Speeding up the water distribution network design optimization using the ΔQ method

Damjan Ivetić, Željko Vasilić, Miloš Stanić and Dušan Prodanović

ABSTRACT

To optimize the design of a water distribution network (WDN), a large number of possible solutions need to be examined; hence computation efficiency is an important issue. To accelerate the computation, one can use more powerful computers, parallel computing systems with adapted hydraulic solvers, hybrid algorithms, more efficient hydraulic methods or any combination of these techniques. This paper explores the possibility to speed up optimization using variations of the ΔQ method to solve the network hydraulics. First, the ΔQ method was used inside the evaluation function where each tested alternative was hydraulically solved and ranked. Then, the convergence criterion was relived in order to reduce the computation time. Although the accuracy of the obtained hydraulic results was reduced, these were feasible and interesting solutions. Another modification was tested, where the ΔQ method was used just once to solve the hydraulics of the initial network, and the unknown flow corrections were added to the list of other unknown variables subject to optimization. Two case networks were used for testing and were compared to the results obtained using EPANET2. The obtained results have shown that the use of the ΔQ method in hydraulic computations can significantly accelerate the optimization of WDN.

Key words | ΔQ method, acceleration, efficiency, genetic algorithms, looped networks, minimal basis loops

Damjan Ivetić (corresponding author)
Željko Vasilić
Miloš Stanić
Dušan Prodanović
Faculty of Civil Engineering,
University of Belgrade,
Bulevar kralja Aleksandra 73,
Belgrade,
Serbia
E-mail: *divetic@hikom.grf.bg.ac.rs*

NOMENCLATURE

$Q_{ij}^{(0)}$	initial flow distribution (m^3/s)
ΔQ	flow correction (m^3/s)
R_{ij}	pipe flow resistance
n	flow exponent (-)
L_{ij}	pipe length (m)
D_{ij}	pipe inside diameter (m)
Λ	friction factor (-)
g	gravity acceleration ($\cong 9.81\ m\ s^{-2}$)
C	Hazen–Williams roughness coefficient (-)
ΔH_{ij}	pipe headloss (m)
ΔH_{res}	head difference between reservoirs (m)
v_k	flow velocity (m/s)
p_j	pressure head (m)
f	fitness function ($),(€)
I	investment in the network ($),(€)
I_p	pressure head condition penalty ($),(€)
I_v	flow velocity condition penaly ($),(€)
C_p	specific value of pressure head penalty function ($), (€)
C_v	specific value of velocity penalty function (€)
t	computation time (s)

doi: 10.2166/hydro.2015.118

INTRODUCTION

In the last decades, modern engineering practice has been moving towards a higher degree of computer and software usage. As computer sciences experience fast progress, engineering practice is trying to catch up and utilize stronger processors and new software capabilities that are

introduced. One of the engineering fields which benefits from this progress is hydraulics and water system analysis, especially the numerical modelling of fluid flow and optimization of the design and performance of water systems. In this paper the focus is on the optimization of water distribution networks (WDN), specifically the long computation time needed for multiple runs of the hydraulic solver and the possibility to reduce it.

In order to accelerate the computation or optimization process, different approaches are possible: the use of more powerful computers (faster processors), parallel computing systems with adapted hydraulic solvers, hybrid algorithms, more efficient hydraulic methods or any combination of the previously mentioned options. The first approach is strictly hardware-driven, in which the focus is on the IT industry and its capability to develop hardware with more raw power. The second approach has been addressed by numerous authors, computer and communication scientists, showing a variety of success in the results (Di *et al.* 2009; Artina *et al.* 2012; Marques *et al.* 2012; Smith *et al.* 2013). The SWMM numerical model for hydrodynamic rainfall–runoff and urban drainage simulations has been parallelized with minimal changes to the original code by Burger *et al.* (2013), where the achieved speedup was from six up to 10 times. On the other hand, several attempts to parallelize hydraulic solvers inside the EPANET2 have not been that successful (Alonso *et al.* 2000; Crous *et al.* 2012; Burger 2014) although it seemed a straightforward task. The idea was to tackle the matrix operations performed in each iteration step of the solver with several available packages that support parallel linear algebra (von zur Gathen 1993; Van de Gejin 1997; Agullo *et al.* 2009). In the work of Diao *et al.* (2013), significant code changes were made in order to implement a domain decomposition as an idea for EPANET2's acceleration. The resulting speedup was encouraging, reaching the value of eight times on one tested network but the issue of severe and complicated code changes makes this approach hard to implement. Zecchin *et al.* (2012) showed that iterative solvers preconditioned with the algebraic multigrid method (AMG) are faster than the current EPANET2 solvers only in cases of large artificial networks, while for typical EPANET2 problems speedup was not obtained. Different optimization algorithms have been used so far for the WDN design optimization, such as differential evolutionary algorithms (Suribabu 2010) and modified genetic algorithms (Montesinos *et al.* 1999; Neelakantan & Suribabu 2005). Hybrid algorithms, in general, as the tools for multi-objective design, have outperformed the non sorting genetic algorithm (NSGA II), mostly in terms of diversity of obtained Pareto fronts (Wang *et al.* 2015a), while lowest computational burden has been achieved with the low-level hybrid algorithm proposed by Craeco & Franchini (2012). Since the focus of the paper is not on the optimization method, the standard genetic algorithm (GA) (Savic & Walters 1997), efficient in handling a single objective optimization problem (Holland 1992), is used in combination with more efficient hydraulic methods.

Currently, when a water supply network is to be optimized, usually EPANET2 is used as the solver (Rossman 2007). EPANET2 is a reliable, free software package, available as a standalone EXE or DLL version, which can be easily integrated with most optimization programs. When used for single runs, EPANET2 is quite efficient in terms of the computation time. Large networks can be solved in a few minutes at maximum. However, a research scenario such as the optimization of a network design, involves multiple runs of a hydraulic model, requiring significant computation time for a single run of the optimization algorithm (Diao & Rauch 2013).

To solve the continuity equation EPANET2 uses a so-called global gradient algorithm (GGA) originated by Todini & Pilati (1987). The hydraulic basis of this method, and all other node methods, originates from Hardy Cross and the method he originally called the method of balancing flows (Cross 1936). At that time, Hardy Cross also introduced the ΔQ method or the method of balancing heads, for hydraulic calculation of a network (Cross 1936). The ΔQ method was derived from Cross's original moment distribution method, which he used for structural analysis. The method was in common use in the 1950s and 1960s until the arrival of computers and the node methods took over the throne. Both methods introduced by Hardy Cross originate from 'paper and pen' solvers time, and thus are classified as the local approaches for a network solution (Todini & Rossman 2013). The most significant difference is that in the node-based method, the number of non-linear equations is equal to the number of nodes while in the ΔQ

method it is equal to the number of loops in the network. The number of loops is usually much smaller than the number of nodes. This already implies a modest reduction of computation time if it is presumed that a similar amount of resources is needed for both types of equations. For example, the large BWSN2 benchmark network (Ostfeld *et al.* 2008) with 12,527 nodes has only 2,308 loops. The authors' experience is that even aggregated, real size water distribution networks usually have about five time less loops than nodes.

