α_k and β_k are constant, the values of which can be found in Elhay and Simpson (2011)

In order to make sure the Reynolds number, a dimensionless variable, is not affected by scaling, $\hat{\boldsymbol{\nu}} = \dfrac{D_0}{q_0}\boldsymbol{\nu}$ is introduced, Reynolds number $\mathbb{R}_j$ becomes $\dfrac{4\hat{q}_j q_0}{\pi\hat{\nu}(q_0/D_0)\hat{D}_j D_0}$, which can be further simplified to $\dfrac{4\hat{q}_j}{\pi\hat{\nu}\hat{D}_j}$, where the scaling factors are canceled.

In order to make sure f is also not affected by the scaling, $\hat{\boldsymbol{\epsilon}} = D_0\boldsymbol{\epsilon}$ is introduced, θ_j becomes

$$\theta_j = \frac{\hat{\epsilon}_j D_0}{3.7\hat{D}_j D_0} + \frac{5.74}{\mathbb{R}_j^{0.9}} \text{ for } j = 1, \ldots, n_p$$

which can be further simplified to

$$\theta_j = \frac{\hat{\epsilon}_j}{3.7\hat{D}_j} + \frac{5.74}{\mathbb{R}_j^{0.9}} \text{ for } j = 1, \ldots, n_p$$

where the scaling factors are canceled. It is evident that the friction factors remain the same because the values for the only two variables $\mathbb{R}$ and θ are unchanged.

Finally, diagonal elements of $\boldsymbol{G}$ can be expressed as: $G_0\hat{\boldsymbol{G}}$, where $G_0 = (\dfrac{8}{\pi^2 g})\dfrac{L_0}{D_0^5}|q_0|$ and $[\hat{G}]_{jj} = diag\left\{\dfrac{\hat{L}_j}{\hat{D}_j^5} f_j|\hat{q}_j|\right\}$ for $j = 1, \ldots, n_p$.

For the Hazen-Williams head loss model, the diagonal elements of the matrix $\boldsymbol{G}$ are modelled by: $[G]_{jj} = diag\left\{\dfrac{10.67L_j}{C_j^{1.852}D_j^{4.871}}|q_j|^{(n-1)}\right\}$ for $j = 1, \ldots, n_p$, where C_j is the Hazen-Williams coefficient for the j-th pipe. The Hazen-Williams coefficient, unlike the friction factor in the Darcy-Weisbach head loss model, is independent of flow rate, pipe wall condition, and flow regimes, which means it is an independent variable. As a result, the scaling factor for Hazen-Williams $\boldsymbol{G}$ can simply be derived by: $[G]_{jj} =$

$diag\left\{\frac{10.67\hat{L}_j L_0}{\hat{C}_j^{1.852} C_0^{1.852} \hat{D}_j^{4.871} D_0^{4.871}} |\hat{q}_j|^{(n-1)} |q_0|^{(n-1)}\right\}$ and the equation for diagonal elements of $\boldsymbol{G}$ for Hazen-Williams equation can be expressed as: $\boldsymbol{G} = diag\left\{G_0 \hat{\boldsymbol{G}}\right\}$, where $G_0 = \frac{10.67 L_0}{C_0^{1.852} D_0^{4.871}} |q_0|^{(n-1)}$ and $[G]_{jj} = diag\left\{\frac{\hat{L}_j}{\hat{C}_j^{1.852} \hat{D}_j^{4.871}} |\hat{q}_j|^{(n-1)}\right\}$ for $j = 1, \ldots, n_p$.

Appendix B Forest Search Algorithm

Algorithm 1: Forest Search Algorithm

input : Adjacency list, $\boldsymbol{d}$ and Deg
output: Forest, RootNode, $\boldsymbol{q}$ and $\boldsymbol{d}$

$k \leftarrow 1$;
for $i \leftarrow 1$ **to** n_j **do**
  $n = i$;
  while $Deg(n) == 1$ **do**
    for $(Adj_p, Adj_v) \in Adj(n)$ **do**
      if $Deg(Adj_v) == 1$ **then**
        $Forest[k] \leftarrow$`Union(`$Forest[k]$, (Adj_p, n)`)`
      else
        $\boldsymbol{q}(Adj_p) \leftarrow \boldsymbol{d}(n)$;
        $\boldsymbol{d}(Adj_v) \leftarrow \boldsymbol{d}(Adj_v) + \boldsymbol{d}(n)$;
        $Deg(Adj_v) \leftarrow Deg(Adj_v) - 1$;
        $n = Adj_v$;
        if $Deg[n] \geq 2$ **then**
          $RootNode[k] = n$;
          $k \leftarrow k + 1$;
        end if
      end if
    end for
  end while
end for

Appendix C Spanning Tree Search Algorithm

Algorithm 2: Spanning Tree Search Algorithm

input : adjList
output: Spanning Tree and Co-Tree elements
$STP \leftarrow \{\}$; `// initialize an empty vector for spanning tree pipes`
$STN \leftarrow \{\}$; `// initialize an empty vector for spanning tree nodes`
$CTP \leftarrow \{\}$; `// initialize an empty vector for co-tree pipes`
$VN \leftarrow \{\}$; `// initialize a boolean vector for visited nodes`
$VP \leftarrow \{\}$; `// initialize a boolean vector for visited pipes`
$SN \leftarrow \{\}$; `// initialize an empty set of searching nodes`
for $i \leftarrow n_j$ **to** n_f **do**
  $VN[i] = true$; `// mark the source i as visited`
  $SN \leftarrow i$; `// insert this node into the search node vector`
end for
while *SN is not empty* **do**
  $i \leftarrow$ the first element in SN;
  foreach *(*Adj_p*,*Adj_v*)* $\in Adj[i]$ **do**
    if *VN[*Adj_v*] == false* **then**
      $STP \leftarrow Adj_p$; `// mark pipe as a spanning tree pipe`
      $STN \leftarrow Adj_v$; `// mark node as a corresponding spanning tree pipe`
      $SN \leftarrow Adj_v$;
      VN[Adj_v]=true;
      VP=[Adj_p]=true;
    end if
    else if *VN[*Adj_p*]==false* **then**
      $CTP \leftarrow Adj_p$; `// removed pipe` j `as a singleton pipe`
      VP[Adj_p]=true;
    end if
  end foreach
  remove i from SN;
end while

Appendix D Complete configuration files

```
[InputFile]
<directory>      %REQUIRED INFORMATION the input file directory
WDSnetwork/
<file>           %REQUIRED INFORMATION the input file name
n1.inp
<end>            %end of input file
%-------------------------------------------------%
[Parameter]
<maxIter>        %maximum number of iteration
50
<initV>          %initial veloc  ity
0.3048         %1ft/s
<StopTol>        %stopping tolerance used
1.000e-6
<NormTyp>      %norm type 1 for 1-norm, 2 for 2-norm, 3 for inf-norm
3.0
<StopTest>     %stopping test used 1 for q-norm, 2 for h-norm 3 for q&h-
    norm
1
<SerP>           %service pressure
20.0
<MinP>           %Minimum pressure
40.0
<Demandfuc>      %type of consumption function
1.00
<MaxNpIterResult> %np treshold for disping iterates result
50.0
<MinImproTol>
1.0000e-3
<end>      %end of parameter
%-------------------------------------------------%
[dispFlag]
<BasicFlag>
false
<NetInfoFlag>
false
<ConvergenceFlag>
false
<StatFlag>
false
<ScalingFlag>
0
<NodalResultflag>
0
<LinkResultflag>
false
<QitersFlag>
false
<HitersFlag>
false
<timeFlag>
false
<end>            %end of dispFlag
%-------------------------------------------------%
[controlFlag]
<SolverFlag>     %1 for GGA 2 for RCTM 3 for SMPA 4 for PDM
1
<FCPAFlag>
0
<end>      %end of controlFlag
%-------------------------------------------------%
```

Figure 10: A configuration file to run the RCTM in WDSLib

Appendix E Nomenclature

Acronyms

CT Co-tree

CTP Co-tree pipes

DW Darcy-Weisbach head loss formula

FCPA Forest-core partitioning algorithm

GGA Global gradient algorithm

HW Hazen-William head loss formula

WDS Water distribution system

RCTM Reformulated co-tree flows method

nnz Number of non-zeros

STP Spanning tree pipes

STN Spanning tree nodes

ST Spanning tree

Constants

g Gravitational acceleration constant

ν Kinematic viscosity of water

FCPA variables

$\boldsymbol{C}$ Permutation matrix for the nodes in the core, E_c, of the network

E_c Set of core pipes (edges) in G_c

E_f Set of forest pipes (edges) in G_f

851 G_c Core subgraph

852 G_f Forest subgraph

853 $\boldsymbol{P}$ Permutation matrix for the pipes in the core, E_c, of the network

854 $\boldsymbol{S}$ Permutation matrix for the pipes in the forest, E_f,of the network

855 $\boldsymbol{T}$ Permutation matrix for the nodes in the forest, E_f, of the network

856 V_c Set of core nodes (vertices) in G_c

857 V_f Set of forest nodes (vertices) in G_f

858 **Hydraulic variables for GGA**

859 $\boldsymbol{A_1}$ Unknown-head node-arc incidence matrix

860 $\boldsymbol{A_2}$ Fixed-head node-arc incidence matrix

861 $\boldsymbol{d}$ Vector of nodal demands

862 $\boldsymbol{d}_i$ Demand of node i

863 $\boldsymbol{D}$ Vector of pipe diameters

864 $\boldsymbol{D}_j$ Diameter of pipe j

865 $\boldsymbol{e_l}$ Vector of Fixed-head nodes elevation heads

866 $\boldsymbol{e}_{\boldsymbol{l}k}$ Fixed-head nodes elevation heads at node k

867 E Set of pipes in graph G

868 $\boldsymbol{EN}$ Vector of end nodes

869 $\boldsymbol{EN}_j$ End nodes of pipe j

870 $\boldsymbol{f}$ Vector of Darcy-Weisbach friction factors

871 $\boldsymbol{f}_j$ Darcy-Weisbach friction factor of pipe j

872 $\boldsymbol{F}$ Diagonal Matrix of generalized headloss derivatives when the headloss is modelled
873 by either the HW and the DW

874 $[F]_{jj}$ Generalized headloss derivatives for pipe j

875 G Full WDS graph

876 $\boldsymbol{G}$ Diagonal matrix with elements $r_j|q_j|^{n-1}$

877 $[G]_{jj}$ $r_j|q_j|^{n-1}$

878 $\boldsymbol{h}$ Vectors of unknown heads

879 $\boldsymbol{h}_i$ Heads at node i

880 $\boldsymbol{J}$ Jacobian matrix

881 L_{1a} Functions run once before multiple simulation

882 L_{2a} Functions run once before hydraulic assessment

883 L_3 Functions run every iteration

884 L_{2b} Functions run once after hydraulic assessment

885 L_{1b} Functions run once after multiple simulation

886 $\boldsymbol{L}$ Vector of pipe lengths

887 $\boldsymbol{L}_j$ Length of pipe j

888 n Head loss exponent

889 n_f Number of forest pipes and nodes

890 n_j Number of junctions

891 n_p Number of pipes

892 n_r Number of fixed-head nodes

893 n_{st} Number of ST pipes and nodes

894 n_{ct} Number of CT pipes

895 $\boldsymbol{q}$ Vector of unknown flows

896 $\boldsymbol{q}_j$ Flow in pipe j

897 $\mathbb{R}$ Vector of Reynolds numbers

898 $\mathbb{R}_j$ Reynolds number for pipe j

899 $\boldsymbol{SN}$ Vector of start nodes

900 $\boldsymbol{SN}_j$ Start nodes of pipe j

901 $\boldsymbol{U}$ Diagonal matrix of Schur Complement when headloss is modelled by HW

902 $\boldsymbol{V}$ Generalized Schur Complement when the headloss is modelled by both the HW
903 and the DW

904 V Set of node in graph G

905 z_i Elevation at node i

906 α_k Interpolating spline coefficient

907 β_k Interpolating spline coefficient

908 $\boldsymbol{\theta}$ Vector as defined in Eq. (24)

909 $\boldsymbol{\epsilon}$ Vector of pipe roughness heights

910 $\boldsymbol{\epsilon}_j$ Roughness height for pipe j

911 **RCTM variables**

912 E_{st} Set of ST pipes (edges)

913 E_{ct} Set of complementary CT pipes (edges)

914 $\boldsymbol{K_1}$ Orthogonal permutation matrix for pipes in the ST

915 $\boldsymbol{K_2}$ Orthogonal permutation matrix for pipes in the CT

916 $\boldsymbol{L_{21}}$ A part of a basis for the null space of the permuted node-arc incidence for the
917 RCTM

918 $\boldsymbol{R}$ Orthogonal permutation matrix for nodes in the ST

919 V_{st} Set of ST nodes (vertices)

920 $\boldsymbol{W}$ Schur complement for the RCTM

This page is intentionally left blank

Appendix B

Submitted version of Publication 2: A Benchmarking Study of Water Distribution System Solution Methods

A Benchmarking Study of Water Distribution System Solution Methods

Mengning Qiu[1], Sylvan Elhay[2], Angus R. Simpson[3], and Bradley Alexander[4]

[1]PhD Student, University of Adelaide, South Australia, 5005. Email:
mengning.qiu@adelaide.edu.au

[2]Visiting Research Fellow, University of Adelaide, South Australia, 5005.

[3]Professor, University of Adelaide, South Australia, 5005.

[4]Senior Lecturer, University of Adelaide, South Australia, 5005.

ABSTRACT

In recent years a number of new WDS solution methods have been developed. These methods have been aimed at improving the speed and reliability of WDS simulations. However, to date, these methods have not been benchmarked against each other in a reliable way. This research addresses this problem by using a newly developed software platform, **WDSLib**, as a fair basis for a detailed comparison of the performance of these methods under different settings. In this work, efficient implementations of three solution methods, the Global Gradient Algorithm (GGA), the forest-core partitioning algorithm (FCPA), and the reformulated co-tree flow method (RCTM), and combinations of these, are compared on eight case study benchmark networks containing between 934 and 19647 pipes and between 848 and 17971 nodes. These simulations were carried out under both a once-off simulation setting and a multiple simulation setting (such as occurs in a genetic algorithm). Timings for these benchmark runs are decomposed into stages so that the performance of these methods can be easily estimated for different settings. The results of this study will help inform the choice of solution methods for given combinations of network features and given design settings. In addition, timing results are compared with EPANET2.

Keywords: water distribution systems solution; Forest-Core Partitioning Algorithm; Global Gradient Algorithm; Reformulated Co-tree Flow Method; hydraulic analysis; EPANET.

INTRODUCTION

Water Distribution Systems (WDSs) are frequently modeled by a system of nonlinear equations, the steady-state solutions of which, the flows and heads in the system, are used in WDS design, management and operation. In a design setting, the solutions might be used as part of an optimization problem to determine the best choices of some network parameters such as pipe diameters. In a management setting, the solutions might be used for the calibration of network parameters such as demand patterns. In an operational environment, new solutions might be needed to adjust control device settings whenever new supervisory control and data acquisition (SCADA) information becomes available.

The most widely used WDS simulation method in current use is the Global Gradient Algorithm (GGA) (Todini and Pilati 1988), which solves the non-linear system of equations representing the WDS. The GGA and its implementations exhibit excellent convergence characteristics for a wide range of starting values and a wide variety of WDS problems. However, some networks have structural properties which can be exploited to further improve the efficiency of the solution process. The GGA, a range space method, exploits the block structure of the full Jacobian matrix in order to produce a smaller key matrix in the linearization of the Newton method. The reformulated co-tree flows method (RCTM) (Elhay et al. 2014), a null-space method (Benzi et al. 2005), can further exploit the block structure of the Jacobian matrix to produce, in realistic WDSs, an even smaller key matrix. This is achieved by dealing separately with the spanning tree and the co-tree in the Newton method linearization.

Another avenue for reducing computation can be exploited by using the Forest-Core Partitioning Algorithm (FCPA) (Simpson et al. 2012) to separate the problem into its linear and non-linear components. The observation underpinning the FCPA is that most WDSs have trees, the collections of which are called forests. The complement of the forest in a network is called the core. The flows in a forest can be computed a-priori by a linear process. Hence, the dimension of the key matrices in the solution process can be significantly reduced when the forest is a large part of the network.

With the development of different solution methods, WDS simulation package users are faced

with a choice of which solution method or methods to apply. Previous publications performed case studies comparing the performances of their respective methods to the GGA. However, these comparisons were often done using different implementation languages, and different levels of code optimization – which makes cross-comparison of methods difficult. Consequently, there is a need for a study which reliably compares the relative performance of these methods using a fast, carefully designed code implementation. To this end, this work presents a thorough benchmark study to compare the performance of GGA, GGA-with-FCPA, RCTM, and RCTM-with-FCPA for a range of case study networks using a fast C++ implementation. The timings for these runs are decomposed according to how often each solution component is executed in different simulation settings. From these timings it is possible to accurately predict runtimes for long-run multiple simulation settings. To confirm the relevance of these results, the timings have been compared with the speed of the industrial and research standard toolkit of EPANET2 (Rossman 2000) and was found to be faster in all cases.

This paper is organized as followed. A detailed review of existing solution methods is given in the next section. The section following presents the mathematical formulation of each method. The motivation for a benchmark study is then given, followed by the methodology used in this paper to carry out a benchmark study. The description of the module categorization is then presented. This is followed by a case study of the four solution methods applied to the eight case study networks. The results are discussed in the next section. The last section offers some conclusions.

LITERATURE REVIEW

This section provides a review of the algorithms that are tested in this paper. A brief development history of WDS solution algorithms is presented in the first subsection. The next subsection gives an overview of the GGA and its development, followed by an overview of solution methods which use the null space approach (such as co-trees flow method (CTM) and RCTM). Finally, a review of the methods that use graph theory to simplify problem complexity are presented.

77 **Development history of the WDS algorithms**

78 This research considers a water distribution model made up of energy conservation equations
79 and the demand driven model continuity equations. The Hardy Cross method (Cross 1936), also
80 known as the loop flow corrections method, is one of the oldest methods and uses successive
81 approximations, solving for each loop flow correction independently. It is a method that was widely
82 used for its simplicity at the time when it was introduced. More than three decades later, Epp and
83 Fowler (1970) developed a computer version of Cross's method and replaced the numerical solver
84 with the Newton method, which solves for all loop flow corrections simultaneously. However, this
85 method has not been widely used because of the need (i) to identify the network loops, (ii) to find
86 initial flows that satisfy continuity equation and (iii) to use pseudo-loops.

87 Many methods have been proposed to improve the computational efficiency of the WDS model.
88 These include: matrix decomposition (Todini and Pilati 1988; Elhay et al. 2014; Deuerlein et al.
89 2015), graph partitioning (Rahal 1995; Simpson et al. 2012; Alvarruiz et al. 2015), network skele-
90 tonization (Saldarriaga et al. 2008; Shamir and Salomons 2008), and network clustering (Anderson
91 and Al-Jamal 1995; Perelman and Ostfeld 2011). Both network skeletonization and network clus-
92 tering can produce a smaller network to solve. However, they are not considered in this study
93 because the solutions from both methods are approximations to the solutions for the full networks,
94 unlike the *exact solutions* produced by the methods used in this study. A summary of methods
95 that improve the computational efficiency of the steady-state demand-driven WDS solution process
96 follows.

97 **Global gradient algorithm**

98 The GGA is a range space method that solves for both flows and heads. It was the first algorithm,
99 in the field of hydraulics, to exploit the block structure of the Jacobian matrix to reduce the size of
100 the key matrix in the linearization of the Newton method. The GGA has gained popularity through
101 its rapid convergence rate for a wide range of starting values. This is the result of using the Newton
102 method on an optimization problem that has a quadratic surface. However, it was reported by Elhay
103 and Simpson (2011) that the GGA fails catastrophically in the presence of zero flows in a WDS

when the head loss is modeled by the Hazen-Williams formula. Regularization methods have been proposed by both Elhay and Simpson (2011) and Gorev et al. (2012) to deal with zero flows when the head loss is modeled by the Hazen-Williams formula.

The GGA as it was first proposed, applied only for the WDSs in which the head loss is modeled by the Hazen-Williams formula, where the resistance factor was independent of flow. In EPANET2, Rossman (2000) extended the GGA to allow the use of the Darcy-Weisbach formula. It has been pointed out in Simpson and Elhay (2010), however, that Rossman incorrectly treated the Darcy-Weisbach resistance factor as independent of the flow. They introduced the correct Jacobian matrix to deal with this. It has been demonstrated that once the correct Jacobian matrix is used, the quadratic convergence rate of the Newton method is restored. Furthermore, Elhay and Simpson (2011) reported that the GGA no longer fails in the presence of zero flows when the derivative of the Jacobian matrix is correctly computed with the Darcy-Weisbach formula.

Null space method

The co-trees flow method (CTM) (Rahal 1995) is a null space method that solves for the co-tree flows and spanning tree flows separately. The CTM, unlike the loop flow corrections method, does not require the initial flows to satisfy continuity. However, it does require: (i) the identification of the associated circulating graph; (ii) the determination of the demands that are to be carried by tree branches; (iii) finding the associated chain of branches closing a circuit for each co-tree chord; (iv) computing pseudo-link head losses. The RCTM (Elhay et al. 2014) is also a null space method that solves co-tree flows and spanning trees flows separately. It represents a significant improvement on the CTM by removing requirements (i) to (iv) above. It uses the Schilders' factorization (Schilders 2009) to permute the node-arc incidence matrix into an invertible spanning tree block and a co-tree block. This permutation reduces the size of the Jacobian matrix from the number of junctions (as in the GGA) to the approximate number of loops in the network.

Abraham and Stoianov (2015) proposed a novel idea to speed-up the solution process when using a null space method to solve a WDS network. Their idea exploits the fact that a significant proportion of runtime is spent computing the head losses. At the same time, flows within some

pipes exhibit negligible changes after a few iterations. As a result, there is no point in wasting computer resources to re-compute the pipe head losses for the pipes that have little or no change in flows. This partial update can be used to economize the computational complexity of the GGA, the RCTM and their variations.

Graph theory

The forest-core partitioning algorithm (FCPA) (Simpson et al. 2012) speeds up the solution process. This algorithm permutes the system equations to partition the linear component of the problem, which is the forest of the WDS, from the non-linear component, which is the core of the WDS. It can be viewed as a method that simplifies the problem by solving for the flows and the heads in the forest just once instead of at every iteration. The FCPA reduces the number of pipes, number of junctions, and the dimension of the Jacobian matrix in the core by the number of forest pipes (or nodes).

The graph matrix partitioning algorithm(GMPA) (Deuerlein et al. 2015) exploited the linear relationships between flows of the internal trees and the flows of the corresponding super-links after the forest of the network has been removed. This was a major breakthrough. The GMPA permutes the node-arc incidence matrix in such a way that all of the nodes with degree two in the core can be treated as a group. By partitioning the network this way, the network can be solved by a global step, which solves for the nodes with degree greater than two (super nodes) and the pipes which connect to them (path chords), and a local step, which solves for the nodes with degree two (interior path nodes) and pipes connected to them (path-tree links).

In a recent paper by Elhay et al. (2018), they proposed a single framework for both the FCPA and GMPA and extended the methods applicability to the pressure dependent problem. Although the flows and heads in the forest component of a pressure driven WDS cannot be determined by a linear process, they can be solved by a similar linear iterative process as the local step in the GMPA.

MOTIVATION

Thus far, this paper has discussed the recent developments in the solution methods. Previous work on WDS simulation has focused on two research areas: (i) hydraulic solution methods (Nielsen

158 1989; Simpson et al. 2012; Elhay et al. 2014; Deuerlein et al. 2015) and (ii) solver software design
159 (Morley et al. 2000; Van Zyl et al. 2003; Guidolin et al. 2010). Two observations can be made
160 when comparing these two areas of research focus:

161 (i) Different platforms have been used to compare algorithm implementations. Some methods
162 have been compared against EPANET, some methods have been implemented by using parts
163 of the EPANET toolkit, some methods have been benchmarked using MATLAB, and others use
164 purpose-written C or C++ code. Comparing timing results between all of these different platforms is
165 especially difficult because MATLAB is a modeling programming language which is not necessarily
166 intended to produce fast production code. As a consequence, the execution of MATLAB code will
167 typically be slower than carefully written C++ code even if the solution method implemented in
168 MATLAB is potentially faster. This will later be discussed in detail.

169 (ii) Timing results can be accurately extrapolated to different design settings only if the imple-
170 mentation code is sufficiently modularized and the timings are available for each module.

171 To address the problems described above, this work describes the methodology employed to
172 ensure a fair comparison between solution methods.

173 NETWORK FORMULATIONS

174 This section provides an unified framework for the mathematical formulations of four solution
175 methods, the GGA with and without FCPA, RCTM with and without FCPA, for WDSs. Each of the
176 solution methods is presented in terms of pure orthogonal permutation of the system of equations
177 to minimize the intermediate steps to ensure a fair comparison between the solution methods. The
178 following starts with the basic definitions and notation, followed by the system equations. The next
179 subsection focuses on the use of network partitioning methods to speed up the solution process for
180 WDSs. Finally, the equations for different solution methods are shown.

181 Definitions and Notation

182 Consider a water distribution system that contains n_p pipes, n_j junctions, n_r fixed head nodes,
183 n_f forest pipes and nodes, n_{p_c} pipes in the core network, n_{j_c} nodes in the core network, n_{ct} pipes
184 in the co-tree network and n_{st} pipes in the spanning tree network. The $j-th$ pipe of the network

can be characterized by its diameter D_j, length L_j, resistance factor r_j. The $i-th$ node of the network has two properties: its nodal demand d_i and its elevation z_{u_i}. Let $\boldsymbol{q} = \left(q_1, q_2, \ldots, q_{n_p}\right)^T$ denote the vector of unknown flows, $\boldsymbol{h} = \left(h_1, h_2, ...h_{n_j}\right)^T$ denote the vector of unknown heads, $\boldsymbol{r} = \left(r_1, r_2, ...r_{n_p}\right)^T$ denote the vector of resistance factors, $\boldsymbol{d} = \left(d_1, d_2, ...d_{n_j}\right)^T$ denote the vector of nodal demands, $\boldsymbol{e_l} = \left(e_{l_1}, e_{l_2}....e_{l_{n_f}}\right)^T$ denote the vector of fixed head elevations.

The head loss exponent n is assumed to be $n = 2$ for the Darcy-Weisbach head loss model and $n = 1.852$ for Hazen-Williams head loss model. The head loss within the pipe j, which connects the node i and the node k, is modeled by $h_i - h_k = r_j q_j |q_j|^{n-1}$. Denote by $\boldsymbol{G}(\boldsymbol{q}) \in \mathbb{R}^{n_p \times n_p}$, a diagonal square matrix with elements $[\boldsymbol{G}]_{ii} = r_i |q_i|^{n-1}$ for $i = 1, 2, ...n_p$. Denote by $\boldsymbol{F}(\boldsymbol{q}) \in \mathbb{R}^{n_p \times n_p}$, a diagonal square matrix where the k-th element on its diagonal $[\boldsymbol{F}]_{kk} = \frac{d}{dq_k}[\boldsymbol{G}]_{kk} q_k$. $\boldsymbol{A_1}$ is the full rank, unknown head, node-arc incidence matrix, where $[\boldsymbol{A_1}]_{ij}$ is used to represent the relationship between pipe i and node j; $[\boldsymbol{A_1}]_{ij} = -1$ if pipe i enters node j, $[\boldsymbol{A_1}]_{ij} = 1$ if pipe i leaves node j, and $[\boldsymbol{A_1}]_{ij} = 0$ if pipe i is not connected to node j. $\boldsymbol{A_2}$ is the fixed-head node-arc incidence matrix, where $[\boldsymbol{A_2}]_{ij}$ is used to represent the relationship between pipe i and fixed head node j, $[\boldsymbol{A_2}]_{ij} = -1$ if pipe i enters fixed head node j, $[\boldsymbol{A_2}]_{ij} = 1$ if pipe i leaves fixed head node j, and $[\boldsymbol{A_2}]_{ij} = 0$ if pipe i is not connected to fixed head node j.

Denote by E_f, the set of indices of the pipes in the forest; by V_f, the set of indices of the nodes in the forest; by E_c, the set of indices of the pipes in the core; and by V_c, the set of indices of the nodes in the core. Denote by E_{st}, the set of indices of the pipes in the spanning tree; by V_{st}, the node indices that correspond with the spanning tree pipes; and by E_{ct}, a set of indices of the pipes in the co-tree.

System of Equations

There are two primary equations that model the underlying relationship of the flows and the heads of a WDS: the demand-driven model (DDM) continuity equations (1) and the energy conservation equations (2):

$$-\boldsymbol{A_1}^T \boldsymbol{q} - \boldsymbol{d} = \boldsymbol{O} \tag{1}$$

$$\boldsymbol{G}(\boldsymbol{q})\,\boldsymbol{q} - \boldsymbol{A_1}\boldsymbol{h} - \boldsymbol{A_2}\boldsymbol{e_l} = \boldsymbol{O}. \tag{2}$$

This non-linear system of equations can be expressed in matrix form as

$$\begin{pmatrix} \boldsymbol{G}(\boldsymbol{q}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{h} \end{pmatrix} - \begin{pmatrix} \boldsymbol{A_2}\boldsymbol{e_l} \\ \boldsymbol{d} \end{pmatrix} = 0 \tag{3}$$

and it is sometimes referred to as a nonlinear saddle point problem (Benzi et al. 2005).

This non-linear system is normally solved by the Newton method, in which $\boldsymbol{q}^{(m+1)}$, a vector of flows at (m+1)-th iteration, and $\boldsymbol{h}^{(m+1)}$, a vector of heads at (m+1)-th iteration, are repeatedly computed from $\boldsymbol{q}^{(m)}$, a vector of flows at (m)-th iteration, and $\boldsymbol{h}^{(m)}$, a vector of heads at (m)-th iteration, by

$$\begin{pmatrix} \boldsymbol{F}^{(m)}(\boldsymbol{q}^{(m)}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}^{(m+1)} - \boldsymbol{q}^{(m)} \\ \boldsymbol{h}^{(m+1)} - \boldsymbol{h}^{(m)} \end{pmatrix} = - \begin{pmatrix} \boldsymbol{G}^{(m)}\boldsymbol{q}^{(m)} - \boldsymbol{A_1}\boldsymbol{h}^{(m)} - \boldsymbol{A_2}\boldsymbol{e_l} \\ -\boldsymbol{A_1}^T\boldsymbol{q}^{(m)} - \boldsymbol{d} \end{pmatrix} \tag{4}$$

until the differences $(\boldsymbol{q}^{(m+1)} - \boldsymbol{q}^{(m)})$ and $(\boldsymbol{h}^{(m+1)} - \boldsymbol{h}^{(m)})$ are sufficiently small.

The Global Gradient Algorithm

The GGA takes advantage of the block structure of Eq. (3) to obtain a two-step solver: Eq. (5) for the heads and Eq. (6) for the flows when the head-loss is modeled by Hazen-William equation.

$$\boldsymbol{U}\boldsymbol{h}^{(m+1)} = -n\boldsymbol{d} + \boldsymbol{A_1}^T\left[(1-n)\,\boldsymbol{q}^{(m)} - \boldsymbol{G}^{-1}\boldsymbol{A_2}\boldsymbol{e_l}\right] \tag{5}$$

where $\boldsymbol{U} = \boldsymbol{A_1}^T\boldsymbol{G}^{-1}\boldsymbol{A_1}$

$$\boldsymbol{q}^{(m+1)} = \frac{1}{n}\{(n-1)\,\boldsymbol{q}^{(m)} + \boldsymbol{G}^{-1}(\boldsymbol{A_2}\boldsymbol{e_l} + \boldsymbol{A_1}\boldsymbol{h})\}. \tag{6}$$

Later, Simpson and Elhay (2010) proposed

$$\boldsymbol{V}\boldsymbol{h}^{(m+1)} = -\boldsymbol{d} + \boldsymbol{A_1}^T \boldsymbol{F}^{-1} \left[(\boldsymbol{G} - \boldsymbol{F})\, \boldsymbol{q}^{(m)} - \boldsymbol{A_2} \boldsymbol{e}_l \right] \tag{7}$$

where $\boldsymbol{V} = \boldsymbol{A_1}^T \boldsymbol{F}^{-1} \boldsymbol{A_1}$

$$\boldsymbol{q}^{(m+1)} = \boldsymbol{q}^{(m)} + \boldsymbol{F}^{-1} \boldsymbol{A_1} \boldsymbol{h}^{(m+1)} - \boldsymbol{F}^{-1} \left[\boldsymbol{G}\boldsymbol{q}^{(m)} - \boldsymbol{A_2} \boldsymbol{e}_l \right] \tag{8}$$

as the generalized equations that can be applied when the head-loss is modeled by the Hazen-William equation or the Darcy-Weisbach equation. The correct Jacobian matrix with the formula for $\boldsymbol{F}$, when head loss is modeled by Darcy-Weisbach equation, can be found in Simpson and Elhay (2010). They showed that the use of the correct Jacobian matrix restores the quadratic rate of convergence.

Network Partitioning

Associated with a WDS is a graph G=(V, E), where the elements of V are the nodes (vertices) of the graph G and elements of E are the pipes (links) of the graph G. In this subsection, the permutation of the system equations (3) for the FCPA is introduced, followed by a description of the RCTM, which further exploits the block structure of the Jacobian matrix.

Forest-Core Partitioning Algorithm

In a demand-driven model, it is possible to exploit the fact that every WDS can be divided into two subgraphs: a treed subgraph (forest) $G_f = \left(V_f, E_f\right)$ and a looped subgraph (core) $G_c = (V_c, E_c)$, so that $E_f \cup E_C = E$, $E_f \cap E_C = \emptyset$, $V_f \cup V_C = V$. All flows and heads in both the forest and the core must be found. The flows in the forest can be found by a linear process before the iterative solution phase and the heads in the forest can be found linearly after the iterative phase.

Simpson et al. (2012) proposed the FCPA, which partitions the network into a treed component and a looped component (referred to as the core) thereby reducing the computation time where the

network has a significant forest component. The FCPA starts by generating a permutation matrix

$$\boldsymbol{P_1} = \begin{array}{c} \\ n_f \\ n_{p_c} \\ n_{j_c} \\ n_f \end{array} \begin{array}{c} \begin{array}{cc} n_p & n_j \end{array} \\ \begin{pmatrix} \boldsymbol{S} & \boldsymbol{O} \\ \boldsymbol{P} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{C} \\ \boldsymbol{O} & \boldsymbol{T} \end{pmatrix} \end{array} \tag{9}$$

, where $\begin{bmatrix} \boldsymbol{S} \\ \boldsymbol{P} \end{bmatrix} \in \mathbb{R}^{n_p \times n_p}$ is the square orthogonal permutation matrix for the pipes, $\boldsymbol{S} \in \mathbb{R}^{n_f \times n_p}$ is the permutation matrix which identifies the pipes in the forest as distinct from those of the core of the WDS, $\boldsymbol{P} \in \mathbb{R}^{n_{p_c} \times n_p}$ is the permutation matrix for the pipes in the core of the WDS, $\begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{T} \end{bmatrix} \in \mathbb{R}^{n_j \times n_j}$ is the square orthogonal permutation matrix for the nodes, $\boldsymbol{C} \in \mathbb{R}^{n_{j_c} \times n_j}$ is the permutation matrix for the nodes in the core of the WDS, $\boldsymbol{T} \in \mathbb{R}^{n_f \times n_j}$ is the permutation matrix which identifies the nodes in the forest as distinct from those of the core of the WDS.

A new lemma is proposed as follows:

LEMMA 1. *Suppose*

$$\mathbf{Q} = \begin{array}{c} \\ m_1 \\ m_2 \end{array} \begin{array}{c} \begin{array}{c} n \end{array} \\ \begin{pmatrix} \mathbf{P} \\ \mathbf{S} \end{pmatrix} \end{array},$$

$\mathbf{Q} \in \mathbb{R}^{n \times n}$*, is an orthogonal permutation matrix and that* $\mathbf{D} = diag\{d_1, d_2, \cdots, d_n\} \in \mathbb{R}^{n \times n}$ *is diagonal. Then*

$$\mathbf{PDS}^T = \mathbf{0} \tag{10}$$

Proof. $P = \left(e_{i_1}, e_{i_2}, \ldots, e_{i_{m_1}}\right)^T$ for a set of indices $T = \{i_1, i_2, \cdots, i_{m_1}\}$ and $S = \left(e_{j_1}, e_{j_2}, \ldots, e_{j_{m_2}}\right)^T$ for a set of indices $V = \{j_1, j_2, \cdots, j_{m_2}\}$. Note that $\mathbf{T} \cap \mathbf{S} = \emptyset$. Now for some $1 \leq i \leq m_1, 1 \leq$

$j \leq m_2$ there exist $i_t \neq j_s$ such that

$$e_i^T \mathbf{PDS^T} e_j = e_{it}^T d_{it} e_{js} = 0$$

from which (10) follows. End of LEMMA 1□

After applying the FCPA permutation, the system equations become

$$\boldsymbol{P_1} \times \begin{bmatrix} \boldsymbol{G} & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{bmatrix} \times \boldsymbol{P_1}^T \times \boldsymbol{P_1} \times \begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{h} \end{pmatrix} - \boldsymbol{P_1} \times \begin{pmatrix} \boldsymbol{A_2 e_l} \\ \boldsymbol{d} \end{pmatrix} = \boldsymbol{O} \tag{11}$$

and with this permutation, Eq. (3) leads to

$$\begin{pmatrix} \boldsymbol{SGS}^T & \boldsymbol{O} & -\boldsymbol{SA_1C}^T & -\boldsymbol{SA_1T}^T \\ \boldsymbol{O} & \begin{bmatrix} \boldsymbol{PGP}^T & -\boldsymbol{PA_1C}^T \\ -\boldsymbol{CA_1}^T\boldsymbol{P}^T & \boldsymbol{O} \end{bmatrix} & \boldsymbol{O} \\ -\boldsymbol{CA_1}^T\boldsymbol{S}^T & & \boldsymbol{O} \\ -\boldsymbol{TA_1}^T\boldsymbol{S}^T & \boldsymbol{O} \quad \boldsymbol{O} & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{Sq} \\ \boldsymbol{Pq} \\ \boldsymbol{Ch} \\ \boldsymbol{Th} \end{pmatrix} - \begin{pmatrix} \boldsymbol{SA_2 e_l} \\ \boldsymbol{PA_2 e_l} \\ \boldsymbol{Cd} \\ \boldsymbol{Td} \end{pmatrix} = \boldsymbol{O} \tag{12}$$

where (i) $\begin{pmatrix} -\boldsymbol{SA_1C}^T & -\boldsymbol{SA_1T}^T \\ -\boldsymbol{PA_1}^T\boldsymbol{C}^T & -\boldsymbol{PA_1T}^T \end{pmatrix}$, which is the original top right two-by-two block in the first matrix of Eq. (12), is the permuted $\boldsymbol{A_1}$ matrix, in which the (2,2) block, which is $-\boldsymbol{PA_1T}^T$, becomes $\boldsymbol{O}$ because the pipes in the core do not connect to any nodes in the forest which are not root nodes, and (ii) $\begin{pmatrix} \boldsymbol{SGS}^T & \boldsymbol{SGP}^T \\ \boldsymbol{PGS}^T & \boldsymbol{PGP}^T \end{pmatrix}$, which is the original top left two-by-two matrix of Eq. (12), is the permuted $\boldsymbol{G}$ matrix, in which it is evident from the Lemma 1 that the (1,2) and (2,1) blocks, which are $\boldsymbol{SGP}^T$ and $\boldsymbol{PGS}^T$, become $\boldsymbol{O}$.

The top right block (the (1,2) block) of the permuted $\boldsymbol{A_1}$ matrix, $-\boldsymbol{SA_1T}^T$, is invertible and can be permuted to be lower triangular form because it represents the union of the trees. The flows of the pipes in the forest, $\boldsymbol{Sq}$ can be found directly from

$$\boldsymbol{Sq} = -\left(\boldsymbol{T}\boldsymbol{A_1}^T\boldsymbol{S}^T\right)^{-1}\boldsymbol{Td}. \tag{13}$$

Rewriting the second and third block equations of Eq. (12) gives:

$$\begin{bmatrix} \boldsymbol{PGP}^T & -\boldsymbol{PA_1C}^T \\ -\boldsymbol{CA_1}^T\boldsymbol{P} & \boldsymbol{O} \end{bmatrix} \begin{pmatrix} \boldsymbol{Pq} \\ \boldsymbol{Ch} \end{pmatrix} = \begin{pmatrix} \boldsymbol{PA_2e}_l \\ \boldsymbol{Cd} + \boldsymbol{CA_1}^T\boldsymbol{S}^T\boldsymbol{Sq} \end{pmatrix}, \tag{14}$$

which is the system for the reduced non-linear problem (for the core heads and flows). This can be expressed as:

$$\begin{bmatrix} \widehat{\boldsymbol{G}} & -\widehat{\boldsymbol{A_1}} \\ -\widehat{\boldsymbol{A_1}}^T & \boldsymbol{O} \end{bmatrix} \begin{pmatrix} \widehat{\boldsymbol{q}} \\ \widehat{\boldsymbol{h}} \end{pmatrix} = \begin{pmatrix} \widehat{\boldsymbol{A_2}}\boldsymbol{e}_l \\ \widehat{\boldsymbol{d}} \end{pmatrix} \tag{15}$$

where $\widehat{\boldsymbol{G}} = \boldsymbol{PGP}^T$, $\widehat{\boldsymbol{A_1}} = \boldsymbol{PA_1C}^T$, $\widehat{\boldsymbol{q}} = \boldsymbol{Pq}$, $\widehat{\boldsymbol{h}} = \boldsymbol{Ch}$, $\widehat{\boldsymbol{A_2}} = \boldsymbol{PA_2}$, and $\widehat{\boldsymbol{d}} = \boldsymbol{Cd} + \boldsymbol{CA_1}^T\boldsymbol{S}^T\boldsymbol{Sq}$ and then the Newton iterative method is applied to Eq. (15).

Finally, once the iterative solution process for the core has stopped, the forest heads can be found by solving a linear system:

$$\boldsymbol{Th} = \left(-\boldsymbol{SA_1T}^T\right)^{-1}\left(\boldsymbol{SA_2e}_l - \boldsymbol{SGS}^T\boldsymbol{Sq} + \boldsymbol{SA_1C}^T\boldsymbol{Ch}\right) \tag{16}$$

after the flows and heads of the core network are found.

The FCPA simplifies the problem by identifying the linear part of the problem and solving it separately from the core to avoid unnecessary computation in the iterative process.

Reformulated co-tree flows method

We first introduce some graph notation before we describe the RCTM in more detail. A spanning tree is an acyclic graph which traverses every node in a graph, such that the addition of any co-tree element creates a loop. A WDS, with or without a forest, can be partitioned into two subgraphs: a spanning tree $G_{st} = (V_{st}, E_{st})$, and a co-tree $G_{ct} = (V_{ct}, E_{ct})$, so that $E_{st} \cup E_{ct} = E_c$, $E_{st} \cap E_{ct} = \emptyset$. The flows in the spanning tree can be found directly from the co-tree flows.

Elhay et al. (2014) proposed the reformulated co-tree flow method (RCTM) to exploit this

relationship between the co-tree flows and spanning tree flows. This is achieved by applying the Schilders' factorization to permute the $\boldsymbol{A_1}$ matrix into a lower triangular square block at the top, representing a spanning tree, and a rectangular block below, representing the corresponding co-tree. The RCTM starts by generating a permutation matrix:

$$\boldsymbol{P_2} = \begin{array}{c} \\ n_{st} \\ n_{ct} \\ n_j \end{array} \begin{array}{c} \begin{array}{cc} n_p & n_j \end{array} \\ \begin{pmatrix} \boldsymbol{K_1} & \boldsymbol{O} \\ \boldsymbol{K_2} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{R} \end{pmatrix} \end{array} \tag{17}$$

where $\begin{bmatrix} \boldsymbol{K_1} \\ \boldsymbol{K_2} \end{bmatrix} \in \mathbb{R}^{n_p \times n_p}$ is the square orthogonal permutation matrix for the pipes, in which $\boldsymbol{K_1} \in \mathbb{R}^{n_{st} \times n_p}$ is the permutation matrix that identifies the pipes in the spanning tree as distinct from those in the co-tree and $\boldsymbol{K_2} \in \mathbb{R}^{n_{ct} \times n_p}$ is the permutation matrix for the pipes in the co-tree, $\boldsymbol{R}$ is the permutation matrix for the nodes to have the same sequence that are traversed by the corresponding spanning tree pipes.

The permuted system equation of the RCTM is:

$$\boldsymbol{P_2} \times \begin{bmatrix} \widehat{\boldsymbol{G}} & -\widehat{\boldsymbol{A}}_{\boldsymbol{1}} \\ -\widehat{\boldsymbol{A}}_{\boldsymbol{1}}{}^T & \boldsymbol{O} \end{bmatrix} \times \boldsymbol{P_2}{}^T \times \boldsymbol{P_2} \times \begin{pmatrix} \widehat{\boldsymbol{q}} \\ \widehat{\boldsymbol{h}} \end{pmatrix} - \boldsymbol{P_2} \times \begin{pmatrix} \widehat{\boldsymbol{A}}_{\boldsymbol{2}} \boldsymbol{e}_l \\ \widehat{\boldsymbol{d}} \end{pmatrix} = \boldsymbol{O} \tag{18}$$

and (14) becomes:

$$\begin{pmatrix} \boldsymbol{K_1} \widehat{\boldsymbol{G}} \boldsymbol{K_1}{}^T & \boldsymbol{O} & -\boldsymbol{K_1} \widehat{\boldsymbol{A}}_{\boldsymbol{1}} \boldsymbol{R}^T \\ \boldsymbol{O} & \boldsymbol{K_2} \widehat{\boldsymbol{G}} \boldsymbol{K_2}{}^T & -\boldsymbol{K_2} \widehat{\boldsymbol{A}}_{\boldsymbol{1}} \boldsymbol{R}^T \\ -\boldsymbol{R} \widehat{\boldsymbol{A}}_{\boldsymbol{1}}{}^T \boldsymbol{K_1}{}^T & -\boldsymbol{R} \widehat{\boldsymbol{A}}_{\boldsymbol{1}}{}^T \boldsymbol{K_2}{}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{K_1} \widehat{\boldsymbol{q}} \\ \boldsymbol{K_2} \widehat{\boldsymbol{q}} \\ \boldsymbol{R} \widehat{\boldsymbol{h}} \end{pmatrix} - \begin{pmatrix} \boldsymbol{K_1} \widehat{\boldsymbol{A}}_{\boldsymbol{2}} \boldsymbol{e}_l \\ \boldsymbol{K_2} \widehat{\boldsymbol{A}}_{\boldsymbol{2}} \boldsymbol{e}_l \\ \boldsymbol{R} \widehat{\boldsymbol{d}} \end{pmatrix} = \boldsymbol{O} \tag{19}$$

in which the (1,2) and (2,1) blocks, which are $\boldsymbol{K_1} \boldsymbol{G} \boldsymbol{K_2}{}^T$ and $\boldsymbol{K_2} \boldsymbol{G} \boldsymbol{K_1}{}^T$, become $\boldsymbol{O}$ for the reasons shown in Lemma 1.

The complexity of the problem is reduced because the (1,3) block of the key matrix, $-\boldsymbol{K_1}\widehat{\boldsymbol{A}}_{\boldsymbol{1}}\boldsymbol{R}^T$, is lower triangular and invertible. As a result, the Newton solver can be partitioned into a different two-step process: (i) solve for the non-linear co-tree flows, (ii) solve for the corresponding spanning tree flows (a linear computation). The heads can be solved once after the iterative process has completed.