Craeco & Franchini (2014) presented a comparison of the Newton–Raphson global algorithm and the ΔQ method (loop flows algorithm), in which the latter had shown a slightly smaller computation burden on fictitious networks. In the same paper, the ΔQ method is recommended for network design optimization algorithms.

In this paper, the possibility of using the ΔQ method in hydraulic calculations of the network inside the optimization algorithm's evaluation function was explored. An iterative solver of the non-linear equations based on the ΔQ method was programmed in C++ as a DLL and called from the optimization algorithm to make it comparable with the EPANET solver. The base point for the analysis was the *upgraded ΔQ method* in which the hydraulics was solved using the ΔQ method in which the shared flow corrections are updated in the equations as the computation progresses. Furthermore, the following three modified variations were derived: the *fixed ΔQ method*, the *variable ΔQ method* and the *fixed iteration ΔQ method*. With the first two variations, an attempt was made to bypass the computational burden of the iterative solver. The idea was to promote intelligent, hydraulically based solutions. The first results presented by the authors (Stanić *et al.* 2012; Ivetić *et al.* 2014), on the well-known problem of the New York tunnels (NYT), were promising both in computational time and in reaching the suboptimal solution. A step forward in terms of the network complexity was made, and the intermediate network of Fossolo (FOS) (Bragalli *et al.* 2012) was tested. All of the obtained results are presented in this paper.

The paper is structured as follows: in the 'Methods' section, the basics of the ΔQ method and minimal loop detection algorithm are recapitulated, followed by a brief description of the used optimization algorithm and variations of ΔQ method implementation. The section is concluded with a brief formulation of the two case networks and the performance indicators used for the comparison with the EPANET2-based algorithm. The obtained improvements of the performance indicators are presented and deliberated in the 'Results and discussion' section. Finally, conclusions are drawn and directions for further analysis are formed.

METHODS

In order to employ the ΔQ method for hydraulic calculations, an analysis of network graph topology has to be performed in the pre-processing stage. It is necessary to detect all cycles, or loops in the network which are later split and thus the network structure is changed (a tree-structured graph is obtained). Tree-structured or branched networks are quite easy to handle in a water distribution analysis and unknown flows and nodal heads can be obtained in a double sweep algorithm (Stanić *et al.* 1998).

The ΔQ method

The ΔQ method was originally presented by Hardy Cross in 1936. It was proposed as one of the two possible methods for flow analysis in networks of conduits. In his original work, the ΔQ method was called the 'Method of balancing heads' (Cross 1936). The method is based on the fact that in every closed loop (circuit) of the water supply network the sum of total head loss is equal to zero. This is derived from the conservation of energy equation for a closed loop.

In order to apply this method, an initial distribution of the flows needs to be assumed. In this research, graph theory algorithms were used to obtain the initial flow distribution. A graph breadth first search (BFS) propagation algorithm (Jungnickel 2005) is started from a randomly selected source node in the network (reservoir) and the propagation continues until all the nodes in the network have been reached. This propagation results in a spanning tree (ST) which is a branched network. The order in which the nodes are visited and the ST is formed is obtained as well, and so it can be used to perform the backward sweeping of node demands in order to determine the initial

flow distribution. The ST is missing pipes that have not been used during propagation, so they will be split at some point (Figure 1(b)), and added to the ST to form a fictitious-branched network (FBN). With one sweep backwards through the FBN, satisfying continuity equation in every node, the initial flow distribution is obtained. The initial flows for the split pipes are equal to zero. These flows are noted as $Q_{ij}^{(o)}$ and most likely will not satisfy the previously mentioned condition for the head losses in the loop (Figure 1(a)). This implies that corrections to that initial distribution must be made.

At the same split point flow correction ΔQ is introduced. The Hazen–Williams (HW) equation is used to calculate pressure head loss in a pipe. The expanded form of the sum of the head losses for the loop (Figure 1(a)), written in the adopted clockwise direction for the flow correction ΔQ, becomes:

$$\begin{aligned} &+ R_{12}\,(Q_{12}^{(o)} + \Delta Q)\left|Q_{12}^{(o)} + \Delta Q\right|^{n-1} \\ &+ R_{23}\,(Q_{23}^{(o)} + \Delta Q)\,|Q_{23}^{(o)} + \Delta Q|^{n-1} \\ &- R_{13}\,(Q_{13}^{(o)} - \Delta Q)\,|Q_{13}^{(o)} - \Delta Q|^{n-1} = 0 \end{aligned} \tag{1}$$

where R_{ij} is the HW pipe flow resistance characteristic, $Q_{ij}^{(o)}$ is the initial pipe flow ($m^3\,s^{-1}$) and $n = 1.85$ is the flow exponent (–). The pipe flow resistance characteristic is calculated as:

$$R_{ij} = \frac{10.67L}{C^{1.85}D_{ij}^{4.87}} \tag{2}$$

where L_{ij} is the pipe length (m), D_{ij} is the pipe inside diameter (m) and C is the HW roughness coefficient (–). Equation (1) is a nonlinear equation that needs to be solved for the flow correction ΔQ. For this purpose, the Newton (also known as Newton–Raphson) iterative method (Hoffman 2001) was employed, which converges quadratically if the initial approximation is sufficiently close to the solution. The general form of the solution is:

$$\Delta Q^{i+1} = \Delta Q^{i} - \frac{\sum_{loop} sign \cdot R_{ij}\left(Q_{ij}^{(o)} + \sum_{pipe} sign \cdot \Delta Q_p^i\right)\left|Q_{ij}^{(o)} + \sum_{pipe} sign \cdot \Delta Q_p^i\right|^{n-1}}{n\sum_{loop} R_{ij}\left|Q_{ij}^{(o)} + \sum_{pipe} sign \cdot \Delta Q_p^i\right|^{n-1}} \tag{3}$$

where i is the iteration number, ij is the ij-th pipe in the loop and ΔQ_p is the p-th flow correction that corrects the initial flow in the pipe ij (there can be more than one, if the pipe is common for two loops (Figure 1(c)). The *sign* equals one (1) if the direction of the introduced correction ΔQ_p is the same as the direction of the initial flow and minus one (−1) if otherwise. The iterative calculation is done until the desired precision is reached.

The number of equations that need to be solved corresponds to the number of loops in the network. In the work of Todini & Rossman (2013), this method is called the loop flows algorithm and it solves equations for all the loops in the network simultaneously. In this research, equations were solved in succession as the speed of the convergence can be improved if the flow correction for one loop, calculated in the current iteration, is used to calculate the flow correction for another loop with which it shares a common pipe. This way the ΔQ solver is upgraded. Take Figure 1(c), for example: If the flow correction for the first

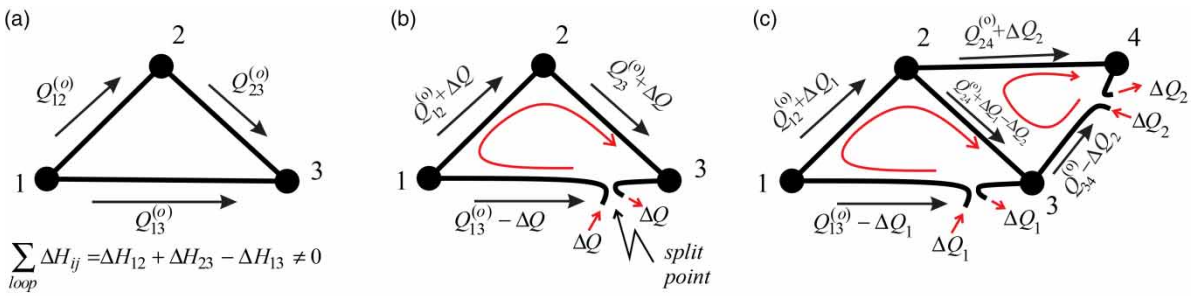

Figure 1 | (a) Head loss sum in loop; (b) introduction of flow corrections; (c) loops with common pipe.

loop is calculated in the i-th iteration $\Delta Q_1^{(i)}$, when the calculation for the flow correction in the second loop $\Delta Q_2^{(i)}$ is conducted, flow in their common pipe 2–3 can be expressed as $Q_{23}^{(o)} + \Delta Q_1^{(i)} - \Delta Q_2^{(i-1)}$.

Aside from the 'ordinary loops', the term 'pseudo loop' is introduced (Larock *et al.* 2000). This is a loop that is formed between two reservoirs/tanks with defined heads. A number of such loops is one less than the number of reservoirs in the network. In that case Equation (3) becomes:

$$\Delta Q^{i+1} = \Delta Q^{i} - \frac{\sum_{loop} sign \cdot R_{ij}\left(Q_{ij}^{(o)} + \sum_{pipe} sign \cdot \Delta Q_p^i\right)\left|Q_{ij}^{(o)} + \sum_{pipe} sign \cdot \Delta Q_p^i\right|^{n-1} - \Delta H_{res}}{n \sum_{loop} R_{ij}\left|Q_{ij}^{(o)} + \sum_{pipe} sign \cdot \Delta Q_p^i\right|^{n-1}} \quad (4)$$

where ΔH_{res} is the difference of heads in the reservoirs.

Minimal basis loops' detection

Prior to conducting the calculations using the ΔQ method, network loops need to be detected. Initial loops' detection is done based on the results of the BFS propagation algorithm previously mentioned. The number of loops corresponds to the number of unused links during the BFS propagation, which are not a part of the spanning tree so they must be closing the circuit and creating the loops. The initial loops are not likely to be geometrically minimal. The geometrical minimal loop is defined as the one that cannot be presented as a union of any other loops. Detecting these loops is not an easy task. Some algorithms are based on using the outer or the 'back edges' of the network (Jha 2007) but these have to be predefined. It is clear that in the case of thousands of pipes this would be a very demanding job. Craeco & Franchini (2014) presented an algorithm which utilizes the Dijkstra algorithm to search for the shortest path (from the topological viewpoint, meaning that all graph links have the same weight of one) between two nodes.

In this research, another approach based on the graph theory algorithms is presented. A network loop is considered to be minimal if it shares common pipes with a minimum number of other loops. Also, the number of these shared pipes should be minimal. The authors refer to these loops as the topologically minimal loops and in the following sections they are referred to as the 'minimal basis loops'. This algorithm does not guarantee that identified minimal basis loops will be the absolutely topological minimal or geometrically minimal, which after all is not necessary for the ΔQ method to be employed. However, minimal basis loops defined in this manner will enable obtainment of the simplest form of nonlinear equations to be solved. The algorithm has been tested on numerous examples and benchmark problems used in urban water modelling and optimization literature and, in most cases, it found minimal basis loops that are both topologically and geometrically minimal. The presented algorithm for the minimal basis loops' identification takes three steps. First, the BFS propagation algorithm is run to obtain the initial spanning tree (ST) and the initial set of loops (*InitSet*). The second step is the transformation of the ST in order to get the set of loops with the smallest number of pipes (*STSet*). Finally, decomposition is done if needed and a set of final minimal basis loops (*FSet*) is extracted from the graph. The algorithm steps are explained in the following section.

Take, for example, the network in Figure 2(a). The BFS propagation provided the ST (solid line) and unused pipes (marked with dashed line) which complete two loops with the following pipes – (p1, p5, p4) and (p1, p2, p3, p4). These loops present the *InitSet* with such configuration that there are two pipes (p1 and p4) in which flow needs to be corrected with both flow corrections ΔQ_1 and ΔQ_2,

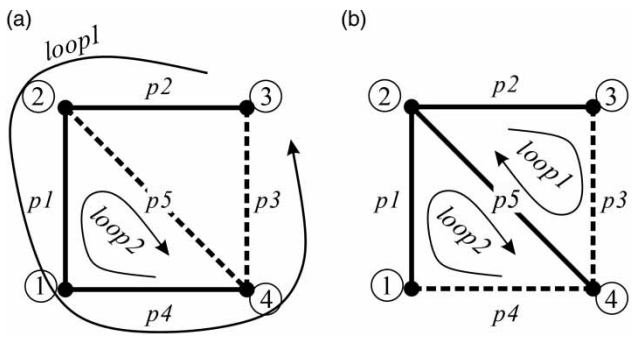

Figure 2 | Different configurations of the spanning tree (ST).

since both of these pipes are common to the two loops. The total number of pipes in the loops for this configuration is $4+3=7$. In order to get a better configuration, unused pipes will be swapped with adjacent ones that are used during the propagation. For loop1, pipe p3 can be swapped either with pipe p2 or p4. In both cases the ST configuration would be such that the number of links in the loops would stay the same, $4+3=7$. Now the second loop, loop2, is tried. Its swapping pipe is p5 and it can be swapped with p1 or p4. If pipe p5 is swapped with p4, a new configuration of the ST gives the total number of links in loops as $3+3=6$ (Figure 2(b)). The same configuration would be obtained if p5 was swapped with p1. So one of these configurations is chosen (no matter which one) as it is better than the previous one and it is marked as the new best configuration. The procedure is repeated until a better configuration cannot be reached. The last configuration is remembered as the *STSet*. For the considered simple example, this configuration is at the same time the one with the minimal basis loops. This means that in the last step of the algorithm, which is the extraction of the final set of minimal basis loops, *STSet* will be identified as *FSet.* This, however, does not have to be the case, so in the following paragraph the last step of the algorithm (extracting the *FSet*) will be explained in detail and then illustrated on a more complex example.