By permuting the network equations into (19), Elhay el al. (2014) proposed the following equations to solve the WDS for the flows, first for the spanning tree flows $\boldsymbol{q}_1^{(m+1)}$:

$$\boldsymbol{q}_1^{(m+1)} = \boldsymbol{L_{21}}^T \boldsymbol{q}_2^{(m)} - \boldsymbol{R}_1^{-T}\widehat{\boldsymbol{d}} \tag{20}$$

and second for the co-tree flows $\boldsymbol{q}_2^{(m+1)}$:

$$\boldsymbol{W}^{(m+1)}\boldsymbol{q}_2^{(m+1)} = \boldsymbol{L_{21}}\left(\boldsymbol{F}_1^{(m+1)} - \boldsymbol{G}_1^{(m+1)}\right)\boldsymbol{q}_1^{(m+1)} + \left(\boldsymbol{F}_2^{(m)} - \boldsymbol{G}_2^{(m)}\right)\boldsymbol{q}_2^{(m)} + \boldsymbol{a_2} \tag{21}$$

where: $\boldsymbol{R}_1 = \boldsymbol{K_1}\widehat{\boldsymbol{A}}_{\boldsymbol{1}}\boldsymbol{R}^T$; $\boldsymbol{R}_2 = \boldsymbol{K_2}\widehat{\boldsymbol{A}}_{\boldsymbol{1}}\boldsymbol{R}^T$; $\boldsymbol{L_{21}} = -\boldsymbol{R}_2\boldsymbol{R}_1^{-T}$; $\boldsymbol{F}_1^{(m)} = \boldsymbol{K_1}\widehat{\boldsymbol{F}^{(m)}}\boldsymbol{K_1}^T$; $\boldsymbol{F}_2^{(m)} = \boldsymbol{K_2}\widehat{\boldsymbol{F}^{(m)}}\boldsymbol{K_2}^T$ $\boldsymbol{G}_1^{(m)} = \boldsymbol{K_1}\widehat{\boldsymbol{G}^{(m)}}\boldsymbol{K_1}^T$; $\boldsymbol{G}_2^{(m)} = \boldsymbol{K_2}\widehat{\boldsymbol{G}^{(m)}}\boldsymbol{K_2}^T$; $\boldsymbol{W}^{(m)} = \boldsymbol{L_{21}}(\boldsymbol{F}_1^{(m)})^{-1}\boldsymbol{L_{21}}^T + (\boldsymbol{F}_2^{(m)})^{-1}$; $\boldsymbol{a_1} = \boldsymbol{K_1}\widehat{\boldsymbol{A}}_{\boldsymbol{2}}\boldsymbol{e}_l$; $\boldsymbol{a_2} = \boldsymbol{L_{21}}\boldsymbol{K_1}\widehat{\boldsymbol{A}}_{\boldsymbol{2}}\boldsymbol{e}_l + \boldsymbol{K_2}\widehat{\boldsymbol{A}}_{\boldsymbol{2}}\boldsymbol{e}_l$. Note that in Eq. 20, an initial set of the co-tree flows $\boldsymbol{q}_2^{(0)}$ is needed to commence the solution process.

The heads are found after the iterative process of the RCTM by using a linear solution process:

$$\boldsymbol{R}_1\boldsymbol{h} = \boldsymbol{F}_1\boldsymbol{q}_1^{(m+1)} - (\boldsymbol{F}_1 - \boldsymbol{G}_1)\,\boldsymbol{q}_1^{(m)} - \boldsymbol{a_1} \tag{22}$$

This partitioning of the network equations reduces the size of the non-linear component of the solver to $n_p - n_j$ (the number of co-tree elements in the network). It has been proven by Elhay et al. (2014) that the RCTM and the GGA have identical iterative results and solutions if the same starting values are used. However, for RCTM, the user only needs to set the initial flow estimates for the co-tree pipes, $\boldsymbol{q}_2^{(0)}$, in contrast to GGA where initial flow estimates are required for all pipes. The flows in the complementary spanning tree pipes are generated by Eq.(20).

METHODOLOGY

This section describes the methodology used to carry out a comparative study of the WDS solution methods. The following describes the software platform used to run the benchmarking simulations. This description is followed by the proposed algorithm evaluation method.

The Software Platform

To run the benchmark tests required by this study a hydraulic simulation toolkit, **WDSLib**, was created. This toolkit, written in C++, incorporated the solution methods studied in this paper, which include the GGA, the GGA with the FCPA, the RCTM, and the RCMT with the FCPA. In order to provide a useful platform for comparison, the solution methods were implemented using fast and modularized code. A focus of attention in this research has been the implementation correctness, robustness and efficiency. The correctness* of the toolkit has been validated against a reference MATLAB implementation. The differences between all results (intermediate and final) produced by the C++ toolkit and the MATLAB implementation were shown to be smaller than 10^{-10}. In the interest of toolkit robustness, special attention has been paid to numerical processes to guard against avoidable failures, such as loss of significance through subtractive cancellation, and numerical errors, such as division by zero. The data structures and code libraries in the toolkit are shared and all solution method implementations have been carefully designed to ensure fairness of performance comparisons between algorithms.

The following subsections describe the measures taken in the implementation the solution methods to help ensure the validity of the timing experiments for the case study results. These include measures to ensure accurate timing results, minimization of memory use, and numerical robustness.

Timing Considerations

C++ was chosen as the implementation language because timings in MATLAB are confounded by a variety of factors. The MATLAB programming language is a hybrid of interpreted language and compiled language: some codes are interpreted by MATLAB with no compilation, some

*terms recognized in Computer Science will be designated by asterisk superscript

codes are partially compiled by a closed-source just-in-time (JIT) compiler, and some codes are fully compiled. MATLAB may also perform additional work and bookkeeping between the interpretation of one line and the next.

In contrast, C++ is a compiled language: the compiler translates the code into native machine instructions which are later executed by the hardware. This faster and much simpler model of execution overcomes many of the problems associated with MATLAB timing. As a consequence, a C++ implementation forms a better basis for a fair comparison of different WDS solution methods.

When executing the timing experiments in this work, each code module reports the time spent in it by sampling wall clock time at the start and end of its execution. Although the overhead for sampling wall clock time is small, there are at least two special considerations involved in the interpretation of these timings: (i) the operating system, at its own discretion, may launch background processes (for example anti-virus software), which distort the timings and (ii) extrapolating the timing for multiple simulations (as may occur, for example, in a genetic algorithm or other evolutionary algorithm run) from a single analysis must be done with care because the relationship between the different settings is not linear. More detail on these issues is given in a later section describing the proposed algorithm evaluation method.

Memory Considerations

Memory management for each method was very carefully handled to advantage that method in the interest of a fair comparison. To offer further assurance of the correctness of memory management, Valgrind (Nethercote and Seward 2007), a programming tool for memory debugging, memory leak detection and profiling tool, was deployed during testing to detect any memory leaks, memory corruption, and double-freeing.

Numerical considerations

The calculations in this paper were performed in C++ under IEEE-standard double precision floating point arithmetic with machine epsilon $\epsilon_{mach} = 2.2204^{-16}$. The constants and parameters in every equation were gathered and replaced by full 20-decimal digit accuracy values. In addition, all dependent constants in mathematical expressions were removed.

394 The stopping test used in this benchmark study is $\frac{||\boldsymbol{q}^{(m+1)} - \boldsymbol{q}^{(m)}||_\infty}{||\boldsymbol{q}^{(m+1)}||_\infty} \leq 10^{-6}$ to ensure a fair
395 comparison between the GGA and the RCTM because one of the benefits of using the RCTM is
396 that it only solves for the pipe flows within the iterative phase.

397 In the setup of the benchmark experiments, there are two primary dangers that are associated with
398 floating point arithmetic that cannot be ignored: (i) subtractive cancellation and (ii) overflow and
399 underflow. To avoid problems associated with these, all input variables are scaled to a similar range
400 to minimize the risk of avoidable computational inaccuracy or failure in floating point arithmetic.
401 It is important to note that unscaled or poorly scaled variables can unnecessarily confound the
402 computation. After scaling, variables become physically dimensionless, which allows computation
403 which is independent of the system of measurement units.

404 The variables that are provided in EPANET input files for the experiments and their corre-
405 sponding units in US Customary and SI system are shown in Table 1. The system equations were
406 modified to use dimensionless variables. Once the stopping test has been satisfied, the original
407 variables can then be recovered by reversing the initial scaling.

408 In these experiments approximate minimum degree permutation (AMD) and sparse Cholesky
409 decomposition from SuiteSparse (Davis et al. 2013) have been used. SuiteSparse is a state-of-the-art
410 sparse arithmetic package with exceptional performance.

411 **Proposed algorithm evaluation method**

412 In this work, there are two settings of interest: a once-off network simulation setting and a
413 multi-run setting such as in a genetic algorithm or evolutionary algorithm (EA) that requires many
414 simulations where say the network topology is invariant but pipe diameters can vary. In the case
415 studies presented in this paper results are presented for: (1) a once-off simulation setting and (2) a
416 multi-simulation setting, as might be used in an EA setting for WDS design.

417 In the experiments, in order to avoid unnecessary computations, each module of implementation
418 code is categorized according to the number of times it needs to be invoked in the context of the
419 given setting. This categorization is described in the following subsection.

Module Categorization

For the purposes of modeling execution times, code modules in a multiple simulation run can be divided into three categories: (i) modules that run only once for every *multiple simulation* are called level-1 modules (L_1). The level of a module is determined by the number of times it would be run. Examples of L_1 modules include the module that loads the WDS network configuration file and the module that identifies the forest pipes in FCPA; (ii) modules that are run once for every *simulation* are called level-2 modules (L_2). Examples of L_2 modules are those that initialize, respectively, all pipe flows in the GGA, core flows in FCPA and co-tree flows in RCTM; (iii) modules that are run once for each *iteration* of every simulation are called level-3 modules (L_3). An example of an L_3 module is the module computing the $\boldsymbol{G}$ and $\boldsymbol{F}$ matrices in any of the solution methods described here.

In a once-off network simulation setting, for each trial, a given solver configuration is used to solve an input network and the time to complete the solution is measured. In this setting, the FCPA and RCTM modules require certain computations which only need to be done once every iterative phase. The computation for these so-called invariants can be lifted* out of the iterative phase and executed once per evaluation, thus saving computation time. The second setting considered here is a multiple simulation run, such as one might find in a GA to optimize the design of a WDS, for example. In this setting, a network with a fixed topology is solved multiple times for say different pipe diameters. In this case, because of the fixed topology, the FCPA and RCTM have modules that need only be run once for each multiple simulation run. This again reduces the overall simulation runtime.

CASE STUDIES

The implementation described above was used to evaluate the efficiency of the four solution methods in two simulation settings: a once-off simulation setting, in which the steady-state heads and flows are computed just once with the given WDS parameters, and a design setting, in which the steady-state heads and flows need to be computed many times to, say, find the least-cost design by EA optimization. In the methodology section, the four solution methods, namely GGA, GGA

447 with FCPA, RCTM, and RCTM with FCPA, were decomposed into modules. These modules were
448 categorized into levels by using the method described in the previous section. Fig.1 shows the
449 module classifications and the level of repetition of different modules for the different solution
450 methods. The columns of the block diagram show different solution methods and the rows of the
451 block diagram show the levels of repetition of the steps as they would be executed in a multiple
452 simulation setting. In the body of the table, the different methods are separated by double vertical
453 lines where column(s) intersect a box, which means the modules that are represented by that box
454 are used by the method(s) that are presented by that column(s). For example, the modules for
455 RCTM that are required to be carried out once before a multiple simulation include: (i) load
456 the configuration file and read EPANET input file, (ii) find the Schilders' spanning tree co-tree
457 factorization and (iii) find and apply the AMD bandwidth reduction.

458 Eight benchmark networks were used to study the effectiveness of each method under different
459 design settings. The networks used here were derived from Simpson et al. (2012) with some slight
460 modifications to remove control devices, patterns, curves and pumps. Details of these networks are
461 given in that paper. The basic network characteristics of the case study networks are summarized
462 in Table 2. All the case study networks are realistic. The ratio between the number of the forest
463 pipes and the total number of pipes ranges between 17% and 42%. The ratio between the number
464 of the co-tree pipes and the total number of the pipes ranges between 3% and 31%. Each of
465 the four solution methods and the GGA implementation in the EPANET are applied to these eight
466 benchmark networks. It has been pointed out by Guidolin et al. (2010) that the code implementation
467 in EPANET are highly optimized for its performance and not written to be readily decomposed
468 into modules for different tasks.. As a result, it is difficult, if not impossible, to apply the module
469 categorization method proposed in the current paper to EPANET. The times taken by both `ENopen`,
470 the EPANET function for reading the input file and memory allocation, and `ENclose`, the EPANET
471 function for memory deallocation are not counted in the final EPANET timing.

472 The next section presents the timing analysis for these case study networks. Of course, the same
473 benchmark tests performed on another computing platform will produce quite different results, but

the authors believe that the relative timings will remain the same.

RESULTS AND DISCUSSION

The benchmark tests were performed on a Intel(R) Xeon(R) CPU E5-2698 v3 running at 2.30 GHz with 16 cores and 40MB L_3 cache on a High Performance Computing machine called Phoenix at the University of Adelaide. The number of cores allocated to each test was limited to one. Each timing test was repeated 15 times on each benchmark network.

Once-off Simulation Setting

The mean, minimum, maximum and median wall clock times for all modules were collected. As an example, the detailed statistics of the time for each module of the GGA method applied to N_1, the first case study network. Table 3 presents the detailed timing results of all modules used in the toolkit implementation of the GGA without FCPA at the three levels of repetition: once every multiple simulation (L_1), once every simulation (L_2) and once every solver iteration (L_3). The sub-total for each level is summarized after each level of repetition and the grand-total is shown in the last row. The percentage inside the bracket shows the contribution of each of the modules towards its level of repetition and the contribution of each of the levels towards the total runtime. For example, the AMD permutation contributes 66.9% of the L_1 time and the all L_1 modules contribute 14.8% towards the total runtime. The mean time for a once-off simulation of the N_1 network is 6.75 ms. Of the total time, 84.3% was spent on L_3 tasks. The two most time-consuming tasks are the linear solver in the iterative process, which solves the linearization of the non-linear problem by using Eq. (7), and getGF-2, which computes the derivatives of the head-loss equations.

Table 4 shows the summary statistics of the 15 repetitions of each solution method applied to the eight benchmark networks. Under a once-off simulation setting, the GGA implementation in this paper has been able to achieve between a 26% and 73% speedup when compared with the GGA implementation in EPANET by implementing the proposed module categorization. The best performing algorithm combination for each network is highlighted in **bold**. Both the GGA and the RCTM benefit from the use of the FCPA (between 19.3% and 37.2% of time saved for the GGA, between 7.6% and 21.4% saved for the RCTM).

The number of non-zeros in the key matrices is commonly used as an indicator of the computational complexity of the Cholesky factorization when sparse arithmetic is used. The numbers of non-zeros in the key matrices of the four WDS solution methods are summarized in Table 5. The number of non-zeros in the key matrix of the GGA is a topology-related constant whereas the number of non-zeros in the key matrix of the RCTM is determined by the choice of spanning tree. Network N_8 is the only case where the number of non-zero elements in the key matrix of the RCTM is significantly greater than that of the GGA, therefore network N_8 is the only case where the per-iteration runtime of the RCTM linear solver is greater than that of the GGA (Table 6). Using the FCPA with the GGA can reduce the number of non-zeros in its key matrix. Moreover, the dimension of the non-linear problem reduces from n_p to n_{p_c} which reduces the per-iteration execution time when computing the head loss derivatives, second phase and the stopping test. Although the number of non-zeros in the key matrix of the RCTM is independent of whether or not the FCPA is used, using the FCPA does: (i) reduce the computation time of the matrix multiplication in the linear solver, (ii) reduce the dimension of the search space which speeds up the process of partitioning the co-tree pipes from the spanning tree pipes in the RCTM, and (iii) reduce the number of pipes in the spanning tree. This can be seen by the per-iteration execution times for each of the L_3 modules, which are shown in the Table 6.

The number of iterations required for each of the four solution methods to satisfy the stopping test for the eight case studies networks is shown in the Table 7. It is evident from Table 7 that the GGA took exactly the same number of iterations to satisfy the stopping test with or without the FCPA. The flows in the forest network satisfy a linear system, which does not change from one iteration to the next. Therefore, the flows in the forest pipes reach their steady-state after the first iteration. Similarly, the RCTM with or without FCPA takes the same number of iterations. In the cases that were analyzed in this study, the RCTM required a greater number of iterations to satisfy the stopping test compared to the GGA. This is because different mechanisms are used to generate a set of initial flows for the two methods as discussed previously.

It is worth using the FCPA in conjunction with both the GGA and RCTM for a once-off

simulation given that FCPA decreases the L_3 per-iteration time without increasing the number of iterations per module. Interestingly, a smaller per-iteration time is required by the L_3 modules of the RCTM except for network N_8. However, RCTM requires a greater number of iterations for all the case study networks. This sometimes causes a greater time for the RCTM to satisfy the stopping test.

Multiple Simulation Setting

The performance of the four solution methods under the multiple simulation setting are compared. Pipe diameters for the eight case study networks were randomly generated at each evaluation to simulate an evolutionary algorithm run. It is important to note that the use of randomly generated pipe diameters gives an overestimate of the total runtime. This is because, as EA's progress, the pipe diameters in its population become increasingly realistic, which, on average, should reduce the number of iterations at the L_3 level.

Table 8 and Table 9 show the detailed timing results of multiple simulations with number of evaluations $N_E = 100,000$ for each of the four solution methods applied to the networks N_1 and N_8. Table 8 shows that exploiting the treed nature of network N_1 gives the FCPA a 29% time saving over the GGA and 15% time saving over the RCTM. A smaller saving is achieved by the use of the FCPA for network N_8: 14% for the GGA and 9% for the RCTM. In a multiple simulation setting, the RCTM is more timing-consuming than the GGA when applied to network N_8 because of the greater number of nonzero elements in its key matrix (Table 5).

Table 10 shows the actual multiple simulation runtime with 100,000 evaluations for each of the four solution methods applied to the eight case study networks. Under a multi-run simulation setting, the GGA implementation in this paper has been able to achieve between a 35% and 81% speedup when compared with the GGA implementation in EPANET by implementing the proposed module categorization. Note that both the upper and lower range values of the speed-up achieved by implementing the proposed module categorization in a multi-run simulation are higher than those in a once-off simulation. This is because the effectiveness of proposed module categorization and the number of evaluation are directly proportional. The fastest solution methods for each of the

555 case study networks are highlighted in bold. Both the GGA and the RCTM benefit from the use of
556 the FCPA, which is also observed under the once-off simulation setting. The relative time saving
557 accruing from the use of the FCPA is smaller for the RCTM than it is for the GGA.

558 CONCLUSIONS

559 This paper presents a reliable benchmark study on four water distribution system demand-driven
560 steady-state solution methods, namely the Global Gradient Algorithm (GGA), the GGA with Forest-
561 Core Partitioning Algorithm (FCPA), the Reformulated Co-Tree flow Method (RCTM), and the
562 RCTM with FCPA. These solution methods were implemented using fast, carefully designed, and
563 modularized C++ code in order to provide a fair basis for comparing these methods.

564 The correctness of the implemented solution methods in C++ has been validated against a MAT-
565 LAB implementation. The robustness of the implementation was achieved by: (i) incorporating
566 necessary precautions in the numerical processes to guard against avoidable computational failures,
567 (ii) using Valgrind to detect any memory leaks and (iii) scaling the variables to avoid overflow,
568 underflow and subtractive cancellation. Implementation efficiency was achieved by, (i) identifying
569 the program loop invariants and hoisting them out of the program loop to avoid any unnecessary
570 computations and (ii) gathering the constants and parameters in every equation to minimize the
571 number of parameters.

572 The following observations can be made for the four solution methods by comparing the detailed
573 timing of four WDS solution methods applied to the eight case study networks:

574 1. In the case studies that have been analyzed, the per-iteration time required to perform the
575 RCTM is less than the GGA except for N_8. However, the RCTM requires a greater number
576 of iterations to satisfy the stopping test which leads to the RCTM requiring more time than
577 the GGA for some case study networks. This is because of the different mechanisms used to
578 generate the initial pipe flow guesses in these methods.

579 2. In the case studies analyzed, the mean time per-iteration of the L_3 modules (iterative solution
580 procedure to solve the nonlinear equations) is affected by the number of non-zeros in the key

matrix and the dimension of the non-linear problem. The smaller the number of non-zeros and the smaller the dimension of the non-linear problem, the smaller the solution time will be.

3. Both the GGA and the RCTM benefit from partitioning the forest component from the core component. The FCPA saves less time for the RCTM than it does for the GGA because the forest component is a part of the spanning tree calculation.

4. Significant time savings have been observed when comparing the implemented solution methods with EPANET for a multiple run simulation setting.

As a final note, a significant proportion of the runtime savings, in the method implementation, can be attributed to the decomposition of the modules of the solution methods into different levels of repetition. This decomposition exploits invariants in the solution process in order to avoid unnecessary computations.

ACKNOWLEDGEMENT

This work was supported with supercomputing resources provided by the Phoenix HPC service at the University of Adelaide.

SUPPLEMENTAL DATA

The data for case study networks N_1, N_3, N_4, and N_7, which are modifications of networks in the public domain, are available on-line. The data for case study networks N2, N5, N6, and N8 are not freely available either because they are proprietary or because of security concerns on the part of the relevant water utilities.

REFERENCES

Abraham, E. and Stoianov, I. (2015). “Sparse null space algorithms for hydraulic analysis of large-scale water supply networks.” *Journal of Hydraulic Engineering*, 04015058.

Alvarruiz, F., Martínez-Alzamora, F., and Vidal, A. (2015). “Improving the efficiency of the loop method for the simulation of water distribution systems.” *Journal of Water Resources Planning and Management*, 141(10), 04015019.

607 Anderson, E. J. and Al-Jamal, K. H. (1995). "Hydraulic-network simplification." *Journal of Water*
608 *Resources Planning and Management*, 121(3), 235–240.

609 Benzi, M., Golub, G., and Liesen, J. (2005). "Numerical solution of saddle point problems." *Acta.*
610 *Num.*, 1–137.

611 Cross, H. (1936). "Analysis of flow in networks of conduits or conductors." *University of Illinois.*
612 *Engineering Experiment Station. Bulletin; no. 286*.

613 Davis, T., Hager, W., and Duff, I. (2013). "Suitesparse." *URL< http://www. cise. ufl.*
614 *edu/research/sparse/SuiteSparse*.

615 Deuerlein, J., Elhay, S., and Simpson, A. (2015). "Fast graph matrix partitioning algorithm for
616 solving the water distribution system equations." *Journal of Water Resources Planning and*
617 *Management*, 142(1), 04015037.

618 Elhay, S., Deuerlein, J., Piller, O., and Simpson, A. R. (2018). "Graph partitioning in the analysis
619 of pressure dependent water distribution systems." *Journal of Water Resources Planning and*
620 *Management*, 144(4), 04018011.

621 Elhay, S. and Simpson, A. R. (2011). "Dealing with zero flows in solving the nonlinear equations
622 for water distribution systems." *Journal of Hydraulic Engineering*, 137(10), 1216–1224.

623 Elhay, S., Simpson, A. R., Deuerlein, J., Alexander, B., and Schilders, W. H. (2014). "Reformulated
624 co-tree flows method competitive with the global gradient algorithm for solving water distribution
625 system equations." *Journal of Water Resources Planning and Management*, 140(12), 04014040.

626 Epp, R. and Fowler, A. G. (1970). "Efficient code for steady-state flows in networks." *Journal of*
627 *the Hydraulics Division*, 96(1), 43–56.

628 Gorev, N. B., Kodzhespirov, I. F., Kovalenko, Y., Prokhorov, E., and Trapaga, G. (2012). "Method
629 to cope with zero flows in newton solvers for water distribution systems." *Journal of hydraulic*
630 *engineering*, 139(4), 456–459.

631 Guidolin, M., Burovskiy, P., Kapelan, Z., and Savic, D. (2010). "Cwsnet: An object-oriented toolkit
632 for water distribution system simulations." *Proceedings of the 12th Annual Water Distribution*
633 *Systems Analysis Conference, WDSA*, 12–15.

634 Morley, M., Atkinson, R., Savic, D., and Walters, G. (2000). "Opennet: An application independent
635 framework for hydraulic network representation, manipulation & dissemination." *Hydroinformat-*
636 *ics 2000 Conference, University of Iowa, Iowa City, USA*, 23–27.
637 Nethercote, N. and Seward, J. (2007). "Valgrind: a framework for heavyweight dynamic binary
638 instrumentation." *ACM Sigplan notices*, Vol. 42, ACM, 89–100.
639 Nielsen, H. B. (1989). "Methods for analyzing pipe networks." *Journal of Hydraulic Engineering*,
640 115(2), 139–157.
641 Perelman, L. and Ostfeld, A. (2011). "Topological clustering for water distribution systems analy-
642 sis." *Environmental Modelling & Software*, 26(7), 969–972.
643 Rahal, H. (1995). "A co-tree flows formulation for steady state in water distribution networks."
644 *Advances in Engineering Software*, 22(3), 169–178.
645 Rossman, L. A. (2000). "Epanet 2 users manual, us environmental protection agency." *Water Supply*
646 *and Water Resources Division, National Risk Management Research Laboratory, Cincinnati, OH*,
647 45268.
648 Saldarriaga, J., Ochoa, S., Rodriguez, D., and Arbeláez, J. (2008). "Water distribution network
649 skeletonization using the resilience concept." *Water Distribution Systems Analysis 2008*, 1–13.
650 Schilders, W. H. (2009). "Solution of indefinite linear systems using an lq decomposition for the
651 linear constraints." *Linear Algebra and its Applications*, 431(3), 381–395.
652 Shamir, U. and Salomons, E. (2008). "Optimal real-time operation of urban water distribution
653 systems using reduced models." *Journal of Water Resources Planning and Management*, 134(2),
654 181–185.
655 Simpson, A. and Elhay, S. (2010). "Jacobian matrix for solving water distribution system equations
656 with the darcy-weisbach head-loss model." *Journal of hydraulic engineering*, 137(6), 696–700.
657 Simpson, A. R., Elhay, S., and Alexander, B. (2012). "Forest-core partitioning algorithm for
658 speeding up analysis of water distribution systems." *Journal of Water Resources Planning and*
659 *Management*, 140(4), 435–443.
660 Todini, E. and Pilati, S. (1988). "A gradient algorithm for the analysis of pipe networks." *Computer*

661 *applications in water supply: vol. 1—systems analysis and simulation*, Research Studies Press
662 Ltd., 1–20.
663 Van Zyl, J., Borthwick, J., and Hardy, A. (2003). "Ooten: An object-oriented programmers toolkit
664 for epanet." *Advances in Water Supply Management (CCWI 2003)*, 1–8.

List of Tables

690 10 The actual multiple simulation runtime with 100,000 evaluations (min.) for each
691 of the four solution methods applied to each of the eight case study networks and
692 the % diff. refers to relative difference compared to the GGA time . . . . . .

TABLE 1. WDS variables and units

Variables	SI unit	US unit	Scaling factor
Length	m	ft	$L_0 = max\,(\boldsymbol{L})$
Diameter	m	ft	$D_0 = max\,(\boldsymbol{D})$
Nodal head	m	ft	$h_0 = max\,(\boldsymbol{e_l})$
Source elevation	m	ft	$e_{l_0} = max\,(\boldsymbol{e_l})$
flow	m^3/s	ft^3/s	$q_0 = max\,(\boldsymbol{d})$
demand	m^3/s	ft^3/s	$d_0 = max\,(\boldsymbol{d})$
G[1], F[1]	s/m^2	s/ft^2	$G_0 = \frac{L_0}{D_0^p}\lvert q_0\rvert^{n-1}$

TABLE 2. Benchmark networks summary

	Full Network			Forest & Core Networks			Co-tree Network
Network	n_p	n_j	n_s	$n_f(n_f/n_p^{\#})$	n_{j_c}	n_{p_c}	n_{ct}
N_1	934	848	8	361 (38%)	573	487	84
N_2	1118	1039	2	321 (29%)	797	718	79
N_3	1975	1770	4	823 (42%)	1152	947	205
N_4	2465	1890	3	429 (17%)	2036	1461	757
N_5	2509	2443	2	702 (28%)	1087	1741	66
N_6	8585	8392	2	1850 (22%)	6735	6542	193
N_7	14830	12523	7	2932 (20%)	11898	9591	2307
N_8	19647	17971	15	4414 (22%)	15232	13557	1676

[#] n_f/n_p shows the proportion of the forest

TABLE 3. Detailed statistics of the time of each module of the GGA applied to network N_1 (15 runs)

Type	Function	Statistical Properties (milliseconds)						
		Min	Max	Mean(%)		Median	Std. Dev	Std. Err
Computed once every multiple simulation								
L_1	AMD	0.54	1.45	0.67(66.9%)		0.57	0.20	0.04
	Housekeeping	0.28	0.58	0.33 (33.1%)		0.29	0.09	0.02
Sub-Total Statistics		0.82	1.74	1.00 (14.8%)		0.90	0.20	0.04
Computed once every simulation								
L_2	GetGF-1	0.01	0.01	0.01(23.1%)		0.01	0.00	0.00
	init	0.00	0.00	0.00(5.4%)		0.00	0.00	0.00
	scaling	0.04	0.05	0.04 (66.3%)		0.04	0.00	0.00
	Housekeeping	0.00	0.00	0.00(5.2%)		0.00	0.00	0.00
Sub-Total Statistics		0.05	0.07	0.06 (0.8%)		0.06	0.00	0.00
Iterative Phase								
					T_{L_3} /N_I			
L_3 (N_I=8)	GetGF-2	1.19	2.52	1.42(24.9%)	0.18	1.28	0.30	0.02
	Linear Solver	3.11	4.32	3.53(62.1%)	0.44	3.49	0.31	0.06
	2nd Phase	0.31	0.59	0.36(6.4%)	0.05	0.33	0.08	0.02
	normTest	0.11	0.57	0.14(2.4%)	0.02	0.12	0.09	0.02
	Other	0.18	0.85	0.24(4.2%)	0.03	0.19	0.14	0.03
Sub-Total Statistics		4.91	6.52	5.69(84.3%)	0.71	5.69	0.41	0.00
Computed once every multiple simulation								
L_1	UndoPermutation	0.00	0.00	0.00(100%)		0.00	0.00	0.00
Sub-Total Statistics		0.00	0.00	0.00(0.05%)		0.00	0.00	0.00
Grand-Total Statistics		6.02	7.55	6.75		6.67	0.44	0.08

TABLE 4. The mean time of once-off simulation run averaged over 15 once-off simulations for each of the four solution methods applied eight case study networks (milliseconds$\pm$standard error) and the % diff. refers to relative difference compared to the GGA mean time

	GGA	GGA with FCPA		RCTM		RCTM with FCPA		EPANET	
	Mean time	Mean time	%diff.	Mean time	%diff.	Mean time	%diff.	Mean time	%diff.
N_1	6.75 ± 0.08	4.66 ± 0.07	-31	5.37 ± 0.09	-20	$\mathbf{4.56 \pm 0.08}$	-32	9.09	+35
N_2	8.48 ± 0.07	$\mathbf{6.61 \pm 0.07}$	-22	9.98 ± 0.12	+18	8.97 ± 0.08	+6	16.75	+98
N_3	13.88 ± 0.11	$\mathbf{8.72 \pm 0.09}$	-37	11.52 ± 0.10	-17	9.05 ± 0.06	-35	21.46	+55
N_4	14.63 ± 0.32	$\mathbf{12.68 \pm 0.53}$	-19	17.09 ± 0.85	+47	16.28 ± 0.47	+35	26.45	+81
N_5	16.87 ± 0.24	12.19 ± 0.13	-28	12.67 ± 0.14	-25	$\mathbf{10.20 \pm 0.13}$	-40	28.46	+69
N_6	49.53 ± 0.19	44.79 ± 0.18	-28	35.34 ± 0.17	-29	$\mathbf{32.53 \pm 0.15}$	-39	172.84	+249
N_7	83.39 ± 0.42	$\mathbf{63.06 \pm 0.65}$	-24	169.50 ± 1.61	+103	156.61 ± 1.16	+88	307.17	+268
N_8	192.10 ± 3.85	$\mathbf{131.82 \pm 4.0}$	-31	352.44 ± 9.25	+83	307.16 ± 7.2	+60	600.08	+212

TABLE 5. The number of non-zeros in the key matrices of each of the four solution methods applied to the eight case studies networks and the relative diff. refers to the relative difference compared to the number of non-zeros in the key matrix of the GGA

	GGA	GGA with FCPA	Relative diff. using FCPA	RCTM	RCTM with FCPA	Relative diff. using RCTM and RCTM with FCPA
N_1	2684	1609	-40%	350	350	-87%
N_2	3265	2302	-29%	1219	1219	-63%
N_3	5708	3239	-43%	2534	2534	-56%
N_4	6714	5429	-19%	6951	6951	+3.5%
N_5	7451	5345	-28%	551	551	-93%
N_6	25554	20004	-22%	2514	2514	-90%
N_7	41147	32351	-21%	32389	32389	-21%
N_8	57233	43991	-23%	73252	73252	+28%

TABLE 6. The mean of per-iteration timings for each of the modules in L_3 for the four solution methods applied to the eight case studies (milliseconds)

	GGA				GGA+FCPA			
	GetGF	Linear Solver[#]	2nd Phase	norm test	GetGF	Linear Solver[#]	2nd Phase	norm test
N_1	0.18	0.44	0.36	0.14	0.14	0.27	0.03	0.01
N_2	0.22	0.61	0.02	0.02	0.21	0.39	0.04	0.01
N_3	0.20	1.11	0.03	0.03	0.12	0.56	0.07	0.02
N_4	0.66	1.67	0.05	0.03	0.47	1.36	0.03	0.02
N_5	0.42	1.47	0.04	0.03	0.33	0.98	0.03	0.02
N_6	0.48	1.49	0.05	0.04	0.40	0.96	0.03	0.01
N_7	1.92	5.70	0.23	0.09	1.57	3.94	0.22	0.07
N_8	3.17	12.38	0.38	0.21	2.86	7.72	0.33	0.12
	RCTM				RCTM+FCPA			
	GetGF	Linear Solver[#]	2nd Phase	norm test	GetGF	Linear Solver[#]	2nd Phase	norm test
N_1	0.16	0.14	0.02	0.01	0.14	0.11	0.02	0.01
N_2	0.22	0.29	0.05	0.02	0.20	0.26	0.04	0.01
N_3	0.20	0.50	0.07	0.02	0.11	0.41	0.06	0.02
N_4	0.56	1.54	0.11	0.03	0.45	1.47	0.10	0.03
N_5	0.43	0.39	0.07	0.03	0.32	0.31	0.05	0.02
N_6	0.47	0.37	0.07	0.03	0.42	0.30	0.05	0.02
N_7	1.90	5.49	0.43	0.10	1.65	5.30	0.40	0.08
N_8	3.51	17.71	1.68	0.22	3.08	15.97	1.35	0.17

[#] within the iterative solution process

TABLE 7. The number of iterations required for each of the four solution methods to satisfy the stopping test for the eight case studies networks. The relative diff. refers to the relative difference compared to the number of iterations for the GGA

	GGA	GGA with FCPA	RCTM	RCTM with FCPA	Relative diff. using RCTM
N_1	8	8	12	12	+50%
N_2	8	8	13	13	+62.5%
N_3	8	8	9	9	+12.5%
N_4	9	9	13	13	+44.4%
N_5	8	8	10	10	+25%
N_6	10	10	12	12	+20%
N_7	9	9	13	13	+44.4%
N_8	9	9	11	11	+22.2%

TABLE 8. The actual time required to perform a multiple simulation, where number of evaluations $N_E = 100,000$, of each of the four solution methods applied to N_1 network (ms unless otherwise stated) and % diff. refers to the relative difference compared to the GGA

		GGA	GGA with FCPA		RCTM		RCTM with FCPA	
		(ms)	(ms)	% diff.	(ms)	% diff.	(ms)	% diff.
L_1	AMD	1.36	0.64		0.19		0.14	
	FCPA	-	0.66		-	-	0.16	
	RCTM	-	-		0.53		0.33	
	scaling	0.09	0.05		0.04		0.02	
	HouseKeeping	2.05	1.78		3.93		0.36	
	Sub-Total	3.50	3.01	-14%	4.69	+34%	1.01	-71%
L_2	GetGF-1	2790.15	1899.44		588.75		422.93	
	init	1345.87	887.32		1703.29		1311.09	
	HouseKeeping	533.19	380.98		811.72		626.93	
	Sub-Total	4669.21	3167.74	-32%	3103.76	-34%	2360.95	-49%
L_3	GetGF-2	105292.0	89779.9		105596.0		98439.6	
	Linsolve	166072.0	100730.0		122200.0		95539.0	
	second phase	36483.3	23477.1		19716.9		14872.4	
	normTest	50892.4	34836.7		12753.1		9440.8	
	HouseKeeping	15748.3	12593.3		6605.0		6340.2	
	Sub-Total	374488	261417	-30%	266871	-29%	224632	-40%
L_2	reverseFCPA	-	6053.5		-		1776.4	
	reverseRCTM	-	-		1335.5		824.5	
	Sub-Total	0	6053.5	—	1335.5	—	2600.93	—
L_1	undo permutation	0.02	0.01		0.002		0.002	
	Sub-Total	0.02	0.01	-40%	0.002	-89%	0.002	-91%
		(min.)	(min.)		(min.)		(min.)	
	EA runtime	6.35	4.53	-29%	4.53	-29%	3.83	-40%

TABLE 9. The actual time required to perform a multiple simulation, where number of evaluations $N_E = 100,000$, of each of the four solution methods applied to N_8 network (ms unless otherwise stated). The % diff. refers to the relative difference compared to the GGA

		GGA	GGA with FCPA		RCTM		RCTM with FCPA	
		(ms)	(ms)	% diff.	(ms)	% diff.	(ms)	% diff.
L_1	AMD	14.16	8.49		10.68		8.60	
	FCPA	-	1.75		-		1.64	
	RCTM	-	-		45.30		24.11	
	scaling	0.69	0.36		0.47		0.34	
	housekeeping	7.86	5.39		8.02		5.70	
Sub-Total		22.71	15.99	-30%	64.47	+184%	40.39	+78%
L_2	GetGF-1	10069.00	7481.70		9920.71		7375.28	
	init	2362.45	1782.70		73221.90		65377.80	
	housekeeping	1686.00	1342.04		42063.10		43556.10	
Sub-Total		14117.45	10606.44	-25%	125205.71	+787%	116309.18	+723%
L_3	GetGF-2	2331270.0	2173440.0		3732280.0		3561510.0	
	Linsolve	6884030.0	5689170.0		11339200.0		10975000.0	
	second phase	314826.0	280212.0		1129820.0		995822.0	
	normTest	162986.0	112226.0		257123.0		183340.0	
	housekeeping	40008.0	33472.0		54777.0		47128.0	
Sub-Total		9733120	8288520	-15%	16513200	+70%	15762800.0	+62%
L_2	reverseFCPA	-	18405.5		-		19017.2	
	reverseRCTM	-	-		24182.3		16772.6	
Sub-Total		0	18405.5		24182.3		35789.8	
L_1	undo permutation	0.03	0.03		0.06		0.06	
Sub-Total		0.03	0.03	-13%	0.06	+83%	0.06	+64%
		(min.)	(min.)		(min.)		(min.)	
EA runtime		162.51	138.68	-15%	277.80	+71%	265.34	+63%

TABLE 10. The actual multiple simulation runtime with 100,000 evaluations (min.) for each of the four solution methods applied to each of the eight case study networks and the % diff. refers to relative difference compared to the GGA time

	GGA	GGA with FCPA		RCTM		RCTM with FCPA		EPANET	
	min.	min.	%diff.	min.	%diff.	min.	%diff.	min.	%diff.
N_1	6.35	4.35	-31	4.53	-29	**3.83**	-40	9.70	+53
N_2	8.39	**5.83**	-31	8.67	+3	7.78	-7	13.96	+66
N_3	14.88	**9.64**	-35	12.45	-16	10.30	-31	25.35	+70
N_4	20.86	**16.86**	-19	24.02	+15	21.93	+5	68.08	+226
N_5	16.47	12.50	-24	10.90	-34	**9.26**	-44	32.18	+95
N_6	70.66	58.62	-17	39.58	-44	**36.25**	-49	238.64	+238
N_7	128.01	**94.71**	-26	216.88	+69	204.59	+60	422.62	+230
N_8	162.511	**138.68**	-15	277.80	+71	265.34	+63	843.03	+419

693 ## List of Figures

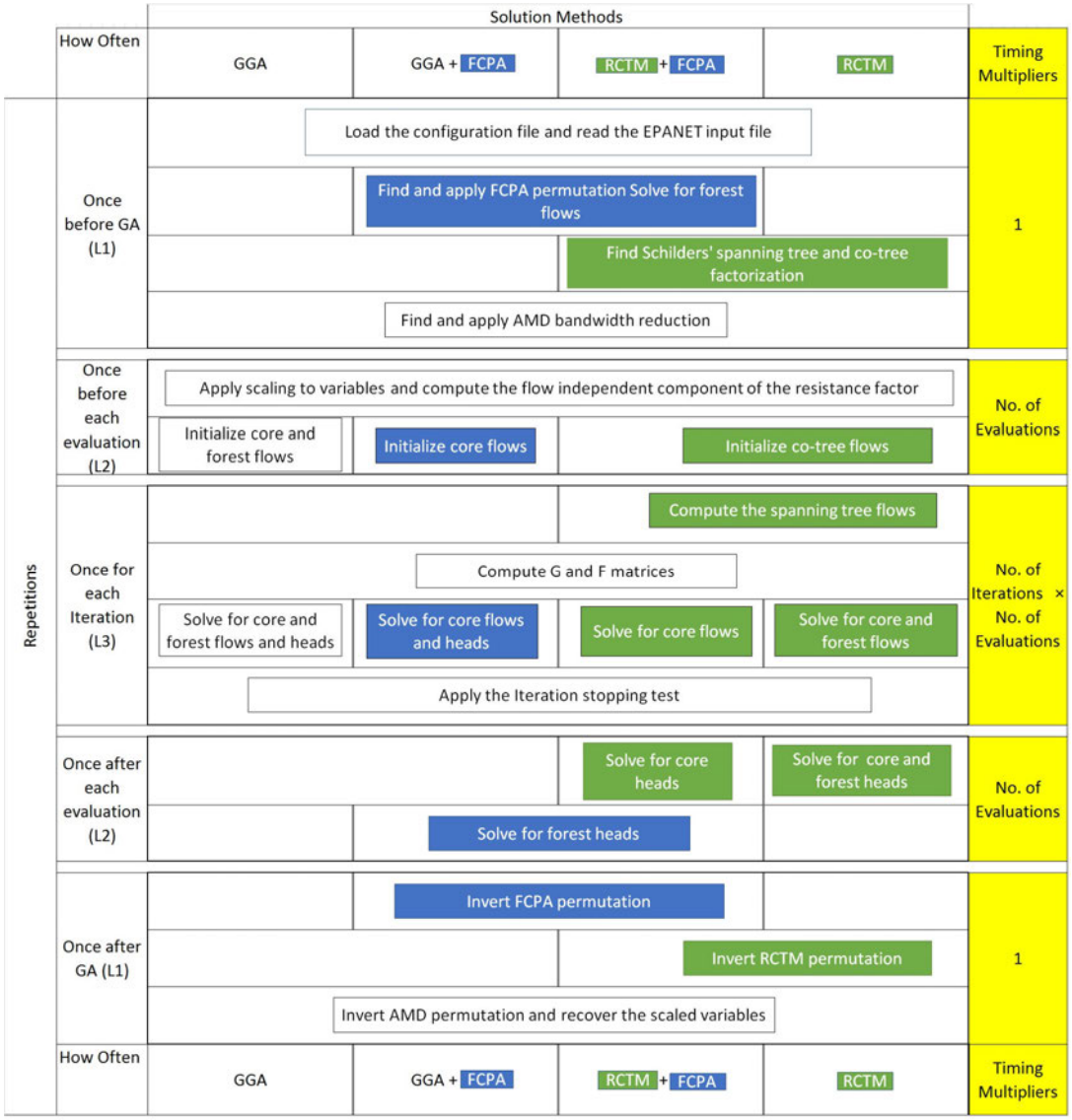


Fig. 1. Module classification for GGA, GGA and FCPA, RCTM and RCTM with FCPA

Appendix C

Submitted version of Publication 3: A Bridge-Block Partitioning Algorithm for Speeding up Analysis of Water Distribution Systems

A Bridge-Block Partitioning Algorithm for Speeding up Analysis of Water Distribution Systems

Mengning Qiu[1], Angus R. Simpson[2], Sylvan Elhay[3], and Bradley Alexander[4]

[1]PhD Student, University of Adelaide, South Australia, 5005. Email:
mengning.qiu@adelaide.edu.au

[2]Professor, University of Adelaide, South Australia, 5005.

[3]Visiting Research Fellow, University of Adelaide, South Australia, 5005.

[4]Senior Lecturer, University of Adelaide, South Australia, 5005.

ABSTRACT

Many water distribution system (WDS) solution methods have been developed to perform demand-driven steady-state analysis. These methods are used to solve the non-linear system of equations that model a WDS. WDS networks have structural properties that can often be exploited to speed up these solution methods. One solution method that exploits these structural properties is the forest-core partitioning algorithm that was proposed as a pre-processing and post-processing method that can be used to separate the network into a linear forest component and a non-linear core component. This paper presents a complementary method for pre-and post-processing called the bridge-block partitioning algorithm (BBPA). This method further partitions the core component of the network into a number of linear bridge components and a number of non-linear block components. The use of BBPA to partition a WDS network provides significant advantages over current solution methods in terms of both speed and solution reliability.

INTRODUCTION

Hydraulic simulation algorithms use mathematical models designed to simulate the hydraulic performance of a water distribution system (WDS) and have played a critical role in the design,

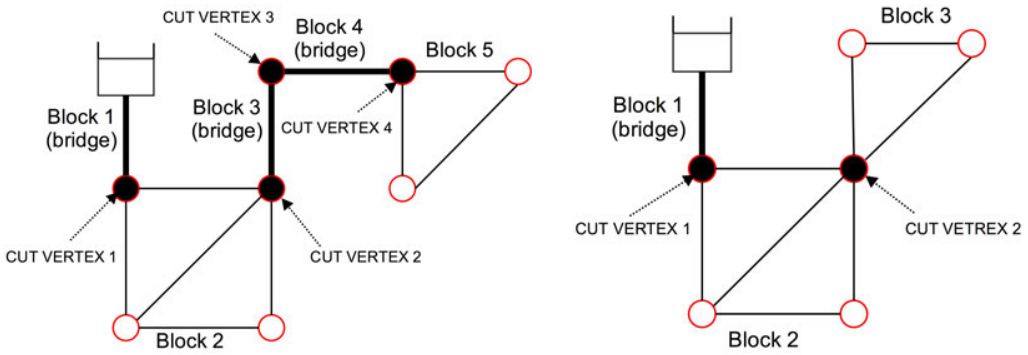
operation, and management of WDSs in research and industry. These models have been used for (1) optimizing WDS network design parameters (such as pipe diameters), (2) for calibrating network parameters (such as demand patterns), (3) conducting real-time monitoring and calibration of the network elements in a supervisory control and data acquisition (SCADA) operational setting, and (4) adjusting control devices (such as valves). In hydraulic simulation, the system of equations can be formulated as a large and sparse non-linear saddle-point problem. There are several well-known iterative methods for solving the non-linear saddle-point problem. These include: range space methods, null space methods, and loop-based methods.