The extraction of the final minimal basis loops set (*FSet*) is based on the sorting of the *STSet* set of loops by their length (number of pipes) and creation of the overlapping matrix which shows the number of common (shared) pipes between any two loops. The overlapping matrix A is a square matrix with the dimensions $NL \times NL$, where NL is the number of the loops. The diagonal elements of the matrix A are $Aii=0$ since it represents an overlapping of loop with itself and Aij is equal to the number of shared pipes between the loops i and j. The decomposition considers two loops at the time and it starts from the loops with the highest number of shared pipes. The loop with more pipes, or with more shared pipes, is considered to be the large one (*Lloop*) and the other one is the small one (*Sloop*). Once the candidates for the decomposition have been detected, they are combined to create a new loop (*Nloop*) which could possibly replace one of them in the *STSet* of loops if it is a 'more minimal basis'. First, it is checked whether the *Sloop* is a minimal one. It is defined as minimal if it shares one pipe with the others, or if it does not share any pipes, in which case it is called a 'hanging loop' (*Hloop*). If it satisfies this condition it is moved to the final set of loops (*FSet*). If *Nloop* has a smaller number of pipes than *Lloop* it replaces it in the *STSet*, which now has one loop less since the *Sloop* is transferred to the *FSet*. If it has more pipes than the *Lloop*, it still can be a candidate to stay in the *STSet* if it shares smaller (or the same) number of links with the other loops from the *FSet* and *STSet*. This comes from the definition of a minimal basis loop in which another criterion, besides the minimum number of pipes in a loop, is the minimum number of shared pipes. After this, the current *STSet* is searched for the loops (*Hloops*). If found, they are also moved from the *STSet* to the *FSet*. The described steps of the algorithm are repeated until the number of loops in the *FSet* corresponds to the number of loops *NL*. A pseudo code for the algorithm is presented in Figure 3.

For illustration purposes, consider the network in Figure 4. The result of the first step of the algorithm is the

```
InitSet = run BFS                    // step ONE
STSet = ST Transformation            // step TWO
init Fset                            // begin step THREE
while 1
   A=sort(STSet)
   find Lloop, Sloop
   create Nloop
   if Sloop is minimal
            Fset =(Fset) U (Sloop)
            STSet=(STSet) / (Sloop)
   end if
   if length(Nloop)<length(Lloop)
            STSet={(STSet) / (Lloop)} U (Nloop)
   else
      if sharedlinks(Nloop) <= sharedlinks(Lloop)
            STSet={(STSet) / (Lloop)} U (Nloop)
      end if
   end if
   A=sort(STSet)
   search for Hloop
   if exist(Hloop)
      Fset =(Fset) U (Hloop)
     STSet=(STSet) / (Hloop)
   end if
   if size(FSet)=NL
     break while
   end if
end while                            // finish step THREE and algorithm
```

Figure 3 | The algorithm for the minimal basis loops detection.

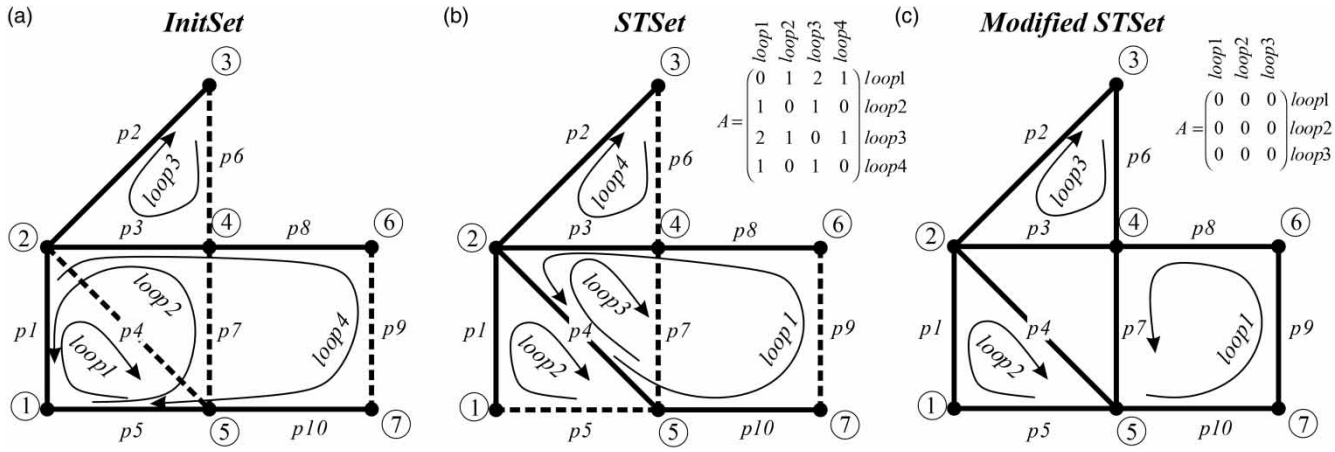

Figure 4 | An illustration of the minimal basis loops detection algorithm.

initial set of the loops *InitSet* which is shown in Figure 4(a). This set consists of four loops with a total of 16 pipes and nine of them are shared between the loops. The second step of the algorithm, described earlier, will provide an improved set of the loops – *STSet* (Figure 4(b)). This set has a smaller number of pipes (14) and smaller number of pipes that are shared (4). Now the previously described third step is employed to extract the final set of loops. The first matrix A is created. Loop1 (p3, p8, p9, p10, p4) is chosen for the *Lloop* since it is the longest one. It shares pipes with all other loops but with loop3 (p3, p7, p4) it shares most of them, two to be precise (p4 and p3). Thus, loop3 is chosen as the *Sloop*. Combining these two loops the *Nloop* is obtained (p8, p9, p10, p7). *Sloop* is minimal as it shares only one pipe with other loops, p3 with loop4 and p4 with loop2, and is transferred from the *STSet* to the *FSet*. *Nloop* has four pipes which is smaller than the five pipes of the *Lloop*, so *Nloop* replaces *Lloop* in the *STSet*. Now *STSet* is modified and it has three loops that share no links (Figure 4(c)). All of them are *HLoops* and are transferred to the *FSet*, which now has four loops and the algorithm is finished. *FSet* has the loops with a total of 13 pipes and three (p3, p4, p7) of them are shared between the loops.

Optimization method

GAs are employed as an optimization method and are efficient in terms of running time and finding suboptimal solutions (Holland 1992). GAs are called inside an optimization algorithm, after the pre-processing stage, with the task to generate coded solutions which are to be tested inside an evaluation function. The EPALG software, developed at the Faculty of Civil Engineering at the University of Belgrade, was chosen for the GA. For every solution examined, a new value of the fitness function is computed. Based on this value, examined solutions are ranked, where the solution with the lowest value of the fitness function is ranked as suboptimal.