The most widely used WDS solution method is the Global Gradient Algorithm (Todini and Pilati 1988). The GGA, a range space method, takes advantage of the block structure of the full Jacobian matrix to achieve a smaller key matrix in the linearization of the Newton method. Since the development of the GGA, numerous new WDS hydraulic solution methods have been proposed and improvements have been made to existing WDS hydraulic solution methods. Most of these new WDS hydraulic solution methods employ graph theory to decompose or partition the WDS network graph into sub-graphs which results in a smaller system of equations. Deuerlein (2008) introduced a decomposition model for a WDS network graph, in which the one-connected components are categorized as the forest component and the biconnected components are categorized as the core component. After removing the forest component, the core component can be further partitioned into blocks that are connected by bridge elements. After the partitioning processes, a loop flow corrections method is then used. Simpson et al. (2012) proposed a matrix based identification method for the forest component and the core component and introduced the forest-core partitioning algorithm (FCPA). In the FCPA, flows and heads in the forest component can be solved for just once. The remaining system of equations, representing the core – which has a smaller dimension if the network has a significant forest component – is then solved iteratively by the Newton method. Deuerlein et al. (2015) proposed another graph partitioning algorithm which exploits the properties of network components in series within the core component of the network. This algorithm exploits the fact that flows in the internal tree pipes are linearly dependent on the topological minor. This

relationship has been used to partition the non-linear Newton solver into a non-linear global step and a linear local step.

The loop-based method is a solution method which attempts to reduce the size of the simulation problem. The oldest loop-based method (and the oldest method overall) is the Hardy Cross method (Cross 1936). In the Hardy Cross method, there are two sets of equations –(i) mass conservation equations and (ii) loop energy conservation equations– which are used to model the underlying relationship of the flows and heads of a WDS. This non-linear system of equations is solved by successive approximation, in which a set of initial flows that satisfies continuity is successively corrected until a predefined stopping test has been met. The Hardy Cross method is an iterative manual method that was popular for its simplicity before the introduction of computers. Epp and Fowler (1970) also explored the possibility of using a loop formulation to perform hydraulic simulations. They proposed a programmable version of the Hardy Cross method. However, this method is not widely used because it required (1) the identification of loops, (2) the use of pseudo-loops if the network has more than one source, and (3) the finding of a set of initial flows that satisfies continuity. Later, Creaco and Franchini (2013) incorporated the concept of minimum cycle basis to identify a set of loops that can be used to achieve the sparsest key matrix for loop formulation. It is reported in their paper that, although the loop method requires less computation time than the GGA, the time taken for identifying the minimum cycle basis can be a major disadvantage. More recently, Alvarruiz et al. (2015) presented two methods to identify the minimum cycle basis that used significantly less time.

The null space method is a special loop-based method: all null space formulations can be rewritten as loop-based formulations, but not all loop-based formulations can be rewritten as null space formulations. The co-tree flows method (CTM) is the first null space method, which partitions the network component into a spanning tree and a co-tree. The CTM has the same disadvantages as the loop flow correction method. Later, the reformulated co-tree flow method (RCTM) was introduced by Elhay et al. (2014) to address the initialization requirements by incorporating Schilders' factorizations (Schilders 2009). Abraham and Stoianov (2015) proposed a partial update method for the



(a) Example A **(b)** Example B

Fig. 1. Two example networks of blocks, bridges, and cut-vertices

null space methods, that is also applicable to the GGA, in which computation time is saved through minimizing the calculation of the head loss component by only calculating the friction factors and the head loss components of the pipes that have not satisfied the stopping test.

In this paper, a new graph partitioning algorithm, referred to as the bridge-block partitioning algorithm (BBPA), is proposed. The BBPA begins by using the FCPA to separate the core component from the forest component. Then, the BBPA further partitions the core component of the network into block and bridge components. Bridge components can be defined as the pipes in the core that are not part of any loop. For example, in Fig. 1(a) the bridge pipes are highlighted in bold. The solutions for these bridge components (block 1, block 3, and block 4) can be found by a linear process – as can the forest component in the FCPA. The remainder of the network consists of blocks, labeled **block 2** and **block 5** and solutions for these components can be found separately. It is possible to separate two blocks with a single node called a cut-vertex. This scenario is illustrated in Fig. 1(b). The node (cut-vertex 2) is a cut-vertex that separates the two blocks. These two blocks can also also be solved separately, as was the case in part (a) of the example. The advantages in speed and reliability for the BBPA arise, in part, from the smaller systems that result from partitioning the network into these smaller blocks if the core component of the WDS graph is one-connected.

The BBPA exploits the fact the flows and heads in one block component are weakly coupled with

these of the other block components and the solution of the flows and heads in a bridge component is a linear process. The convergence rate for the solution of the core component of a WDS, without the BBPA, is restricted to that of the worst block of the network. Solving each block separately reduces the number of iterations executed to the number of iterations required by that block.

There is a number of advantages to using the BBPA to identify the linear bridge components and the block components of a WDS network:

1. The number of iterations required by each block is bounded by that required by the unpartitioned system – solving the flows and heads in each block separately significantly reduces the overall computational time for the non-linear solver in almost all cases.

2. It improves the numerical reliability of the solution. The numerical reliability of the solution can be determined by the condition number of the Schur complement. The condition number of a matrix is the ratio of the largest to the smallest singular value of any square matrix. A rough rule of thumb is: one digit of reliability in the solution is lost for every power of ten in the condition number. If a square matrix is partitioned into block diagonal form by orthogonal permutations, the condition numbers of blocks can be no greater than that of the full matrix. In most cases, the condition numbers for all the individual blocks will be smaller than the condition number of the full matrix. This phenomenon is illustrated later in this paper.

3. It reduces the need to regularize for the presence of zero flows (Elhay and Simpson 2011). It has been pointed out by Simpson et al. (2012) that solving for the flows and heads separately can avoid the numerical failure that occurs when there are nodes with zero demand present in the forest. It is shown in this paper that there are blocks, in some networks, that have zero accumulative demands. The solutions of these networks need a regularization method to deal with the presence of the zero flows to avoid catastrophic numerical failure when the Hazen-William head loss model is used. Using the BBPA avoids this failure which reduces the need for regularization.

4. It reduces the computational time in a management setting because the flows in the blocks with unchanged nodal demands do not need to be solved again and the heads in the corresponding block only need to be adjusted a posteriori.

5. The solution of each block can be found in parallel in a demand-driven model because the flows and heads in one block component can be found separately from those of the other block components.

The main contributions of this paper are: (1) to extend the concept of using bridge and block components in the loop flow correction method, proposed in Deuerlein (2008), to a generalized graph partitioning algorithm that can be used with any demand-driven WDS solution method, (2) to establish the theoretical advantages of using the BBPA in terms of reducing computational load and improving numerical reliability, (3) to provide a detailed case study to demonstrate BBPA's usefulness in terms of performance and accuracy.

This paper is organized as follows. Some definitions and notations are given in the next section. The section following provides the derivation of the method with some examples. The algorithmic description of the BBPA is then given, followed by the a discussion of the relation of the BBPA and other methods. This is followed by a benchmark analysis of the BBPA applied to the eight case study networks that supports the claim about the advantages of using the BBPA. These results are then discussed in the section that follows. Finally, the last section summarizes the overall findings.

GENERAL WDS DEMAND-DRIVEN STEADY-STATE PROBLEM

This section describes the general WDS demand-driven steady-state problem. The following starts with the basic definition and notations, followed by the system equations. Finally, the Global Gradient Algorithm, which is used as the hydraulic solver to separately solve each block, are shown.

Definitions and Notation

Consider a water distribution system that contains n_p pipes, n_j junctions, n_r fixed head nodes and n_f forest pipes and nodes. The $j-th$ pipe of the network can be characterized by its diameter

D_j, length L_j, resistance factor r_j. The $i-th$ node of the network has two properties: its nodal demand d_i and its elevation head z_i.

Let $\boldsymbol{q} = \left(q_1, q_2, ...q_{n_p}\right)^T$ denote the vector of unknown flows, $\boldsymbol{h} = \left(h_1, h_2, ...h_{n_j}\right)^T$ denote the vector of unknown heads, $\boldsymbol{r} = \left(r_1, r_2, ...r_{n_p}\right)^T$ denote the vector of resistance factors, $\boldsymbol{d} = \left(d_1, d_2, ...d_{n_j}\right)^T$ denote the vector of nodal demands, $\boldsymbol{e_l} = \left(e_{l_1}, e_{l_2}....e_{l_{n_r}}\right)^T$ denote the vector of fixed head elevations.

The head loss exponent n is assumed to be dependent only on the head loss model: $n = 2$ for the Darcy-Weisbach head loss model and $n = 1.852$ for Hazen-Williams head loss model. The head loss within the pipe j, which connects the node i and the node k, is modelled by $h_i - h_k = r_j q_j |q_j|^{n-1}$. Denote by $\boldsymbol{G}(\boldsymbol{q}) \in \mathbb{R}^{n_p \times n_p}$, a diagonal square matrix with elements $[\boldsymbol{G}]_{jj} = r_j |q_j|^{n-1}$ for $j = 1, 2, ...n_p$. Denote by $\boldsymbol{F}(\boldsymbol{q}) \in \mathbb{R}^{n_p \times n_p}$, a diagonal square matrix where the j-th element on its diagonal $[\boldsymbol{F}]_{jj} = \frac{d}{dq_j}[\boldsymbol{G}]_{jj} q_j$. The matrix $\boldsymbol{A_1}$ is the full rank, unknown head, node-arc incidence matrix. The matrix $\boldsymbol{A_2}$ is the fixed-head node-arc incidence matrix.

System of Equations

The steady-state flows and heads in a WDS system are modeled by the demand-driven model (DDM) continuity equations (1) and the energy conservation equations (2):

$$-\boldsymbol{A_1}^T \boldsymbol{q} - \boldsymbol{d} = \boldsymbol{O} \tag{1}$$

$$\boldsymbol{G}(\boldsymbol{q})\,\boldsymbol{q} - \boldsymbol{A_1}\boldsymbol{h} - \boldsymbol{A_2}\boldsymbol{e_l} = \boldsymbol{O}, \tag{2}$$

which can be expressed as

$$\begin{pmatrix} \boldsymbol{G}(\boldsymbol{q}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{h} \end{pmatrix} - \begin{pmatrix} \boldsymbol{A_2}\boldsymbol{e_l} \\ \boldsymbol{d} \end{pmatrix} = 0, \tag{3}$$

where its Jacobian matrix used in the solution process is

$$\boldsymbol{J} = \begin{pmatrix} \boldsymbol{F}(\boldsymbol{q}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \tag{4}$$

and it is sometimes referred to as a nonlinear saddle point problem (Benzi et al. 2005).

This non-linear system is often solved by the Newton method, in which $\boldsymbol{q}^{(m+1)}$ and $\boldsymbol{h}^{(m+1)}$ are repeatedly computed from $\boldsymbol{q}^{(m)}$ and $\boldsymbol{h}^{(m)}$ by

$$\begin{pmatrix} \boldsymbol{F}^{(m)}(\boldsymbol{q}^{(m)}) & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}^{(m+1)} - \boldsymbol{q}^{(m)} \\ \boldsymbol{h}^{(m+1)} - \boldsymbol{h}^{(m)} \end{pmatrix} = - \begin{pmatrix} \boldsymbol{G}^{(m)}\boldsymbol{q}^{(m)} - \boldsymbol{A_1}\boldsymbol{h}^{(m)} - \boldsymbol{A_2}\boldsymbol{e_l} \\ -\boldsymbol{A_1}^T\boldsymbol{q}^{(m)} - \boldsymbol{d}, \end{pmatrix} \tag{5}$$

until the relative differences $\frac{||\boldsymbol{q}^{(m+1)}-\boldsymbol{q}^{(m)}||}{||\boldsymbol{q}^{(m+1)}||}$ and $\frac{||\boldsymbol{h}^{(m+1)}-\boldsymbol{h}^{(m)}||}{||\boldsymbol{h}^{(m+1)}||}$ are sufficiently small.

Global Gradient Algorithm

Todini and Pilati (1988) applied block elimination to Eq. (5) to yield a two-step Hazen-Williams only solver: Eq. (6) for the heads and Eq. (7) for the flows.

$$\boldsymbol{h}^{(m+1)} = \boldsymbol{U}^{-1}\left\{-n\boldsymbol{d} + \boldsymbol{A_1}^T[(1-n)\boldsymbol{q}^{(m)} - \boldsymbol{G}^{-1}\boldsymbol{A_2}\boldsymbol{e_l}]\right\} \tag{6}$$

where $\boldsymbol{U} = \boldsymbol{A_1}^T\boldsymbol{G}^{-1}\boldsymbol{A_1}$ is the Schur complement, and

$$\boldsymbol{q}^{(m+1)} = \frac{1}{n}\left\{(n-1)\boldsymbol{q}^{(m)} + \boldsymbol{G}^{-1}(\boldsymbol{A_2}\boldsymbol{e_l} + \boldsymbol{A_1}\boldsymbol{h})\right\} \tag{7}$$

Later, Simpson and Elhay (2010) proposed

$$\boldsymbol{V}\boldsymbol{h}^{(m+1)} = -\boldsymbol{d} + \boldsymbol{A_1}^T\boldsymbol{F}^{-1}\left[(\boldsymbol{G}-\boldsymbol{F})\,\boldsymbol{q}^{(m)} - \boldsymbol{A_2}\boldsymbol{e_l}\right] \tag{8}$$

where $\boldsymbol{V} = \boldsymbol{A_1}^T\boldsymbol{F}^{-1}\boldsymbol{A_1}$ is the Schur complement, and

$$\boldsymbol{q}^{(m+1)} = \boldsymbol{q}^{(m)} + \boldsymbol{F}^{-1}\boldsymbol{A_1}\boldsymbol{h}^{(m+1)} - \boldsymbol{F}^{-1}\left[\boldsymbol{G}\boldsymbol{q}^{(m)} - \boldsymbol{A_2}\boldsymbol{e_l}\right] \tag{9}$$

as the generalized equations that can be applied when the head-loss is modeled by the Hazen-William equation or the Darcy-Weisbach equation. The correct Jacobian matrix with the formula for $\boldsymbol{F}$, when head loss is modeled by Darcy-Weisbach equation, can be found in Simpson and Elhay (2010). They showed that the use of the correct Jacobian matrix restores the quadratic rate of convergence.

DERIVATION OF THE BRIDGE-BLOCK PARTITIONING ALGORITHM

The following terminology will be used in this paper. Associated with a WDS is a graph G=(V, E), where the elements of V are the nodes (vertices) of the graph G and elements of E are the pipes (links or edges) of the graph G. Every WDS can be divided into two subgraphs: a treed subgraph (forest) $G_f = \left(V_f, E_f\right)$ and a looped subgraph (core) $G_c = (V_c, E_c)$, so that $E_f \cup E_c = E$, $E_f \cap E_c = \emptyset$, $V_f \cup V_c = V$. A cut-vertex is a node in a WDS graph, the removal of which will increase the number of connected components, and a bridge is a pipe in a WDS graph, the removal of which will separate its two end nodes. A block is a maximal connected subgraph without a cut-vertex. A WDS graph can be decomposed into a tree of blocks, cut-vertices, and bridges called a block-cut tree (Diestel 2005). A root block is a block which includes one or more water sources. Note that every water source is defined to be within the root block of its network component. That is, all water sources are in the root block of their connected component of the network. The level of block i in a rooted block-cut tree is the length of the unique path, composed of blocks, from the root block to block i. The parent of block i is the block connected to block i on the path to the root block. If block i is the parent of block j, then block j is the child of block i. A block of a graph G containing only one cut-vertex is called an end block of G. Note that any block except for the root block has a unique parent block, and any block except for an end block can have multiple child blocks.

A WDS graph can be divided into n_b subgraphs, G_{b_1}=(V_{b_1},E_{b_1}), G_{b_2}=(V_{b_2},E_{b_2}), ..., $G_{b_{n_b}}$ = ($V_{b_{n_b}}$,$E_{b_{n_b}}$). If two blocks, G_{b_i}=(E_{b_i},V_{b_i}) and G_{b_j}=(E_{b_j},V_{b_j}), are adjacent, then $E_{b_i} \cap E_{b_j} = \emptyset$ and $V_{b_i} \cap V_{b_j} = c_{ij}$ where c_{ij} is the cut-vertex that connects the parent block i and child block j. The cut-vertex, c_{ij}, in the parent block, b_i, is a cluster of the demands of this cut-vertex and all its

descendant blocks. A block except for the end block can have multiple cut-vertices behaving as clusters of demands because a parent block can have multiple child blocks. The cut-vertex, c_{ij}, in the child block, b_j, is considered as a pseudo-source. The head of the cut-vertex, c_{ij}, that is found in the parent block, b_i, is used as the elevation head of the pseudo-source for the corresponding child block. With the exception of the root block, every block has a single cut-vertex that behaves as a pseudo-source. The ancestors of a block are the blocks in the path from the root block to this block, excluding the block itself and including the root block. The descendants of block i are all the blocks that have block i as an ancestor.

The BBPA is now derived by generating two orthogonal permutation matrices and using them to manipulate the matrix $\boldsymbol{A_1}$ to find n_b unknown-head node-arc incidence matrices for each block, $\boldsymbol{B_{11}}$, $\boldsymbol{B_{22}}$, ...$\boldsymbol{B_{n_b n_b}}$, and $n_b - 1$ fixed head node-arc incidence matrices, $\boldsymbol{C_1}$, $\boldsymbol{C_2}$, ..., $\boldsymbol{C_{n_b-1}}$. Note that in the following, $\boldsymbol{B_{ij}}$, the block in the i-th block row and the j-th block column, is used to denote the fixed head node-arc incidence matrices, where the subscripts i and j are used to indicate the location of the block, row j and column i, and also to indicate a direct connection between the block i and block j.

Recall that all blocks except for the root block have exactly one cut-vertex that behaves as a pseudo-source. The terms involving these pseudo-sources are moved to the right-hand-side of the system leaving the remaining node-arc incidence matrix full rank. This is because each of the diagonal block matrices of $\boldsymbol{A_1}$, a full rank matrix, is also full rank. The permutation matrix that is used to permute the system equation, Eq. (3), is

$$\boldsymbol{P_1} = \begin{array}{c} \\ n_p \\ n_j \end{array} \begin{array}{c} \begin{array}{cc} n_p & n_j \end{array} \\ \begin{pmatrix} \boldsymbol{P} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{R} \end{pmatrix} \end{array}, \tag{10}$$

where $\boldsymbol{P} = \begin{pmatrix} \boldsymbol{P}_{e_{b_1}} & \boldsymbol{P}_{e_{b_2}} & \dots \boldsymbol{P}_{e_{n_b}} \end{pmatrix}^T \in \mathbb{Z}^{n_p \times n_p}$ is the square orthogonal permutation matrix for the pipes in each block, in which $\boldsymbol{P}_{e_{b_i}} \in \mathbb{Z}^{n_p \times n_{p_{b_i}}}$, for $i = 1, 2, ...n_b$, is the permutation

matrix that identifies the pipes in the block i as distinct from the pipes in other blocks and $\boldsymbol{R} = \begin{pmatrix} \boldsymbol{R}_{v_{b_1}} & \boldsymbol{R}_{v_{b_2}} & \dots \boldsymbol{R}_{v_{n_b}} \end{pmatrix}^T \in \mathbb{Z}^{n_j \times n_j}$ is the square orthogonal permutation matrix for the nodes in each block, in which $\boldsymbol{R}_{v_{b_i}} \in \mathbb{R}^{n_j \times n_{v_{b_i}}}$, for $i = 1, 2, ...n_b$, is the permutation matrix that identifies the nodes in the block i as distinct from the nodes in other blocks.

The permuted system of the BBPA equations is:

$$\boldsymbol{P_1} \begin{pmatrix} \boldsymbol{G} & -\boldsymbol{A_1} \\ -\boldsymbol{A_1}^T & \boldsymbol{O} \end{pmatrix} \boldsymbol{P_1}^T \boldsymbol{P_1} \begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{h} \end{pmatrix} - \boldsymbol{P_1} \begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{d} \end{pmatrix} = \boldsymbol{O} \tag{11}$$

where $\boldsymbol{a} = \boldsymbol{A_2} \boldsymbol{e_l}$. With this permutation, Eq. (3) becomes:

$$\begin{pmatrix} \boldsymbol{P G P}^T & -\boldsymbol{P A_1 R}^T \\ -\boldsymbol{R A_1}^T \boldsymbol{P}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{Pq} \\ \boldsymbol{Rh} \end{pmatrix} - \begin{pmatrix} \boldsymbol{Pa} \\ \boldsymbol{Rd} \end{pmatrix} = \boldsymbol{O} \tag{12}$$

where

$$\boldsymbol{P A_1 R}^T = \begin{pmatrix} \boldsymbol{B_{11}} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \boldsymbol{B_{21}} & \boldsymbol{B_{22}} & \dots & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{B_{n_b 1}} & \boldsymbol{B_{n_b 2}} & \dots & \boldsymbol{B_{n_b n_b}} \end{pmatrix},$$

in which all the block entries above the diagonal blocks become zero matrices because there is no pipe in a parent block that connects to any node in any of its child blocks. The block entries below the diagonal blocks, $\boldsymbol{B_{ij}}$ represent the connection between the nodes in the parent block, block j, and the pipes in the child block, block i, which are $\boldsymbol{O}$ when block j and block i are not adjacent blocks. It has been pointed out above that any block, except for the end block, can have multiple child blocks. Furthermore, any block, except for the root block, can have only one parent block. As a result, each block column can have more than two non-zero block entries (including the diagonal block in that block column) and each block row, except for the root block row, has exactly two

non-zero block entries (including the diagonal block in that block row).

$$\boldsymbol{PGP}^T = \begin{pmatrix} \boldsymbol{G}_{b_1} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{G}_{b_2} & \dots & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{G}_{b_{n_b}} \end{pmatrix}, \boldsymbol{Pq} = \begin{pmatrix} \boldsymbol{q}_{b_1} \\ \boldsymbol{q}_{b_2} \\ \vdots \\ \boldsymbol{q}_{b_{n_b}} \end{pmatrix}, \boldsymbol{Rh} = \begin{pmatrix} \boldsymbol{h}_{b_1} \\ \boldsymbol{h}_{b_2} \\ \vdots \\ \boldsymbol{h}_{b_{n_b}} \end{pmatrix}, \boldsymbol{Pa} = \begin{pmatrix} \boldsymbol{a}_{b_1} \\ \boldsymbol{a}_{b_2} \\ \vdots \\ \boldsymbol{a}_{b_{n_b}} \end{pmatrix}$$

in which any block that is not a root block becomes $\boldsymbol{O}$, and $\boldsymbol{Rd} = \begin{pmatrix} \boldsymbol{d}_{b_1}^T & \boldsymbol{d}_{b_2}^T & \dots & \boldsymbol{d}_{b_{n_b}}^T \end{pmatrix}^T$. The matrix $\boldsymbol{PA_1R}^T$ in Eq. (12) can be divided into two block matrices: a block diagonal matrix:

$$\boldsymbol{A_B} = \begin{pmatrix} \boldsymbol{B_{11}} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{B_{22}} & \dots & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{B_{n_b n_b}} \end{pmatrix}, \tag{13}$$

where each block matrix on its block diagonal represents the node-arc incidence matrix of the corresponding graph block, and a lower block triangular matrix that only has entries below its block diagonal:

$$\boldsymbol{A_C} = \begin{pmatrix} \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \boldsymbol{B_{21}} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots \\ -\boldsymbol{B_{n_b 1}} & -\boldsymbol{B_{n_b 2}} & \dots & \boldsymbol{O} \end{pmatrix}, \tag{14}$$

where each matrix block represents the connection from the cut-vertex acting as a pseudo-source to a child block (row) and the connection from the same cut-vertex acting as a cluster of demand nodes to the parent block (column). Recall that, each block row of the matrix $\boldsymbol{A_B} + \boldsymbol{A_C}$, except for the block row representing the root graph blocks, has exactly two non-zero block entries: one of two non-zero block entries is on the block diagonal of the matrix $\boldsymbol{A_B}$ and the other one of the two non-zero block entries is in the lower triangular part of the matrix $\boldsymbol{A_C}$.

Defining $\boldsymbol{G}_B = \boldsymbol{PGP}^T$, $\boldsymbol{q}_B = \boldsymbol{Pq}$, $\boldsymbol{h}_B = \boldsymbol{Rh}$, $\boldsymbol{a_B} = \boldsymbol{Pa}$, and $\boldsymbol{d}_B = \boldsymbol{Rd}$, Eq. (12) can be rewritten as

$$\begin{pmatrix} \boldsymbol{G}_B & -\boldsymbol{A_C} - \boldsymbol{A_B} \\ -\boldsymbol{A_C}^T - \boldsymbol{A_B}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_B \\ \boldsymbol{h}_B \end{pmatrix} = \begin{pmatrix} \boldsymbol{a_B} \\ \boldsymbol{d}_B \end{pmatrix}. \tag{15}$$

The matrix $\boldsymbol{A_C}$ can be moved from the left-hand-side of Eq. (15) to its right-hand-side and Eq. (15) becomes:

$$\begin{pmatrix} \boldsymbol{G}_B & -\boldsymbol{A_B} \\ -\boldsymbol{A_B}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_B \\ \boldsymbol{h}_B \end{pmatrix} = \begin{pmatrix} \boldsymbol{a_B} + \boldsymbol{A_C h}_B \\ \boldsymbol{d}_B + \boldsymbol{A_C}^T \boldsymbol{q}_B \end{pmatrix} \tag{16}$$

Defining $\widehat{\boldsymbol{a}}_{\boldsymbol{B}} = \boldsymbol{a_B} + \boldsymbol{A_C h}_B$ and $\widehat{\boldsymbol{d}}_B = \boldsymbol{d}_B + \boldsymbol{A_C}^T \boldsymbol{q}_B$, Eq. (16) expands to

$$\left(\begin{array}{cccc|cccc} \boldsymbol{G}_{b_1} & \boldsymbol{O} & \dots & \boldsymbol{O} & -\boldsymbol{B_{11}} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{G}_{b_2} & \dots & \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{B_{22}} & \dots & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{G}_{b_{n_b}} & \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{B_{n_b n_b}} \\ \hline -\boldsymbol{B_{11}}^T & \boldsymbol{O} & \dots & \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \boldsymbol{O} & -\boldsymbol{B_{22}}^T & \dots & \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots & \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{O} & \dots & -\boldsymbol{B_{n_b n_b}}^T & \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{O} \end{array}\right) \left(\begin{array}{c} \boldsymbol{q}_{b_1} \\ \boldsymbol{q}_{b_2} \\ \vdots \\ \boldsymbol{q}_{b_{n_b}} \\ \boldsymbol{h}_{b_1} \\ \boldsymbol{h}_{b_2} \\ \vdots \\ \boldsymbol{h}_{b_{n_b}} \end{array}\right) = \left(\begin{array}{c} \widehat{\boldsymbol{a}}_{\boldsymbol{b_1}} \\ \widehat{\boldsymbol{a}}_{\boldsymbol{b_2}} \\ \vdots \\ \widehat{\boldsymbol{a}}_{\boldsymbol{b_{n_b}}} \\ \widehat{\boldsymbol{d}}_{b_1} \\ \widehat{\boldsymbol{d}}_{b_2} \\ \vdots \\ \widehat{\boldsymbol{d}}_{b_{n_b}} \end{array}\right). \tag{17}$$

It is evident from the expanded system of equations, Eq. (17), of the BBPA that the amount of computation can be significantly reduced by solving each block separately. Moreover, these blocks can be solved in parallel. This is because the permuted $\boldsymbol{A_1}$ matrix, $\boldsymbol{PA_1R}^T$, can be rearranged into a block diagonal matrix, which allows the non-linear system of equations in Eq. (3) be partitioned into n_b smaller independent non-linear systems.

Update of the demands and nodal heads

The demands for each block are only required to be updated once before every evaluation and the head for each unknown-head node is only required to be updated once after the solution of each

block is found. As stated previously, each block row of the matrix $\boldsymbol{A_C}$ has only one non-zero block entry below its block diagonal. The matrix $\boldsymbol{B_{ij}}$ only has one column entry that is non-zero. This column entry is the $\boldsymbol{A_2}$ matrix for that block, which is the node-arc incidence matrix representing the connection between the pseudo-source and the pipes in the child block.

Lemma 1. *Suppose $\boldsymbol{v} \in \mathbb{R}^{n_j \times 1}$ is a column vector of all ones $\boldsymbol{A_1} \in \mathbb{R}^{n_p \times n_j}$, is an unknown-head node-arc incidence matrix and $\boldsymbol{A_2} \in \mathbb{R}^{n_p \times 1}$ is a fixed-head node-arc incidence matrix for one of the WDS's blocks that is not the root block. Then*

$$-\boldsymbol{A_1 v} = \mathbf{A_2} \tag{18}$$

Proof. Denote by p_1, a set of indices for the pipes that are not connected to a water source; by p_2, a set of indices for the pipes that are connected to a water source. Let $\boldsymbol{A_1} = \begin{pmatrix} \boldsymbol{a_1}^T & \boldsymbol{a_2}^T & \dots & \boldsymbol{a_{n_p}}^T \end{pmatrix}^T$. The i-th row of the matrix $\boldsymbol{A_1}$ has two non-zero entries, 1 and -1, and the i-th row of the matrix $\boldsymbol{A_2}$ is zero if $i \in p_1$. It is evident that the inner product of $\boldsymbol{a_i}$ and $\boldsymbol{v}$ becomes 0. The j-th row of the matrix $\boldsymbol{A_1}$ has only one entry, -1, and the j-th row of the matrix $\boldsymbol{A_2}$ has only one entry, 1, if $j \in p_2$. It is evident that the inner product of $\boldsymbol{a_j}$ and $\boldsymbol{v}^T$ is -1. End of LEMMA 1. □

The relationship shown in Eq. (18) can be used to calculate term $\boldsymbol{A_c h}_B$ in Eq. (16). The relationship between the unknown head node-arc incidence matrix, $\boldsymbol{B_{ii}}$, and the fixed head node-arc incidence matrix, $\boldsymbol{B_{ij}}$, is

$$\boldsymbol{B_{ij}} = -\boldsymbol{B_{ii} v}, \tag{19}$$

the transpose of which is $\boldsymbol{B_{ij}}^T = -\boldsymbol{v}^T \boldsymbol{B_{ii}}^T$ and multiplying both sides by $\boldsymbol{q_{B_j}}$, the flows in block j , we get $\boldsymbol{B_{ij}}^T \boldsymbol{q_{B_j}} = -\boldsymbol{v}^T \boldsymbol{B_{ii}}^T \boldsymbol{q_{B_j}}$. Therefore,

$$\boldsymbol{B_{ij}}^T \boldsymbol{q_{B_j}} = -\boldsymbol{v}^T \boldsymbol{d_{B_j}}, \tag{20}$$

which is in fact the sum of the demands in the child block to the cut-vertex in the parent block.

Eq. (20) is used repeatedly from the end block to the root block until the $\boldsymbol{A_c}^T \boldsymbol{q}_B$ in Eq. (16) has been replaced. This process is performed only once before the iterative phase.

Multiplying both sides of the Eq. (19) by the unknown head at cut-vertex c_j, $\boldsymbol{h}_{c_j}$, we get

$$\boldsymbol{B_{ij}} \boldsymbol{h}_{c_j} = -\boldsymbol{B_{ii}} \boldsymbol{v} \boldsymbol{h}_{c_j}, \tag{21}$$

which is used to move $\boldsymbol{A_c} \boldsymbol{h}_b$ from the left-hand-side of the equation to the right-hand-side of the equation so that each block can be solved in parallel. The heads need only be computed just once after the iterations for all blocks have been completed.

The properties of the system of equations after bridge-block partitioning

In the BBPA, a full WDS network is partitioned into n_b smaller independent non-linear systems by permuting the original full system of equations using two orthogonal permutations $\boldsymbol{P}$ and $\boldsymbol{R}$. One of the main contributions of this paper is to show that the use of the BBPA can significantly reduce the computational loads and improve the numerical reliability of the results.

The BBPA can be used to improve the reliability of solution of the looped component in the final WDS solution. This is because the condition number, the ratio between the largest to the smallest singular value of a matrix, can be used to estimate the loss of reliable digits in solving a linear system with that matrix. The orthogonal permutations of the BBPA shuffle the n_j singular values of the Schur Complement into their corresponding blocks. This is because pre-and-post-multiplying a matrix by orthogonal matrices preserves the singular values. The upper bound of the largest singular value of all blocks is the largest singular value of the full system and the lower bound for the smallest singular value of all blocks is the smallest singular value for the full system. Therefore, the condition number of each block at the solution is bounded above by the condition number of the full system of equations but in most cases will be smaller. Moreover, the only occasions when one of the blocks has the same condition number as the full system is where both the highest and lowest singular values are present in the *same* block. Even in this particular case the other blocks in the system will have lower condition numbers than the full system.

330 Furthermore, the use of the BBPA can minimize the need to use regularization methods for
331 handling zero-flows. In the FCPA paper (Simpson et al. 2012), the authors pointed out that it is
332 common for zero flows to occur at the ends of trees with zero demands. Similarly, it is also possible
333 for all nodes in the end blocks to have zero demands. The GGA fails catastrophically at these
334 blocks when the head loss is modelled by the Hazen-William head loss model. One side-effect of
335 identifying these end blocks with zero nodal demands is zero flows can be assigned to all pipes in
336 these blocks and the head of pseudo-source can be assigned to all nodes in these blocks. When
337 zero flows occur in other blocks, regularization is needed only for the blocks with the presence of
338 zero flows instead of the full system.

339 In addition to the improvement of the numerical reliability of the final result, the use of the BBPA
340 can significantly reduce computational loads. This reduction in computational loads is achieved
341 through: (1) the bridge component being solved by a linear process, the removal of which reduces
342 the number of non-zeroes in Schur component, (2) the probable reduction in the number iterations
343 required by each block as shown in the Appendix, and (3) the non-linear system of equations for
344 each block is independent of other blocks which allows each block to be solved in parallel.

345 BRIDGE-BLOCK PARTITIONING ALGORITHM

346 The steps of the BBPA are now described. The BBPA starts with a forest search algorithm
347 to identify the forest component as distinct from the core. This is followed by identifying all the
348 blocks and bridges in the core, and updating the demands for the cut-vertices by using Stage 1 as
349 given below, a variation of the algorithm detailed by Hopcroft and Tarjan (1973). Note that this
350 algorithm is based on the depth-first search and runs in linear time. There are two ways to solve the
351 core of the network: in parallel or serially.

352 **Parallel:** It can be more efficient to solve all the blocks in parallel when the solution of the entire
353 system is needed, such as in a design setting. After the network has been permuted, each block is
354 then individually solved by using Stage 2 in no particular order. Once the solutions for all blocks
355 are found, the heads for the core nodes are recovered by using Stage 3 from the root block to the

end blocks. Finally, the heads for the forest nodes are solved.

Stage 1: Bridge block partitioning determination & bottom-up demand adjustment

```
/* Serial determination of network block from the end blocks to the root blocks
   and bottom-up cut-vertex demand accumulation                              */
input : Adjacency List and d
output: The system of equations of all blocks
1  Procedure DFS(currentNode,d,dm)
2      visited[currentNode] = true;
3      d=d+1;
4      depth[currentNode] = d;
5      low[currentNode] = d;
6      foreach (nextNode,nextPipe)∈ adjList(currentNode) do
7          if nextNode is not a Forest node then
8              if nextNode is not visited then
9                  stack.push_back(adjList[currentNode]);
10                 parent[nextnode]=nextpipe;
11                 DFS(nextnode,d,dm);
12                 if low[currentNode]≥ depth[currentNode] then
13                     BlockSource[N_B].push_back(currentNode);
14                     do
15                         temp←stack->pop_back();
16                         if (temp.first<np) then
17                             BlockPipe[N_B].insert(temp.first);
18                         end if
19                         if (temp.second!=currentNode) then
20                             if (temp.second<nj) then
21                                 if (BlockNode[N_B].insert(temp.second).second==true) then
22                                     (*dm)[currentNode]+=(*dm)[temp.second];
23                                 end if
24                             else if (temp.second>=nj) then
25                                 BlockSource[N_B].insert(temp.second);
26                         end if
27                     while (temp.first!=nextPipe);
28                     N_B = N_B + 1;
29                 end if
30                 low[currentNode]=min(low[nextNode],low[currentNode]);
31             else if (parent[currentNode]≠nextpipe&&depth[nextnode]<depth[currentNode]) then
32                 stack.push_back(adjList[currentNode]);
33                 low[currentNode]=min(low[currentNode],depth[nextnode]);
34         end if
35     end foreach
36 Algorithm BBPA()
37     for currentNode← n_j to n_j + n_f do
38         DFS(currentNode,depth,dm);
39     end for
40     Initialize the system of equations for each block using Eq. (16);
```

Stage 2: Serial or parallel block solution

```
/* Nonlinear solution for the blocks can either be found serially or in parallel  */
```

input : The system of equations for a block
output: The solution of the flows in the input block and heads that need to be updated

```
foreach Block do
    if The size of the block =1 then
        This block is a bridge and assign the demand of the only node to the flow of the only
        pipe;
    else
        if Sum of the demands in this block=0 then
            assign the flows of the pipes=0;
            continue
        endif
        Using a WDS solution method to solve the nonlinear system for the flows and
        interim heads.;
    endif
end foreach
```

Stage 3: Top-down head correction

```
/* Top-down determinations of corrected heads from the relative heads.  Actual heads in
   any block can only be found when the flows and interim heads of its ancestor blocks
   have been found                                                                  */
```

input : The unrecovered heads of a block
input : The head of the pseudo-source from the parent block of the current block
output: The recovered heads of the input block

```
foreach Block do
    if The input block is not the root bock then
        Recover the actual heads of the input block from the interim heads by using Eq. (21).
    endif
end foreach
```

Serial: Alternatively, each of the blocks can be solved in sequence. After the network has been permuted, each block is then separately solved by using Stage 2 and Stage 3 from the root block to the end blocks. Note that it is possible to only solve for a part of the system which includes the blocks of interest and their ancestor blocks. Solving the system this way reduces the computational time in a management setting because (1) the flows in the blocks with unchanged nodal demands do not need to be solved again and the heads in the corresponding block only need to be adjusted a posteriori and (2) a different priority can be assigned to a different block which allows blocks with

different priorities to be updated in a different time interval. Finally, the heads for the forest nodes are solved.

EXAMPLE

In this section, the use of the BBPA is demonstrated by applying it to the example network shown in Fig. 2. The system of equations for each block are displayed. This network has eight pipes, six nodes with unknown heads, and one water source. The solution for this example is demonstrated below in two steps: (1) network permutation and (2) network solution.

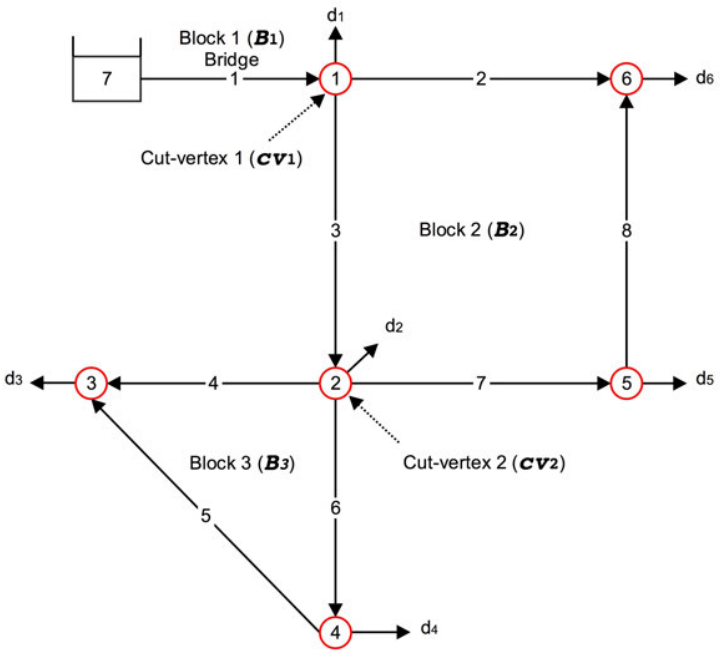


Fig. 2. A simple example network that is made up of three blocks, and two cut-vertices. Block 1 is referred to as $\boldsymbol{B_1}$, Block 2 is referred to as $\boldsymbol{B_2}$, and Block 3 is referred to as $\boldsymbol{B_3}$. Cut-vertex 1 is referred to as $\boldsymbol{cv_1}$ and Cut-vertex 2 is referred to as $\boldsymbol{cv_2}$.

Permutation for example network

The unknown-head node-arc incidence matrix, $\boldsymbol{A_1}$, and the fixed-head node-arc incidence matrix, $\boldsymbol{A_2}$ for this example network are

$$\boldsymbol{A_1} = \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix}, \boldsymbol{A_2} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

The system of pipe head loss and nodal continuity equations for the example network is

$$\begin{pmatrix} G_1 & & & & & & & & 1 & 0 & 0 & 0 & 0 & 0 \\ & G_2 & & & & & & & -1 & 0 & 0 & 0 & 0 & 1 \\ & & G_3 & & & & & & -1 & 1 & 0 & 0 & 0 & 0 \\ & & & G_4 & & & & & 0 & -1 & 1 & 0 & 0 & 0 \\ & & & & G_5 & & & & 0 & 0 & 1 & -1 & 0 & 0 \\ & & & & & G_6 & & & 0 & -1 & 0 & 1 & 0 & 0 \\ & & & & & & G_7 & & 0 & -1 & 0 & 0 & 1 & 0 \\ & & & & & & & G_8 & 0 & 0 & 0 & 0 & -1 & 1 \\ 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & & & & & & \\ 0 & 0 & 1 & -1 & 0 & -1 & -1 & 0 & & & & & & \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & & & & & & \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & & & & & & \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & & & & & & \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \\ q_7 \\ q_8 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \end{pmatrix} = \begin{pmatrix} e_{l_7} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{pmatrix}. \tag{22}$$

By permuting the rows (pipes) in the ordering given by $p = \{1; 2; 3; 7; 8; 4; 5; 6\}$ and the columns (nodes) in the ordering given by $v = \{1; 6; 2; 5; 3; 4\}$, the system of equations in Eq. (22) can be rearranged into the following block structure:

$$
\begin{array}{cc|cccccccc|cccccc|c|c}
 & & \multicolumn{8}{c|}{\textit{Pipes}} & \multicolumn{6}{c|}{\textit{Nodes}} & & \\
 & \textbf{Block} & \boldsymbol{B_1} & \multicolumn{3}{c}{\boldsymbol{B_2}} & & \multicolumn{3}{c|}{\boldsymbol{B_3}} & \boldsymbol{B_1} & \multicolumn{2}{c}{\boldsymbol{B_2}} & & \multicolumn{2}{c|}{\boldsymbol{B_3}} & & \\
\multirow{8}{*}{\textit{Pipes}} & \boldsymbol{B_1} & G_1 & & & & & & & & \mathbf{1} & 0 & \mathbf{0} & 0 & 0 & 0 & q_1 & e_{l_7} \\
\cline{3-18}
 & & & G_2 & & & & & & & \mathbf{-1} & 1 & \mathbf{0} & 0 & 0 & 0 & q_2 & 0 \\
 & & & & G_3 & & & & & & \mathbf{-1} & 0 & \mathbf{1} & 0 & 0 & 0 & q_3 & 0 \\
 & \boldsymbol{B_2} & & & & G_7 & & & & & \mathbf{0} & 0 & \mathbf{-1} & 1 & 0 & 0 & q_7 & 0 \\
 & & & & & & G_8 & & & & \mathbf{0} & 1 & \mathbf{0} & -1 & 0 & 0 & q_8 & 0 \\
\cline{3-18}
 & & & & & & & G_4 & & & \mathbf{0} & 0 & \mathbf{-1} & 0 & 1 & 0 & q_4 & 0 \\
 & \boldsymbol{B_3} & & & & & & & G_5 & & \mathbf{0} & 0 & \mathbf{0} & 0 & 1 & -1 & q_5 = & 0 \\
 & & & & & & & & & G_6 & \mathbf{0} & 0 & \mathbf{-1} & 0 & 0 & 1 & q_6 & 0 \\
\hline
\multirow{8}{*}{\textit{Nodes}} & \boldsymbol{B_1} & \mathbf{1} & \mathbf{-1} & \mathbf{-1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & & & & & & & h_1 & d_1 \\
\cline{3-18}
 & & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & & & & & & & h_6 & d_6 \\
 & \boldsymbol{B_2} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{-1} & \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{-1} & & & & & & & h_2 & d_2 \\
 & & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & & & & & & & h_5 & d_5 \\
\cline{3-18}
 & & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & & & & & & & h_3 & d_3 \\
 & \boldsymbol{B_3} & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & & & & & & & h_4 & d_4
\end{array}
\tag{23}
$$

*the bold numbers in the matrix represent the cut-vertices

Eq. (23) has three graph blocks as shown in Fig. 2 include Block 1 (a bridge), Block 2, and Block 3. Note that, for cross-referencing purposes, this equation has been labeled with the block numbers (affiliated with pipes and nodes) corresponding to each entity in the example network. The cut-vertices ($\boldsymbol{cv_1}$ and $\boldsymbol{cv_2}$ in Fig. 2) are highlighted in bold in their corresponding matrix blocks. In the equation, it is evident that the permuted $\boldsymbol{A_1}$ matrix is a block three by three, lower block triangular matrix which represents a WDS with the three graph blocks ($\boldsymbol{B_1}$, $\boldsymbol{B_2}$, and $\boldsymbol{B_3}$).

The end block ($\boldsymbol{B_3}$ in Fig. 2) is a sub-network consisting of three pipes {4; 5; 6}, two nodes {3; 4}, and a pseudo-source at node {2}. The nodal demands of this block do not need to be updated because this is the end block. The head of the node 2 ($\boldsymbol{cv_2}$) , which is the cut-vertex behaving as the pseudo-source for this block, can be moved to the right-hand-side of system of equations using Eq. (16). The solution of block $\boldsymbol{B_3}$ can be found separately after the head of the pseudo-source at node {2} is found.

The second block diagonal row ($\boldsymbol{B_2}$ in Fig. 2) is a sub-network consisting of four pipes {2; 3; 7; 8}, three nodes {2; 5; 6}, and one pseudo-source at node {1}. This is an intermediate

block so that the demand at the node 2 ($\boldsymbol{cv_2}$), a cut-vertex that is not a pseudo-source, needs to be updated by increasing its demand by the sum of demands at all nodes of its child block ($\boldsymbol{B_3}$) as follows: $\widehat{d}_2 = d_2 + d_3 + d_4$ using Eq. (20). Node 1 ($\boldsymbol{cv_1}$), which is the cut-vertex behaving as the pseudo-source for this block, $\boldsymbol{B_2}$, can be moved to the right-hand-side of system of equations using Eq. (16). The solution of block $\boldsymbol{B_2}$ can be found separately after the head of the pseudo-source at node {1} is found.