In this paper, some of the EPALG settings were kept fixed throughout all of the optimization algorithm runs to have comparable results between different methods. These settings regard the GAs' ability to converge to the best suboptimal solution and affect the way mutation, crossover, selection and replacement are done inside GAs themselves. The settings used were:

- Mutation: Type: Reinit number of bits affected: 1
- Crossover: Type: Two point Probability: 0.8
- Selection: Type: Tournament Number of solutions competing: 2
- Replacement: Type: Uniform Portion of population being replaced: 1

Implemented optimization algorithms

In the pre-processing stage after the FBN is derived, unknown flows were calculated for the new network using a back sweep algorithm. These flows satisfy the continuity

equation in the network nodes, but are not an exact solution for the pipe flows, which means that they can be used as a starting assumption of the flows for the ΔQ method. This is done only once, so later calls of the evaluation function will just include an iterative solution for the flow corrections, in the manner explained in 'The ΔQ method' section. When the convergence condition is met, final flows can be calculated and pressure head distribution is obtained using a forward sweep algorithm. This idea was named the *upgraded ΔQ method* (*Method A*), and it became the starting point of the investigation.

Several variations of optimization algorithm were derived and examined in order to reduce computation time. Tests were made with the version of the optimization algorithm in which values of the flow corrections were kept fixed, equal to the initial values obtained in the pre-processing stage – the *fixed ΔQ method* (*Method B*). There was a major applicability issue with this method in the more complex and new networks. In complex networks, with a large number of loops with diameters inside a single loop varying significantly, the starting assumptions for flow corrections, if they can be computed, can have a significant difference from their final, exact values. For the new networks for which there are no predefined pipe diameters, starting assumptions for flow corrections cannot be calculated in the pre-processing stage.

In order to overcome these shortcomings, another approach was tested, the *variable ΔQ method* (*Method C*). The initial values of flow corrections, as calculated during the pre-processing stage, were assumed to be new unknown variables, subject to optimization together with other unknowns (e.g., pipe diameters). In addition, one more penalty function has to be added in the fitness function calculation, which will guide the optimization of the flow corrections. Mutual for these two approaches (*Methods B* and *C*) is the fact that in each evaluation function call, no time-consuming iterative hydraulic computation has to be done. Furthermore, based on the results of previous tests (Ivetić *et al.* 2014) with *fixed ΔQ method (B)*, the idea to run only a few iterations inside ΔQ method solver in order to obtain some 'near to exact' flow corrections (*fixed iteration ΔQ method* (*Method D*) was explored.

In order to make a comparison with the EPANET2 solver, an unbiased, iterative solver for the ΔQ method was made in the form of a DLL file which will be integrated inside the evaluation function of the optimization algorithm.

The upgraded ΔQ method inside an optimization algorithm (Method A)

The key points for the hydraulic solver implementation inside the optimization algorithm are the pre-processing stage and the evaluation function. The ways that these two parts of the algorithm are formed will be the focal points for all of the presented methods' description. Here the ΔQ method iterative solver for Equation (1) is programmed as a DLL file. Equation (1) is solved using the flow corrections obtained through an iterative computation of Equation (3). In every pass through the evaluation function, a solver is run externally and the results of the hydraulic calculation (pressure and flow distribution) are used to compute a value of the fitness function. The pre-processing stage, which is mutual for all alternatives, and the evaluation function computation are described below.

- Pre-processing stage (preceded by a call of GA):

1. The BFS algorithm is run, a directed ST is obtained, the FBN is formed and the initial flow distribution is determined.
2. Minimal basis loop detection as explained in the previous section (Figure 4).
3. If the starting disposition includes prebuilt pipes (pipes with known diameter e.g., NYT), a network flow distribution is computed and the resulting flow corrections ΔQ can be stored as starting values for further GA calls. It is presumed that flow corrections calculated this way will not differ much from their final values which will be calculated for every call of the evaluation function, thus, this should speed up the convergence of Equation (3), if it is possible.

- Evaluation function calculation:

1. For every tested alternative, the 'exact' values of the flow corrections are computed by calling the iterative solver (DLL file) from the optimization algorithm. In order to improve the convergence and thus to reduce the computational burden, flow corrections from current iteration are used as explained in 'The ΔQ method' section.

2. Using exact flow corrections, the pressure head distribution for the network is calculated in only one pass.
3. The fitness function is calculated.

The fixed ΔQ method inside an optimization algorithm (Method B)

The fixed ΔQ method is developed and implemented in a similar manner to *Method A*. It was differentiated from *Method A* by omitting the iterative calculation of Equation (1), which means that there is no need to call the DLL solver inside the evaluation function.

- The evaluation function calculation:

1. For every tested alternative, the previously determined values of flow corrections ΔQ are used. This variation of ΔQ method implementation is feasible only for networks with pre-existing pipes (pipes with known diameters) due to the fact that the initial values of the flow corrections must be computed in the pre-processing stage.
2. Using the fixed flow corrections, the pressure head distribution for the fictitious branch network is calculated in only one pass.
3. The fitness function is calculated.

The variable ΔQ method inside an optimization algorithm (Method C)

In *Method A*, each evaluation function's calculation involves the iterative calculation of the 'exact' flow corrections. In the variable ΔQ method, it is assumed that flow corrections are unknown variables, whose values are, together with pipes' diameters values, optimized.

- The evaluation function calculation:

1. For every tested alternative (having pipes' diameters and flow corrections as variables to optimize) only the pressure head distribution for the FBN is calculated in one pass.
2. The fitness function is calculated.
3. An additional penalty function is added to the value of the fitness function, to compensate for the fact that the used flow corrections are not 'exact'. If the pressure drop between the neighbouring nodes where the loop is split is positive in the direction of water flow, the value of the penalty function is zero. This solution is feasible, meaning that the pressure reducing valve could be installed in the place where the pressure drop occurs. However, if the pressure drop between neighbouring nodes where the loop is split is negative, the solution would require pumping between these nodes, and it has to be penalized. The same penalty value as for pressure head deficiency is used.

To reduce the search space for flow corrections, the value is entered as multiplication of the flow corrections computed in the pre-processing stage ΔQ^0, $M \times \Delta Q^0$. The range of possible values for multiplication factors can vary, meaning that the used search space is problem dependent and should be handled with care.

Method C does not solve the network for the 'exact' flows, instead it produces some sort of suboptimal network flows which further implies that the accuracy of the hydraulic results is degraded compared to *Method A*.

The fixed iteration ΔQ method inside an optimization algorithm (Method D)

This approach uses the DLL iterative solver with a predefined number of iterations.

- The evaluation function calculation:

1. For every tested alternative, near to exact values of flow corrections are computed by a predefined number of iterations in a solver. As in the previous case, convergence is improved through a current iteration flow correction update.
2. Using near to exact flow corrections, the pressure head distribution for the network is calculated in only one pass.
3. The fitness function is calculated.