Finally, the root block ($\boldsymbol{B_1}$ in Fig. 2) is a sub-network consisting of pipe {1}, node {1}, and source {7}. Block B_1 is a bridge component. The bridge component can be solved by using a linear process. The demand for the node 1 in Fig. 2 ($\boldsymbol{cv_1}$), a cut vertex in the root block, is updated by increasing its demand by the sum of demands at all nodes of its child block ($\boldsymbol{B_2}$) as follows: $\widehat{d}_1 = d_1 + d_2 + d_3 + d_4 + d_5 + d_6$ and the elevation head for the source stays the same. After updating the demands and heads, the system of equations in Eq. (23) becomes:

$$
\begin{array}{cc|ccccccc|ccccccc|c|c}
 & & \multicolumn{8}{c|}{\textit{Pipes}} & \multicolumn{7}{c|}{\textit{Nodes}} & & \\
 & \textbf{Block} & \boldsymbol{B_1} & \multicolumn{3}{c}{\boldsymbol{B_2}} & & \multicolumn{3}{c|}{\boldsymbol{B_3}} & \boldsymbol{B_1} & \multicolumn{3}{c}{\boldsymbol{B_2}} & \multicolumn{3}{c|}{\boldsymbol{B_3}} & & \\
\multirow{7}{*}{\textit{Pipes}} & \boldsymbol{B_1} & G_1 & & & & & & & & \mathbf{1} & 0 & \mathbf{0} & 0 & 0 & 0 & q_1 & e_{l_7} \\
 & & & G_2 & & & & & & & \mathbf{0} & 1 & \mathbf{0} & 0 & 0 & 0 & q_2 & h_1 \\
 & & & & G_3 & & & & & & \mathbf{0} & 0 & \mathbf{1} & 0 & 0 & 0 & q_3 & h_1 \\
 & \boldsymbol{B_2} & & & & G_7 & & & & & \mathbf{0} & 0 & \mathbf{-1} & 1 & 0 & 0 & q_7 & 0 \\
 & & & & & & G_8 & & & & \mathbf{0} & 1 & \mathbf{0} & -1 & 0 & 0 & q_8 & 0 \\
 & & & & & & & G_4 & & & \mathbf{0} & 0 & \mathbf{0} & 0 & 1 & 0 & q_4 & h_2 \\
 & \boldsymbol{B_3} & & & & & & & G_5 & & \mathbf{0} & 0 & \mathbf{0} & 0 & 1 & -1 & q_5 & 0 \\
 & & & & & & & & & G_6 & \mathbf{0} & 0 & \mathbf{0} & 0 & 0 & 1 & q_6 & h_2 \\
\multirow{7}{*}{\textit{Nodes}} & \boldsymbol{B_1} & \mathbf{1} & \mathbf{-1} & \mathbf{-1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & & & & & & & h_1 & d_1 + d_2 + d_3 + d_4 + d_5 + d_6 \\
 & & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & & & & & & & h_6 & d_6 \\
 & \boldsymbol{B_2} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{-1} & \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{-1} & & & & & & & h_2 & d_2 + d_3 + d_4 \\
 & & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & & & & & & & h_5 & d_5 \\
 & \boldsymbol{B_3} & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & & & & & & & h_3 & d_3 \\
 & & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & & & & & & & h_4 & d_4
\end{array}
\tag{24}
$$

Note that the system of equations obtained in Eq. (24) is equivalent to performing block Gauss-Jordan elimination on Eq. (23). Solving the system of equations in this way requires solving each block in a particular sequence, from the root block ($\boldsymbol{B_1}$) to the end block ($\boldsymbol{B_3}$). The sequence that

is required in the example network in Fig. 2 is: (1) to find the solution of block $\boldsymbol{B_1}$, the root block; (2) to find the solution of block $\boldsymbol{B_2}$ using the head of the node one, $\boldsymbol{cv_1}$, in block $\boldsymbol{B_1}$; and (3) to find the solution of block $\boldsymbol{B_3}$, the end block, using the head of the node two, $\boldsymbol{cv_2}$, in block $\boldsymbol{B_2}$.

Furthermore, the second pipe head-loss block equation or the second block equation ($\boldsymbol{B_2}$) in Eq. (24) is:

$$\boldsymbol{G_{b_2} q_{b_2} - B_{22} h_{b_2} = B_{21} h_{b_1}},$$

which expands to:

$$\begin{pmatrix} G_2 & & & \\ & G_3 & & \\ & & G_7 & \\ & & & G_8 \end{pmatrix} \begin{pmatrix} q_2 \\ q_3 \\ q_7 \\ q_8 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} h_6 \\ h_2 \\ h_5 \end{pmatrix} = \begin{pmatrix} h_1 \\ h_1 \\ 0 \\ 0 \end{pmatrix}, \tag{25}$$

the right-hand-side of which can be rewritten as:

$$\boldsymbol{B_{21} h_{b_1} = -B_{22}[v_3 h_1]}, \tag{26}$$

which expands to:

$$\begin{pmatrix} h_1 \\ h_1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} h_1 \\ h_1 \\ h_1 \end{pmatrix}$$

using Eq. (21). Substituting it back into Eq. (25), we get:

$$\boldsymbol{G_{b_2} q_{b_2} - B_{22} h_{b_2} = -B_{22}[v_3 h_1]},$$

425 which expands to:

$$
\begin{pmatrix} G_2 & & & \\ & G_3 & & \\ & & G_7 & \\ & & & G_8 \end{pmatrix}
\begin{pmatrix} q_2 \\ q_3 \\ q_7 \\ q_8 \end{pmatrix}
+
\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{pmatrix}
\begin{pmatrix} h_6 \\ h_2 \\ h_5 \end{pmatrix}
=
\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{pmatrix}
\begin{pmatrix} h_1 \\ h_1 \\ h_1 \end{pmatrix},
$$

427 which can further simplified into:

$$
\boldsymbol{G_{b_2} q_{b_2}} - \boldsymbol{B_{22}}[\boldsymbol{h_{b_2}} + \boldsymbol{v}_3 h_1] = \boldsymbol{O},
$$

429 which expands to:

$$
\begin{pmatrix} G_2 & & & \\ & G_3 & & \\ & & G_7 & \\ & & & G_8 \end{pmatrix}
\begin{pmatrix} q_2 \\ q_3 \\ q_7 \\ q_8 \end{pmatrix}
+
\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{pmatrix}
\begin{pmatrix} h_6 - h_1 \\ h_2 - h_1 \\ h_5 - h_1 \end{pmatrix}
= \boldsymbol{O}.
$$

431 The third pipe head-loss block equation or the third block equation ($\boldsymbol{B_3}$) in Eq. (24) is:

$$
\boldsymbol{G_{b_3} q_{b_3}} - \boldsymbol{B_{33}} \boldsymbol{h_{b_3}} = \boldsymbol{B_{32}} \boldsymbol{h_{b_2}},
$$

433 which expands to:

$$
\begin{pmatrix} G_4 & & \\ & G_5 & \\ & & G_6 \end{pmatrix}
\begin{pmatrix} q_4 \\ q_5 \\ q_6 \end{pmatrix}
+
\begin{pmatrix} 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{pmatrix}
\begin{pmatrix} h_3 \\ h_4 \end{pmatrix}
=
\begin{pmatrix} h_2 \\ 0 \\ h_2 \end{pmatrix}. \tag{27}
$$

Eq. (27) can be further simplified to

$$\begin{pmatrix} G_4 & & \\ & G_5 & \\ & & G_6 \end{pmatrix} \begin{pmatrix} q_4 \\ q_5 \\ q_6 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} h_3 - h_2 \\ h_4 - h_2 \end{pmatrix} = \boldsymbol{O}$$

using a similar manipulation as for Block 2 above. Finally, the system of equations in Eq. (24) may be rewritten as:

		Pipes								***Nodes***								
	Block	$\boldsymbol{B_1}$	$\boldsymbol{B_2}$				$\boldsymbol{B_3}$			$\boldsymbol{B_1}$	$\boldsymbol{B_2}$			$\boldsymbol{B_3}$				
Pipes	$\boldsymbol{B_1}$	G_1								**1**	0	**0**	0	0	0	q_1	=	e_{l_7}
	$\boldsymbol{B_2}$		G_2							**0**	1	**0**	0	0	0	q_2		0
				G_3						**0**	0	**1**	0	0	0	q_3		0
					G_7					**0**	0	**−1**	1	0	0	q_7		0
						G_8				**0**	1	**0**	−1	0	0	q_8		0
	$\boldsymbol{B_3}$						G_4			**0**	0	**0**	0	1	0	q_4		0
								G_5		**0**	0	**0**	0	1	−1	q_5		0
									G_6	**0**	0	**0**	0	0	1	q_6		0
Nodes	$\boldsymbol{B_1}$	**1**	**−1**	**−1**	**0**	**0**	**0**	**0**	**0**							h_1		$d_1 + d_2 + d_3 + d_4 + d_5 + d_6$
	$\boldsymbol{B_2}$	0	1	0	0	1	0	0	0							$h_6 - h_1$		d_6
		0	**0**	**1**	**−1**	**0**	**−1**	**0**	**−1**							$h_2 - h_1$		$d_2 + d_3 + d_4$
		0	0	0	1	−1	0	0	0							$h_5 - h_1$		d_5
	$\boldsymbol{B_3}$	0	0	0	0	0	1	1	0							$h_3 - h_2$		d_3
		0	0	0	0	0	0	−1	1							$h_4 - h_2$		d_4

(28)

Solving the example network

Consider the network shown in Fig. 2 and its permuted system of equations, Eq. (28). Each block becomes an independent system and can be solved sequentially from the root block to the end block. The system of equations for the root block, $\boldsymbol{B_1}$ (Block 1 in Fig. 2), which also represents a bridge, is:

$$\begin{pmatrix} G_1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} q_1 \\ h_1 \end{pmatrix} = \begin{pmatrix} e_{l_7} \\ d_1 + d_2 + d_3 + d_4 + d_5 + d_6 \end{pmatrix}, \tag{29}$$

446 the solution of which can be used to solve its child block, block $\boldsymbol{B_2}$ (Block 2 in Fig. 2) by using:

447
$$\begin{pmatrix} G_2 & & & & 1 & 0 & 0 \\ & G_3 & & & 0 & 1 & 0 \\ & & G_7 & & 0 & -1 & 1 \\ & & & G_8 & 1 & 0 & -1 \\ 1 & 0 & 0 & 1 & & & \\ 0 & 1 & -1 & 0 & & & \\ 0 & 0 & 1 & -1 & & & \end{pmatrix} \begin{pmatrix} q_8 \\ q_7 \\ q_3 \\ q_2 \\ h_6 - h_1 \\ h_2 - h_1 \\ h_5 - h_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ d_6 \\ d_2 + d_3 + d_4 \\ d_5 \end{pmatrix}, \tag{30}$$

448 and finally, the end block, block $\boldsymbol{B_3}$ (Block 3 in Fig. 2) can be solved by using:

449
$$\begin{pmatrix} G_4 & & & 1 & 0 \\ & G_5 & & 1 & -1 \\ & & G_6 & 0 & 1 \\ 1 & 1 & 0 & & \\ 0 & -1 & 1 & & \end{pmatrix} \begin{pmatrix} q_6 \\ q_5 \\ q_3 \\ h_3 - h_2 \\ h_4 - h_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ d_3 \\ d_4 \end{pmatrix}. \tag{31}$$

450 The systems of equations for each of the three blocks can also be solved in parallel.

451 Note that, when using BBPA, if the head loss of the example network shown in Fig. 2 is modeled
452 by the Hazen-William formula and the nodal demands at nodes three and four are zero, this does
453 not cause a failure of the method due to singularity of the Schur complement, unlike the GGA
454 and RCTM on the same network (Elhay and Simpson 2011). In addition, the block with zero total
455 demand can be solved (1) prior to the iterative phase by assigning zero flows to all applicable pipes
456 and (2) by assigning the heads of the source to all nodes in this block after the iterative phase.

457 RELATION OF BBPA TO OTHER SOLUTION METHODS

458 The BBPA can be described as a pre-and-post-processing method for the following reasons: (1)
459 it finds the blocks and bridges of a WDS, (2) the bridges can be solved by using a linear process
460 similar to the forest component, and (3) then uses any WDS solution method, for example GGA,

461 RCTM, or GMPA, to, independently, solve each block.

462 The BBPA can also be used to identify the forest component of the network. However, the use
463 of the FCPA requires less overhead than the BBPA.

464 The same topological properties exploited by FCPA and BBPA are partly responsible for
465 the savings achieved by partial-update (Abraham and Stoianov 2015). The forest and bridge
466 components - being linear - converge after just one iteration of application of a non linear solver.
467 The partial update scheme is able to exploit this by checking for convergence every iteration. Once
468 the convergence test for a pipe has been met, the head-loss of the converged component does
469 not need to be re-computed, whereas the linear solver for the full system is required until the
470 convergence tests for all pipes have been met. In contrast, FCPA and BBPA have the advantage
471 of identifying these components in advance and removing them from non-linear solution process.
472 BBPA also has the additional advantage of being able to exploit earlier convergence of different
473 blocks in the core network and removing them from the problem once they have converged. As a
474 result, the authors recommend that it is inefficient to implement the partial update for a full WDS
475 system before applying the FCPA and the BBPA. The usefulness of applying the partial update to
476 each block requires further investigation.

477 CASE STUDIES

478 A comparison of the GGA with or without BBPA on eight case study networks has been carried
479 out in order to support the above discussion. Note that the first step each method is to use FCPA to
480 remove the forest component from the case study networks, to ensure a fair comparison.

481 The efficiency and reliability of the BBPA in a once-off simulation setting, in which the steady-
482 state heads and flows are computed just once with the given WDS parameters, was benchmarked
483 against an efficient GGA implementation. As a baseline, timings of the solution process for the
484 benchmark networks using EPANET2 were also recorded. The benchmark tests were performed on
485 a Intel(R) Core(TM) CPU i5-4590 running at 3.30 GHz with 4 cores in C++ under IEEE-standard
486 double precision floating point arithmetic with machine epsilon $\epsilon_{mach} = 2.22 \times 10^{-16}$. The number
487 of cores allocated to each test was limited to one. Each timing test, measuring wall-clock time, was

TABLE 1. Benchmark networks summary, their core network size, the number of blocks and the number of bridges

	Full Network			Core network		BBPA	
Network	n_p	n_j	n_s	n_{j_c}	n_{p_c}	The number of blocks	The number of bridges
N_1	934	848	8	573	487	33(1)*	118
N_2	1118	1039	2	797	718	10(2)	45
N_3	1975	1770	4	1152	947	7	6
N_4	2465	1890	3	2036	1461	47(3)	62
N_5	2509	2443	2	1087	1741	8(1)	45
N_6	8585	8392	2	6735	6542	7(2)	58
N_7	14830	12523	7	11898	9591	487(19)	895
N_8	19647	17971	15	15232	13557	17(2)	59

*numbers in the brackets refers to the number of blocks with no nodal demands

repeated 15 times on each benchmark network.

It is shown that the use of an efficiently implemented BBPA can provide a significant runtime reduction and improvement in the reliability of the solution. The BBPA with the GGA and the standalone GGA were each applied to eight case studies with between 932 and 19,651 pipes and between 848 and 17,977 nodes with no pumps and no valves.

RESULTS AND DISCUSSION

The basic details of the case study networks considered in this study are described in columns 2 to 4 in Table 1 and more information can be found in Simpson et al. (2012). The size of the core component for each of the eight case studies is shown in the columns 5 and 6, the number of blocks in column 7, with the number of blocks with no nodal demands in the brackets, and the number of bridges in column 8. Table 2 shows the detailed profile of the size of each block in each of the eight case study networks. The size of the largest block, smallest block, and median block and the number in brackets is the percentage of the corresponding block size as a proportion of the core component of the network

Table 3 shows the summary statistics of the 15 repetitions of each solution method applied to the eight benchmark networks. The GGA benefits from the use of the BBPA by between 33% and 70%.

TABLE 2. The profile of blocks in each of the eight case study networks: size of the largest, the smallest and the median blocks

	Largest size block		Smallest size block		Median size block	
Network	n_p	n_j	n_p	n_j	n_p	n_j
N_1	81(18)	62(17)	3(0.7)	2(0.5)	7(1.6)	5(1.4)
N_2	684(91.9)	615(92.2)	2(0.3)	1 (0.1)	9.5 (1.3)	8(1.2)
N_3	953 (83.1)	78(33.1)	6(0.5)	5(2.1)	31 (2.7)	25.5(10.8)
N_4	1549(78.7)	1100(78.8)	2 (0.1)	1 (0.1)	7(0.4)	5(0.4)
N_5	1061(60.3)	1026(60.5)	2(0.1)	1(0.1)	53(3.0)	52(3.1)
N_6	5578(83.7)	5418(83.7)	2(0.03)	1(0.02)	51(0.8)	50(0.8)
N_7	8418(77.00)	6970(88.52)	2(0.02)	1(0.01)	4(0.04)	1(0.01)
N_8	14961(98.78)	13309(98.79)	3(0.02)	2(0.01)	12(0.08)	11(0.08)

numbers in the brackets refers to the percentage of the corresponding block size in the core component of the network

It has been established in Elhay et al. (2014) and Abraham and Stoianov (2015) that the number of non-zeros can be used as a surrogate to approximate the runtime of the non-linear system. The saving in runtime is partially achieved through the reduction in the number of non-zeros by the removal of the bridge components.

TABLE 3. The mean time of once-off simulation runs averaged over 15 once-off simulations for each of the two solution methods applied eight case study networks (milliseconds$\pm$standard error) and the % diff. refers to relative difference compared to the GGA mean time

	EPANET	GGA with FCPA	GGA with BBPA	
	Mean time	Mean time	Mean time	%diff.
N_1	9.09	4.66 ± 0.07	2.32 ± 0.05	-50%
N_2	16.75	6.61 ± 0.07	2.86 ± 0.04	-56%
N_3	21.46	8.72 ± 0.09	3.64 ± 0.05	-58%
N_4	26.45	22.76 ± 0.53	6.64 ± 0.11	-70%
N_5	28.46	12.19 ± 0.13	5.97 ± 0.12	-51%
N_6	172.84	44.79 ± 0.18	28.53 ± 0.12	-36%
N_7	307.17	63.06 ± 0.65	42.35 ± 0.67	-33%
N_8	600.08	131.82 ± 3.99	59.08 ± 0.6	-55%

Another important factor of the algorithm efficiency is the number of iterations required to satisfy the stopping test. Fig. (3) shows that the number of iterations required by each block of network N_1 and the number of iteration required by the full system to satisfy the stopping test. The

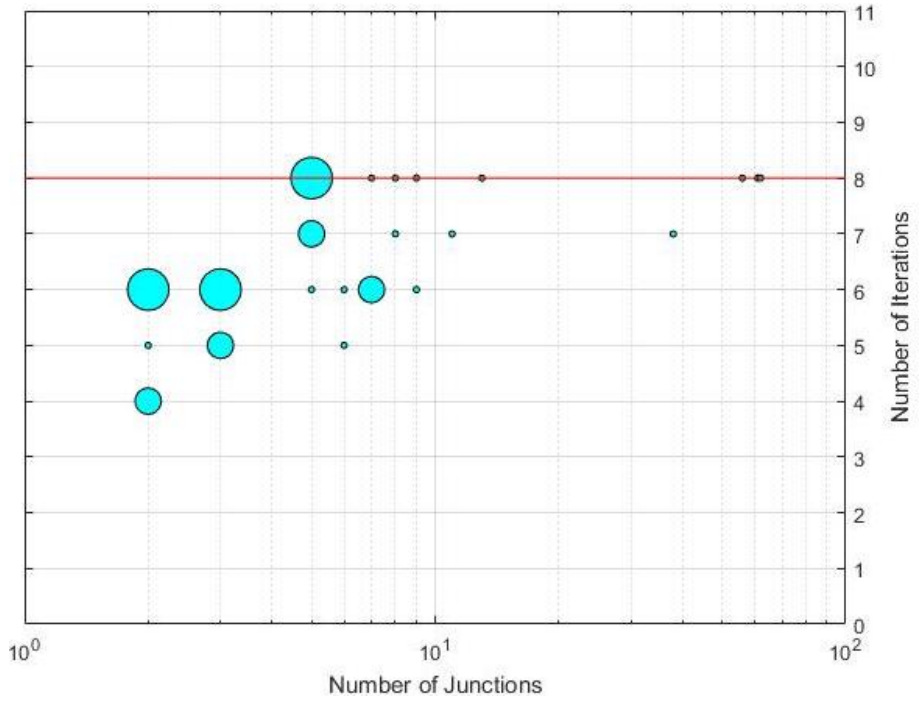


Fig. 3. The number of iterations for each block of networks N_1 against the number of junctions (the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test)

horizontal axis shows the number of junctions and the vertical axis shows the number of iterations, and the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test. For example, in network N_1 there are six blocks that have two nodes, three of which require six iterations to satisfy the stopping test; one of which requires five iterations to satisfy the stopping test; and four of which require four iterations to satisfy the stopping test. The number of iterations that is required by each block of N_1 is bounded above by that which is required by the full network of N_1. The bubble plots for networks N_2 to N_8 can be found in the supplemental data.

On another note, the BBPA can also be used to improved reliability of the solution. Fig. (4) shows that the condition number at the solution and the condition number for the full system. The following observation can be made from the Fig. (4) that (i) the condition number for each block is bounded above by the condition number for the full matrix, (ii) each of networks N_2, N_5, N_6, and N_8 has one block with the same condition number as the full system.

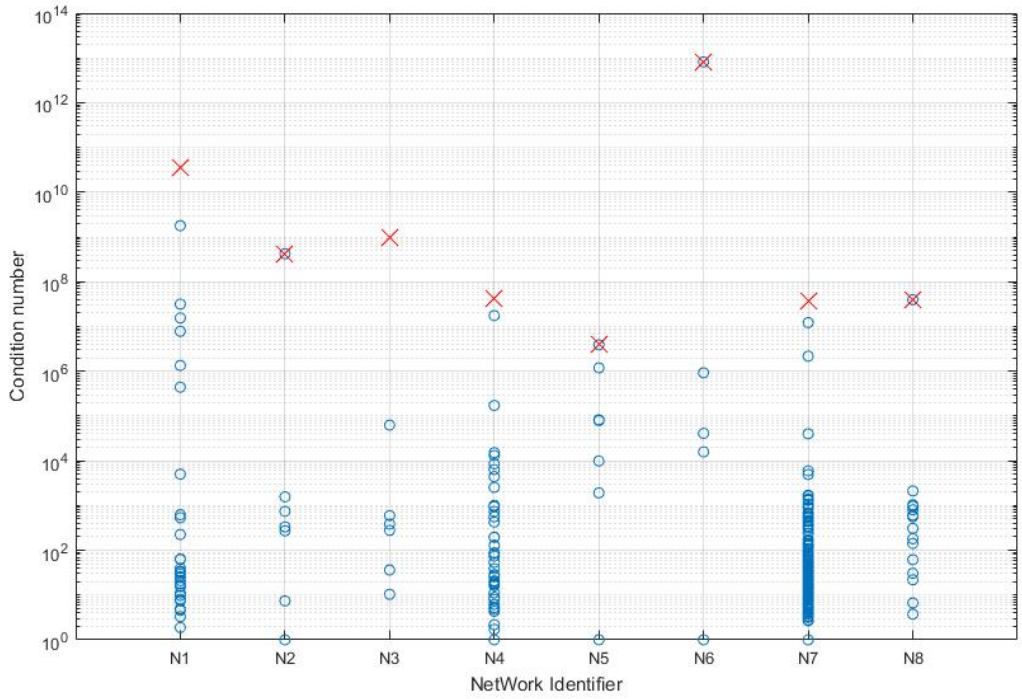


Fig. 4. The condition number of the Schur complement at the solution for each block (scatter point) and the condition number of the Schur complement for the full system (red line)

CONCLUSIONS

In this paper, the bridge-block partitioning algorithm is introduced. The BBPA is a pre-processing and post-processing algorithm that (1) first partitions the network into bridge components and block components, (2) then solves for the flows in the bridge components by a linear process, (3) after that it separately solves for the flows and the estimated heads for each independent block by using any WDS solver, and (4) finally the heads are recovered by a linear process at the end. This partitioning of the network can be used to speed-up the solution process of the steady state demand-driven hydraulic simulation and to improve the reliability of the results if the core component of the WDS graph is one-connected. The speed-up of the solution process is achieved by (1) solving the bridge component in the BBPA by a linear process similar to that of solving for the forest in the FCPA, which reduces the number of non-zeroes in the Schur complement (2) solving each block by using the minimum number of iterations that is required by that block. Moreover, the BBPA

improves the reliability of the results because the condition number of the Schur Complement for each block is bounded above by the condition number for the Schur Complement of the full system.

The usefulness of the BBPA has also been demonstrated by applying it to eight benchmark networks with between 934 and 19,647 pipes and between 848 and 17,971 nodes. The total savings in wall clock time after applying the BBPA to the GGA are between 33% and 70%. It is shown that, the number of iterations and the condition number required by each block are bounded by the number of iterations and the condition number required by the full system, respectively. The use of the BBPA can also minimize the need to regularize the zero flows when the head loss is modelled by the Hazen-William head loss equation. This is because in real life systems, such as the case study networks used in this paper, can have blocks, such that the nodes in these block all have zero demands, which can be handled by use of BBPA. Moreover, when regularization is needed, it is only required to be applied at the corresponding block instead of the full system of equations.

REFERENCES

Abraham, E. and Stoianov, I. (2015). "Sparse null space algorithms for hydraulic analysis of large-scale water supply networks." *Journal of Hydraulic Engineering*, 04015058.

Alvarruiz, F., Martínez-Alzamora, F., and Vidal, A. (2015). "Improving the efficiency of the loop method for the simulation of water distribution systems." *Journal of Water Resources Planning and Management*, 141(10), 04015019.

Benzi, M., Golub, G., and Liesen, J. (2005). "Numerical solution of saddle point problems." *Acta. Num.*, 1–137.

Creaco, E. and Franchini, M. (2013). "Comparison of newton-raphson global and loop algorithms for water distribution network resolution." *Journal of Hydraulic Engineering*, 140(3), 313–321.

Cross, H. (1936). "Analysis of flow in networks of conduits or conductors." *University of Illinois. Engineering Experiment Station. Bulletin; no. 286.*

Deuerlein, J., Elhay, S., and Simpson, A. R. (2015). "Fast graph matrix partitioning algorithm for solving the water distribution system equations." *Journal of Water Resources Planning and Management*, 142(1), 04015037–1–04015037–11.

563 Deuerlein, J. W. (2008). "Decomposition model of a general water supply network graph." *Journal*
564 *of Hydraulic Engineering*, 134(6), 822–832.
565 Diestel, R. (2005). "Graph theory." *Graduate Texts in Mathematics*, 173.
566 Elhay, S. and Simpson, A. R. (2011). "Dealing with zero flows in solving the nonlinear equations
567 for water distribution systems." *Journal of Hydraulic Engineering*, 137(10), 1216–1224.
568 Elhay, S., Simpson, A. R., Deuerlein, J., Alexander, B., and Schilders, W. H. (2014). "Reformulated
569 co-tree flows method competitive with the global gradient algorithm for solving water distribution
570 system equations." *Journal of Water Resources Planning and Management*, 140(12), 04014040–
571 1–04014040–10.
572 Epp, R. and Fowler, A. G. (1970). "Efficient code for steady-state flows in networks." *Journal of*
573 *the Hydraulics Division*, 96(1), 43–56.
574 Hopcroft, J. and Tarjan, R. (1973). "Algorithm 447: Efficient algorithms for graph manipulation."
575 *Commun. ACM*, 16(6), 372–378.
576 Schilders, W. H. (2009). "Solution of indefinite linear systems using an lq decomposition for the
577 linear constraints." *Linear Algebra and its Applications*, 431(3), 381–395.
578 Simpson, A. R. and Elhay, S. (2010). "Jacobian matrix for solving water distribution system
579 equations with the darcy-weisbach head-loss model." *Journal of Hydraulic Engineering*, 137(6),
580 696–700.
581 Simpson, A. R., Elhay, S., and Alexander, B. (2012). "Forest-core partitioning algorithm for
582 speeding up analysis of water distribution systems." *Journal of Water Resources Planning and*
583 *Management*, 140(4), 435–443.
584 Todini, E. and Pilati, S. (1988). "A gradient algorithm for the analysis of pipe networks." *Computer*
585 *applications in water supply: vol. 1—systems analysis and simulation*, Research Studies Press
586 Ltd., 1–20.

587 APPENDIX: WHY THE NUMBER OF ITERATIONS REQUIRED BY EACH 588 BLOCK IS BOUNDED ABOVE BY THAT OF THE FULL SYSTEM

589 The BBPA is derived to partition the WDS network into a number of blocks to improve the

efficiency and reliability of the WDS solution process. The number of iterations that is required by each block is bounded above by the number of iterations that is required by the full system. The permuted system of equations shown in Eq. (15) can be rewritten as

$$\begin{pmatrix} \boldsymbol{F}_B^{(m)} & -\boldsymbol{A_B} \\ -\boldsymbol{A_B}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_B^{(m+1)} \\ \boldsymbol{h}_B^{(m+1)} \end{pmatrix} - \begin{pmatrix} 0 & \boldsymbol{A_C} \\ \boldsymbol{A_C}^T & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_B^{(m+1)} \\ \boldsymbol{h}_B^{(m+1)} \end{pmatrix} = - \begin{pmatrix} (\boldsymbol{G_B^{(m)}} - \boldsymbol{F_B^{(m)}})\boldsymbol{q_B^{(m)}} - \boldsymbol{a}_B \\ -\boldsymbol{d}_B. \end{pmatrix}. \quad (32)$$

Note that each block row of the matrix $\boldsymbol{A_C}$, that represents a root block, is entirely zero. As a result, the system of equations for the root block B_i is

$$\begin{pmatrix} \boldsymbol{F}_{b_i}^{(m)} & -\boldsymbol{B_{ii}} \\ -\boldsymbol{B_{ii}}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_{b_i}^{(m+1)} \\ \boldsymbol{h}_{b_i}^{(m+1)} \end{pmatrix} = - \begin{pmatrix} (\boldsymbol{G_{b_i}^{(m)}} - \boldsymbol{F_{b_i}^{(m)}})\boldsymbol{q_{b_i}^{(m)}} - \boldsymbol{a_{b_i}} \\ -\widehat{\boldsymbol{d}}_{b_i}. \end{pmatrix}. \quad (33)$$

Also note that each block row of the matrix $\boldsymbol{A_C}$, that does not represent a root block, has exactly one non-zero block and each of these blocks has exactly one non-zero column. As a result, the system of equations for a block, B_j, that is not a root block is

$$\begin{pmatrix} \boldsymbol{F}_{b_j}^{(m)} & -\boldsymbol{B_{kj}} & -\boldsymbol{B_{jj}} \\ -\boldsymbol{B_{jj}}^T & \boldsymbol{O} & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_{b_j}^{(m+1)} \\ \boldsymbol{h}_{b_k}^{(m+1)} \\ \boldsymbol{h}_{b_j}^{(m+1)} \end{pmatrix} = - \begin{pmatrix} (\boldsymbol{G_{b_j}^{(m)}} - \boldsymbol{F_{b_j}^{(m)}})\boldsymbol{q_{b_j}^{(m)}} - \boldsymbol{a_{b_j}} \\ -\widehat{\boldsymbol{d}}_{b_j} \end{pmatrix}, \quad (34)$$

where the non-zero block row entry at block row j, $\boldsymbol{B_{kj}} = \boldsymbol{P}_{e_{b_j}} \boldsymbol{A_1} \boldsymbol{R}_{v_{b_k}}^T \in \mathbb{R}^{n_{p_{b_j}} \times n_{j_{b_j}}}$, which represents the connection between the current block j and its parent block k, has only one non-zero column entry, $\boldsymbol{A}_{2_{b_j}} = \boldsymbol{P}_{e_{b_j}} \boldsymbol{A_1} \boldsymbol{R}_{cv_{b_k}}^T \in \mathbb{R}^{n_{p_{b_j}} \times 1}$. This non-zero column entry is the unknown-head node-arc incidence matrix for block b_k and cv_{b_k} is the cut-vertex that behaves as the pseudo-source in block b_k. We can write the term $\boldsymbol{B_{kj}} \boldsymbol{h}_{b_k}^{(m+1)}$ as $[\boldsymbol{P}_{e_{b_j}} \boldsymbol{A_1} \boldsymbol{R}_{cv_{b_k}}^T][\boldsymbol{R}_{cv_{b_k}} \boldsymbol{h}]$, which is $\boldsymbol{A}_{2_{b_j}} \boldsymbol{h}_{cv_{b_j}}$ (see (Eq. (25) and Eq. (27)).

In addition, the combination of matrices $\boldsymbol{B_{jj}}$ and $\boldsymbol{A}_{2_{b_j}}$ is the Laplacian matrix of the graph of block B_j. Every row of a Laplacian matrix has exactly two non-zero entries: 1 and -1. Therefore, $\boldsymbol{B_{jj}} \boldsymbol{v_{n_{j_{b_j}}}} + \boldsymbol{A}_{2_{b_j}} \boldsymbol{v_{n_{f_{b_j}}}} = \boldsymbol{o}$. We also know that $n_{f_{b_j}} = 1$ which is equivalent to $\boldsymbol{B_{jj}} \boldsymbol{v_{n_{j_{b_j}}}} = -\boldsymbol{A}_{2_{b_j}}$

as shown in Lemma 1.

Thus, the left-hand-side of the first block equation of Eq. (34) is:

$$\boldsymbol{F}_{b_j}^{(m)}\boldsymbol{q}_{b_j}^{(m+1)} - \boldsymbol{B_{kj}}\boldsymbol{h}_{b_k}^{(m+1)} - \boldsymbol{B_{jj}}\boldsymbol{h}_{b_j}^{(m+1)}$$

and can be rewritten as

$$\boldsymbol{F}_{b_j}^{(m)}\boldsymbol{q}_{b_j}^{(m+1)} - \boldsymbol{B_{jj}}\boldsymbol{v}_{\boldsymbol{n}_{\boldsymbol{j}\boldsymbol{b}_j}}\boldsymbol{h}_{b_k}^{(m+1)} - \boldsymbol{B_{jj}}\boldsymbol{h}_{b_j}^{(m+1)}$$

and finally, denoting $\widehat{\boldsymbol{q}}^{(m+1)} = \boldsymbol{q}_{b_j}^{(m+1)}$ and $\widehat{\boldsymbol{h}}^{(m+1)} = \boldsymbol{h}_{b_j}^{(m+1)} + \boldsymbol{v}_{\boldsymbol{n}_{\boldsymbol{j}\boldsymbol{b}_j}}\boldsymbol{h}_{b_k}^{(m+1)}$, gives

$$\begin{pmatrix} \boldsymbol{F}_{b_j}^{(m)} & -\boldsymbol{B_{jj}} \\ -\boldsymbol{B_{jj}}^T & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \widehat{\boldsymbol{q}}^{(m+1)} \\ \widehat{\boldsymbol{h}}^{(m+1)} \end{pmatrix} = - \begin{pmatrix} (\boldsymbol{G}_{\boldsymbol{b_j}}^{(\boldsymbol{m})} - \boldsymbol{F}_{\boldsymbol{b_j}}^{(\boldsymbol{m})})\boldsymbol{q}_{\boldsymbol{b_j}}^{(\boldsymbol{m})} - \boldsymbol{a}_{\boldsymbol{b_j}} \\ -\widehat{\boldsymbol{d}}_{b_j} \end{pmatrix}. \tag{35}$$

The matrices on the left-hand-side of Eq. (33) and Eq. (35) are identical and invertible and the right-hand-side of both equations are also identical. Therefore

$$\begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{h} \end{pmatrix} = \begin{pmatrix} \widehat{\boldsymbol{q}} \\ \widehat{\boldsymbol{h}} \end{pmatrix}$$

The Newton equation shown in Eq. (32), which is the GGA solution of an orthogonal permutation of the original system of equations, has the same flows and heads iterates as the GGA solution of Eq. (5). Moreover, it is shown above that the Newton equation in Eq. (33) has the same flow and head iterates as the Newton equation in Eq. (32). At the same time, the Newton equation in Eq. (35) has the same flow iterates as the Newton equation in Eq. (32) and the actual heads can be recovered from the interim heads a posteriori. Thus, solving each block individually produces the same flow iterates as solving the unpartitioned WDS network. The number of iterations for each block to satisfy the stopping test, $\frac{||\boldsymbol{q}^{(m+1)}-\boldsymbol{q}^{(m)}||_\infty}{||\boldsymbol{q}^{(m+1)}||_\infty}$, is bounded above by the number of iterations required by the whole system.

SUPPLEMENTARY DATA

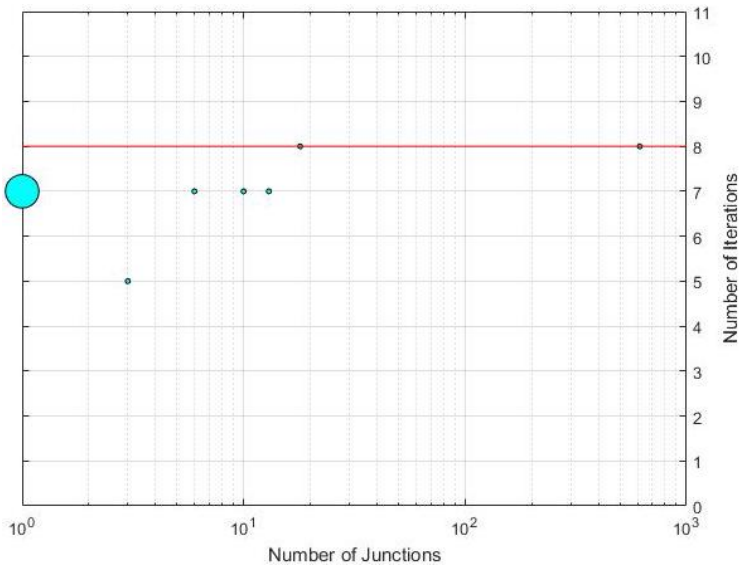


Fig. 5. The number of iterations for each block of network N_2 against the number of junctions (the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test)

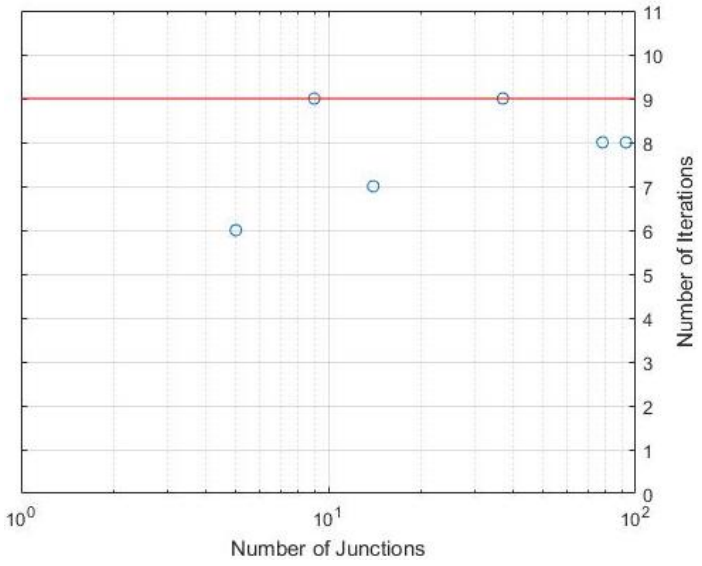


Fig. 6. The number of iterations for each block of network N_3 against the number of junctions (the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test)

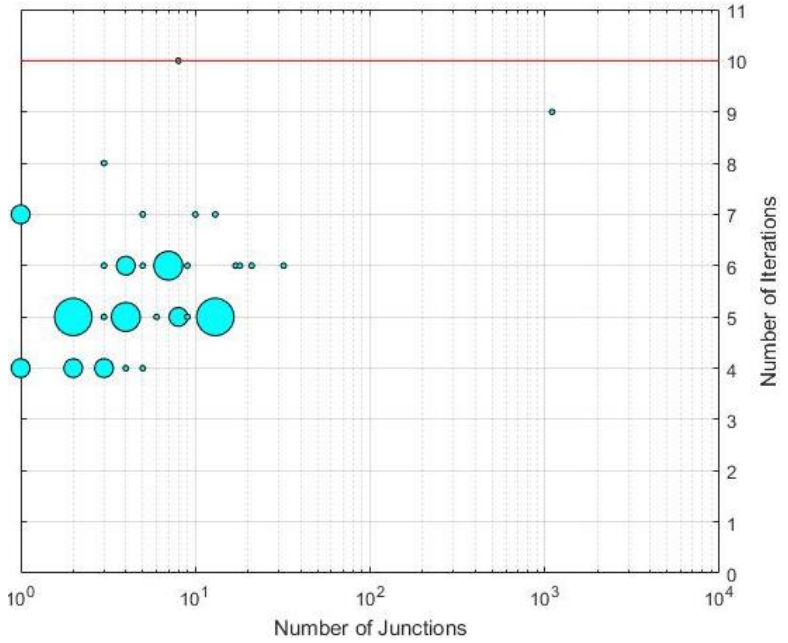


Fig. 7. The number of iterations for each block of network N_4 against the number of junctions (the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test)

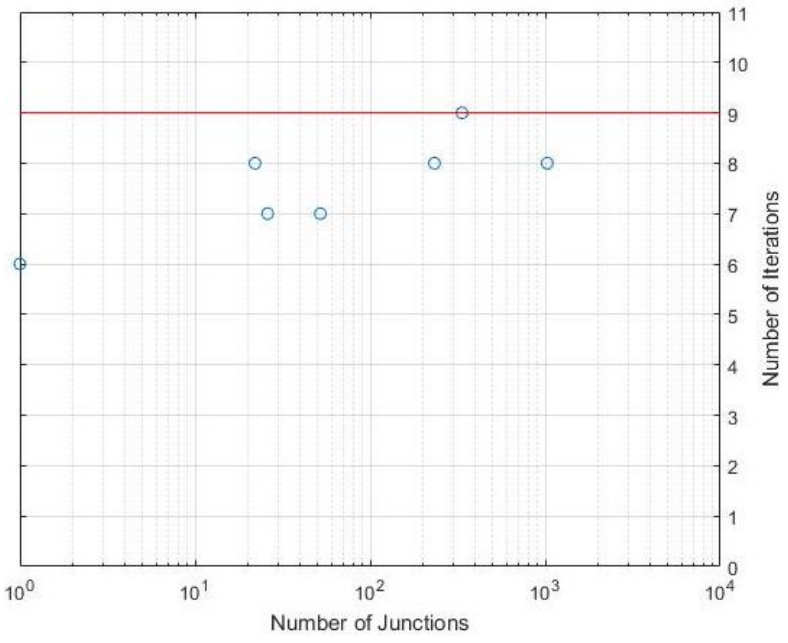


Fig. 8. The number of iterations for each block of network N_5 against the number of junctions (the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test)

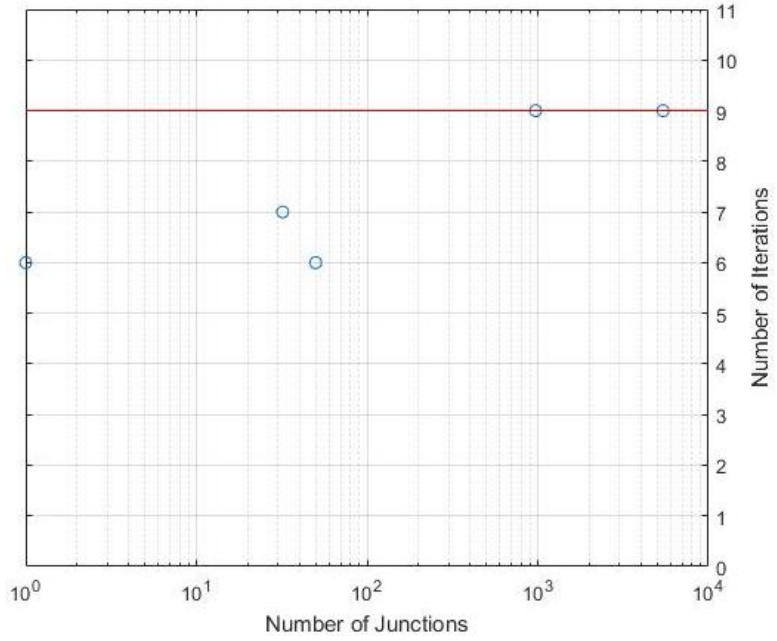


Fig. 9. The number of iterations for each block of network N_6 against the number of junctions (the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test)

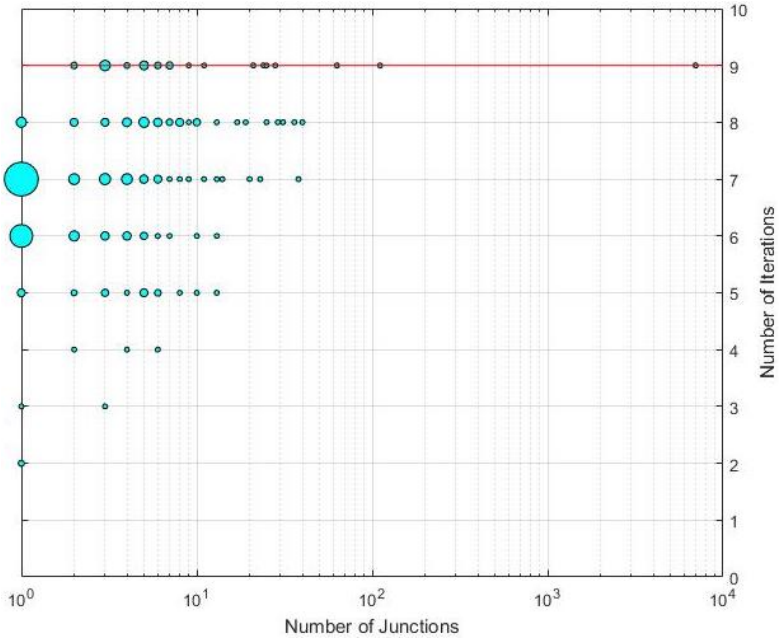


Fig. 10. The number of iterations for each block of network N_7 against the number of junctions (the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test)

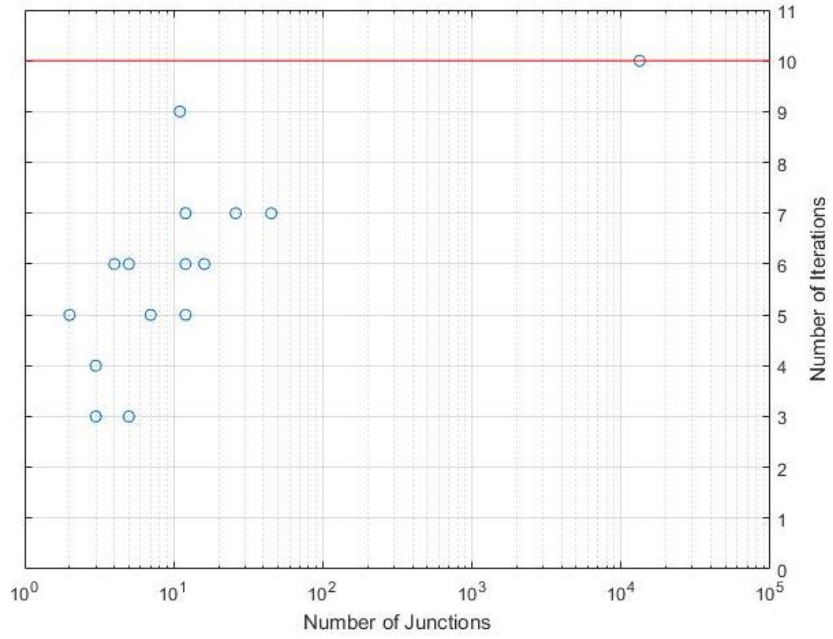


Fig. 11. The number of iterations for each block of network N_8 against the number of junctions (the diameter of the bubble represents the number of blocks with the same number of junctions which required the same number of iterations to satisfy the stopping test)