The reference method: EPANET2 DLL inside an optimization algorithm (Method R)

The reference optimization algorithm, used for comparison purposes, in terms of both suboptimal solutions and needed computation time, is based on the EPANET2 hydraulic solver. The solver is called as a DLL file in every pass through the evaluation function, and obtained results are used for the fitness function value computation. In this case the pre-processing stage is not needed, while the evaluation function processing has the following form.

- The evaluation function calculation:

1. For every tested alternative, an EPANET2 DLL file is called for hydraulic simulation of the network, in the same way as the DLL for the ΔQ method is called in optimization algorithms (*A*) and (*D*). The results, needed for the fitness function calculation, are extracted from the solver and stored.
2. The fitness function is calculated.

The comparison between different optimization algorithms is made through the following performance indicators: 1) value of fitness function f; 2) computation time t; and 3) speedup factor, expressed as the ratio of the reference optimization algorithm (R) computation time and examined algorithm (A, B, C or D) computation time.

$$\text{Speedup factor} = \frac{t_R}{t_{A,B,C,D}} \tag{5}$$

In the case of algorithm A, the convergence criterion was $10^{-6}\ \text{m}^3\,\text{s}^{-1}$ in two successive iterations. For *Method D*, the number of iterations was fixed to 10 if the convergence criterion is not met.

Case networks

NYT network

The NYT reconstruction used for the initial testing of the GA with the ΔQ method application was extracted from the literature (Dandy *et al.* 1996). The starting network for optimization is presented in Figure 5(a).

This is an example of a gravitational WDN made out of $n_n = 20$ nodes with $n_r = 1$ source node or reservoir. The nodes and the reservoir are interconnected with $n_p = 21$ large pipes forming $n_l = 2$ loops. The current disposition of the system cannot satisfy the minimal nodal pressure head values of 20 m of water column. Therefore, it is necessary to reconstruct the network in order to meet the given condition in nodes. Changing the diameters of existing pipes is not possible due to the problem of excessive water demand shortfall so the remaining options are either: (a) to duplicate the existing tunnel with some of the diameters offered; or (b) to do nothing. The number of diameters in the catalogue for new pipes is 15. Together with the do nothing option, this makes 16 possible solutions for every pipe in

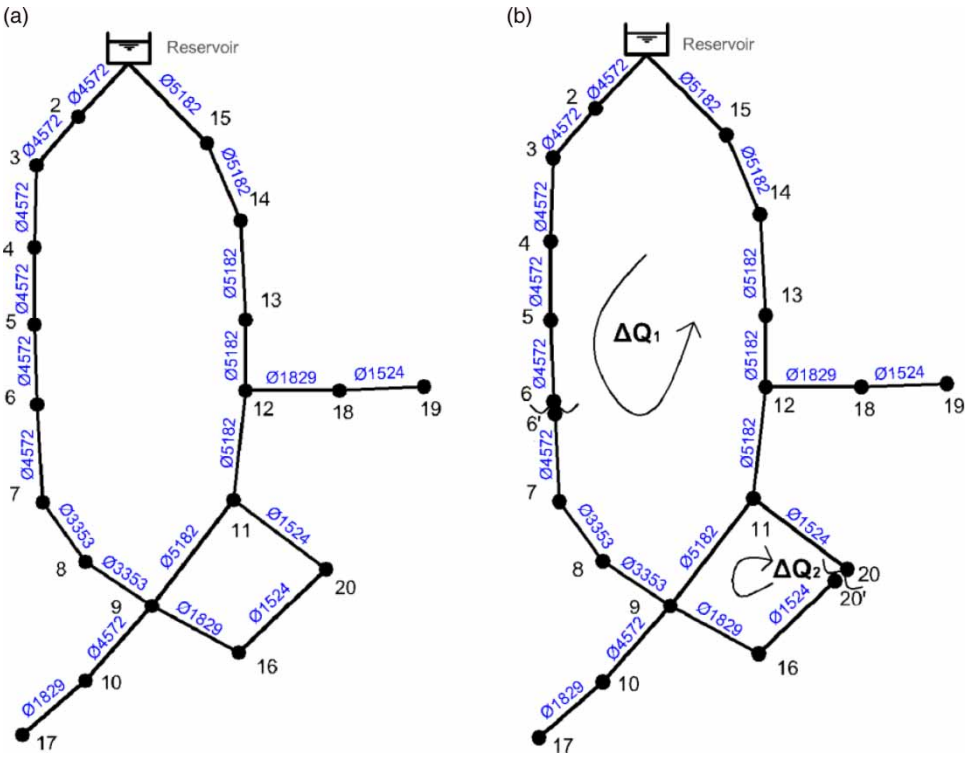

Figure 5 | (a) Starting NYT network; (b) modified fictitious branch network.

the network, forming a search space of size 1.93×10^{25}. The fitness function f for this example is made up of two parts; the first one is investment in the water network I (Equation (6)) and the second is the penalty I_p for failing to meet the minimal nodal pressure head values ($p^{min} = 20$ m H_2O).

$$\min_{\{D_k\}} f = I + I_p; \quad I = \sum_k C_k(D_k)L_k; \quad I_p = \sum_j \max(O, (p_j^{\min} - p_j)) \cdot C_p \tag{6}$$

In the above equation, C_k, D_k and L_k are cost of the new pipe per meter, diameter and length of the pipe, respectively, j is the number of a node, p_j is the pressure head value in the node j while O is the function whose value is above zero if the $p_j < p^{min}$. The specific value of the penalty function is $C_p = 15{,}000{,}000$ \$/m.

FOS network

With the NYT reconstruction issue being classified as a medium sized optimization problem (Wang *et al.* 2015b), a step forward was to test an intermediate size optimization problem. The FOS network, another benchmark example from the literature, was chosen. The starting network topography was defined from the EPANET2's inp file (http://emps.exeter.ac.uk/media/universityofexeter/emps/research/cws/downloads/data/3-epanet/FOS.inp).

The FOS network, as well as the NYT network, is a gravitational WDN. Pipe diameters are unknown and have to be optimized to meet the given conditions. The network includes $n_n = 36$ nodes with $n_r = 1$ source node or reservoir, interconnected with $n_p = 58$ pipes forming $n_l = 22$ loops. The minimum pressure head of all the demand nodes is 40 m, while the maximum pressure head of each node is given in Table 1. Besides pressure head condition, flow velocity condition needs to be satisfied, where in each pipe it has to be less than 1 m/s. For the network pipes, 22 different diameters are available in catalogue. This means that the search space has the size of 7.25×10^{77}. The fitness function is made of three parts: investment in the distribution network I, penalty I_p for unfulfilled pressure head conditions and I_v penalty for failing to meet maximum velocity condition.

$$\begin{aligned} &\min_{\{D_k\}} f = I + I_p + I_v; \quad I = \sum_k C_k(D_k)L_k \\ &I_p = \sum_j \max(O, (p_j^{\min} - p_j)), (p_j - p_j^{\max}) \cdot C_p \\ &I_v = \sum_k \max(O, (v_k - v_k^{\max})) \cdot C_v \end{aligned} \tag{7}$$

The pressure head penalty function has an expanded form compared to Equation (6), with a maximal pressure head condition introduced, where p_j^{max} is the maximal pressure head for j-th node. In the velocity penalty function v_k is the flow velocity in k-th pipe, v_k^{max} is the maximal velocity for k-th pipe. The specific values of penalty functions for pressure and velocity are $C_p = 15{,}000{,}000$ €/m and $C_v = 50{,}000{,}000$ €/m, respectively.

RESULTS AND DISCUSSION

The results of all presented ΔQ methods (A, B, C and D) are compared to the results obtained using the EPANET2 hydraulic solver *(R)*. The first tests of the ΔQ method

Table 1 | The maximum pressure heads of each demand node of FOS

N_i	P_{max} (m)	N_i	P_{max} (m)	N_i	P_{max} (m)	N_i	P_{max} (m)	N_i	P_{max} (m)	N_i	P_{max} (m)
1	55.85	7	53.10	13	59.10	19	58.10	25	56.6	31	56.60
2	56.60	8	54.50	14	58.40	20	58.17	26	57.6	32	56.80
3	57.65	9	55.00	15	57.50	21	58.20	27	57.1	33	56.40
4	58.50	10	56.83	16	56.70	22	57.10	28	55.35	34	56.30
5	59.76	11	57.30	17	55.50	23	56.80	29	56.5	35	55.57
6	55.60	12	58.36	18	56.90	24	53.50	30	56.9	36	55.10

implementation, including all variations (*A*, *B*, *C* and *D*) were run on the NYT network. These different approaches share the same modified FBN (Figure 5(b)). There are two loops in the network: the larger loop 1 contains a reservoir along with the nodes numbered from 2 to 15, while loop 2 has nodes 11, 9, 16 and 20. The location of the split is arbitrary and inconsequential. Loop 1 was split in the proximity of node 6 where a new node 6′ is introduced as a start node for the downstream pipe. Loop 2 was split close to node 20, with the new node 20′ being generated. Flow corrections for loops 1 and 2, ΔQ_1 and ΔQ_2, respectively, are introduced as demands in the nodes 6 and 20, and in the nodes 6′ and 20′ as negative demands or inflows. A comparison of obtained results, for 1,000 generations and population of 100, is given in Table 2.

For the FOS network 22 minimum basis loops were obtained. Loops are also geometrically minimal. The *InitSet* had a total of 175 pipes, the *STSet* 143 and the *FSet* 101 pipes. Optimization algorithms *A* and *D* were tested for four different GA settings (Table 3). Comparisons of computed performance indicators are shown in Figures 6 and 7, respectively.

The computer used for testing was an Intel i7-2630QM CPU with 6 GB of RAM memory. The obtained optimal solution using the EPANET2 DLL, for the NYT problem, is the same as one taken from the literature $f_{opt} = 38.6 \times 10^6$ \$ (Dandy *et al.* 1996), proving it as a valid reference for this investigation.

Table 2 | A comparison of the optimization algorithms performance indicators for NYT

GA optimization algorithm	Fitness function *f* [10^6 $]	CPU time *t* [s]	Speedup factor [-]
EPANET2 DLL (R)	38.6	390	/
Upgraded ΔQ (A)	38.6	18.5	21.1
Fixed ΔQ (B)	40.2	5	78.0
Variable ΔQ (C)	39.8	5.5	70.9
Fixed iteration ΔQ (D)	39.0	18.4	21.2

Table 3 | GA evaluation function settings for FOS network tests

No. test	1	2	3	4
GA generations	1,000	1,000	2,000	2,000
GA population	100	200	100	200

The computed performance indicators for the NYT network (Table 2) showed significant reduction of the computation time for the optimization algorithm in all of the examined variations (*A*, *B*, *C* and *D*). In terms of the speedup factor, best results were derived with the variation *B*, with the value of 78.0. This was expected since no iterative computation was performed inside the evaluation function. Approach *C* had a slightly smaller speedup factor of 70.9, probably due to the fact that an extra penalty function had to be computed for every examined alternative. It is to be expected to have a further reduction of this factor in real applications, since we have knowledge from the previous tests that the correct values of ΔQ_1 and ΔQ_2 deviate not more than 25% from the first iteration (Ivetić *et al.* 2014), and we could make the search space rather narrow. This is due to the fact that it is a case of an existing, simple network with just two loops. It is obvious that this approach needs to be adjusted in order to allow the GA to perform a successful search for the suboptimal values of flow corrections, in the rather broad search space. Methods *A* and *D* had shown a similar computational burden, with the speedup factors of 21.1 and 21.2, respectively. The similar computational burden is probably due to the fact that for the NYT example, a prefixed number of iterations (10) for variation *D* was sufficient to compute the 'exact' or at least near to the 'exact' values of flow corrections. Furthermore, approach *A* produced the best-known solution, while approach *D* was the second best with almost the same solution as the previous. Methods *B* and *C* produced slightly worse solutions. The suboptimal solution degradation occurs mostly due to the hydraulic results' inaccuracy, which resulted in higher penalty function value. The acceleration in algorithms using *ΔQ* methods is owed to the fact that less equations need to be solved, as well as the way the network is processed. The pre-processing stage, called just once, performs a large proportion of the necessary analysis, therefore later calls for the evaluation function take much less computation time than in reference algorithm.

In the case of the FOS network, tests with four different GA evaluation function settings using two variations *A* and *D* were done. Methods *B* and *C* could not be used in the presented manner since none of the pipe diameters was known at the beginning. From Figure 6, it is clear that in all tests,

the values of the fitness function were similar for reference algorithm R and the two examined variations of the ΔQ method implementation A and D. For this example, none of the used algorithms reached the best known solution, and it can be concluded that in terms of the suboptimal solution, algorithms A, D and R do not outperform each other. In case of A and D, finding these solutions took much less computation time (Figure 7). The speedup factor for algorithm A ranged from 39.3 to 62.6 and for algorithm D from 57.6 to 105.6. Between approaches A and D, roughly the same value of the fitness function was achieved, with algorithm D taking significantly less time. It is appropriate to point out the benefits of the current iteration flow correction update. For a single test run of evaluation function in

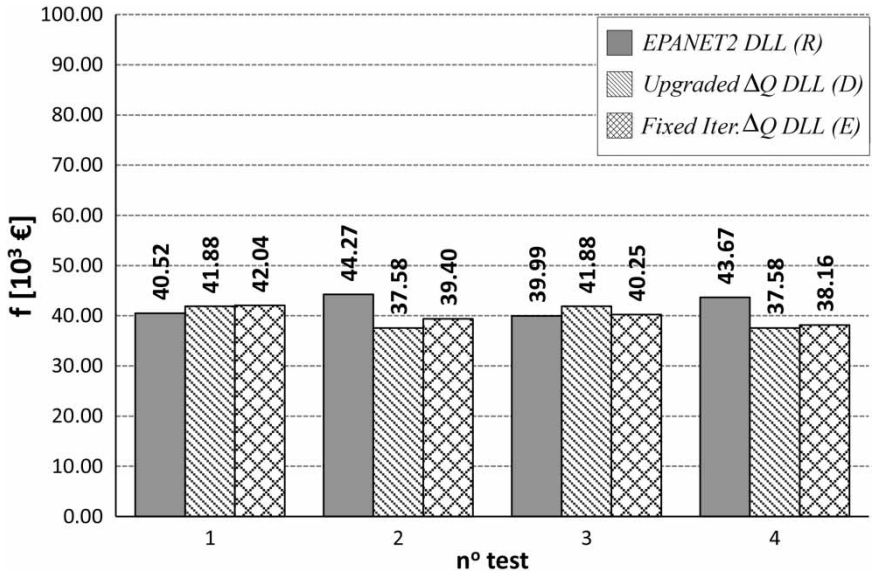

Figure 6 | Comparison of suboptimal solutions obtained for different optimization algorithm runs with FOS.

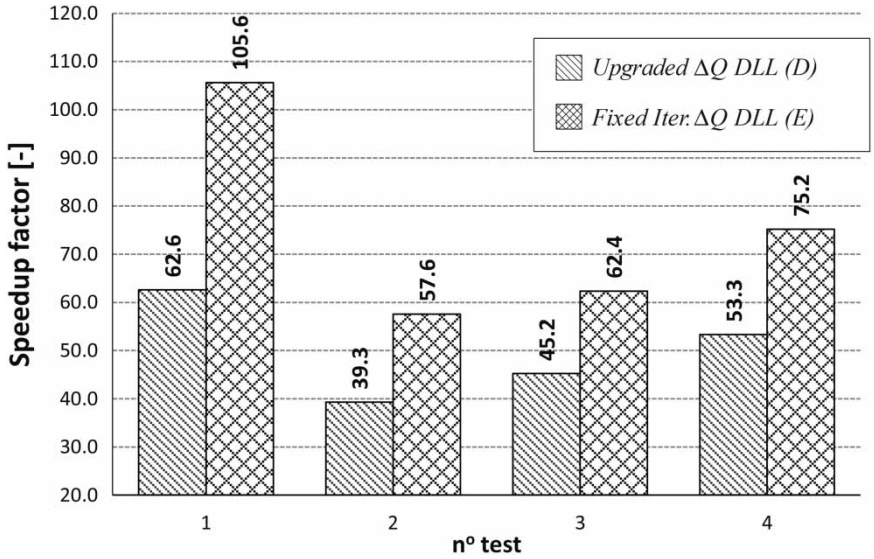

Figure 7 | Comparison of speedup factors (Equation (5)) for different optimization algorithm runs with FOS.

approach *A* for the FOS network, it took 23 iterations to reach the desired accuracy for the pipe flows of 10^{-6} m^3 s^{-1}, as opposed to 39 iterations for solution without this modification (about 40% reduction in number of iterations). Similar numbers were obtained for different test runs but this percentage clearly depends on the loops; configuration and pipe diameters.

CONCLUSIONS

During the optimization of the WDN, hydraulic computation of a network inside an evaluation function consumes most of the computation time. In this paper, use of the *ΔQ* method for hydraulic computation inside an optimization algorithm is presented through several different approaches. Each of these approaches requires a preprocessing stage, in which the loops in the original water network are detected through minimal basis loop detection algorithm, and network split into the FBN. If a prebuilt network exists, as in the NYT case, initial values of the flow corrections are assumed through hydraulic computation of this branch network.

The upgraded ΔQ method (A) approach computes the correct values of the flow corrections inside each evaluation function, the *fixed ΔQ method (B)* omits the iterative computation of the flow corrections by using the initial values throughout the optimization run, while the *variable ΔQ method (C)* includes the values of the flow corrections, through multiplicative factors, as additional variables for optimization. Finally, the *fixed iteration ΔQ method (D)* is actually the approach *A* used with a fixed number of iterations of Equation (3). Only methods *A* and *D* use the hydraulic solver programmed as a DLL, as well as the reference algorithm (*R*) which uses EPANET2 in a DLL form. The first tests of the presented variations were done on the NYT problem. In all of the cases, significant computation time reduction was achieved, primarily due to the fact that the *ΔQ* method solves fewer equations than the GGA integrated in EPANET2. These type of results are expected for real size networks in which the number of loops is rarely higher than 20% of the number of nodes (e.g., BWSN2) (Ostfeld *et al.* 2008). This implies 80% less equations to be solved with the *ΔQ* method.

On top of this, in the variations *B* and *C,* the network hydraulics is solved only once in the pre-processing stage which makes them even faster. In terms of the quality of the suboptimal solutions obtained, only algorithm *A* managed to compute a global optimum. The others, *B, C* and *D*, have shown a slight degradation in this performance indicator, which is caused by the hydraulic inaccuracy.

Further testing was undertaken on the benchmark example of the FOS network, where approaches *A* and *D* were compared with the reference EPANET2 based algorithm *R*. The tests, with four different GA configurations, have shown a major speedup, with the speedup factor reaching the value of over 100. Since method *D* had a fixed number of iterations it was the fastest method tested. In terms of finding the suboptimal solutions, both *A* and *D* found similar solutions as algorithm *R*.

The obtained results have shown that the *ΔQ* method is remarkably faster than EPANET2 when used in medium and intermediate optimization problems. Further investigation will be undertaken on real size water distribution networks. Apart from just changing the hydraulic solver, this approach can be utilized combined with other ways to reduce computation time, such as parallelization, network decomposition, etc.

ACKNOWLEDGEMENTS

The authors express their gratitude to the Serbian Ministry of Education and Science for support through the project TR37010: 'Rain water drainage systems as part of the urban and transport infrastructure'.

REFERENCES

Agullo, E., Demmel, J., Dongarra, J., Hadri, B., Kurzak, J., Langou, J., Ltaief, H., Luszczek, P. & Tomov, S. 2009 Numerical linear algebra on emerging architectures: The plasma and magma projects. *J. Physics Conf. Ser.* **180**, IOP Publishing, 012037.

Alonso, J. M., Alvarruiz, F., Guerrero, D., Hernández, V., Ruiz, P. A., Vidal, A. M., Martínez, F., Vercher, J. & Ulanicki, B. 2000 Parallel computing in water network analysis and leakage minimization. *J. Water Resour. Plann. Manage.* **126** (4), 251–260.

Artina, S., Bragalli, C., Erbacci, G., Marchi, A. & Rivi, M. 2012 Contribution of parallel NSGA-II in optimal design of water distribution networks. *J. Hydroinform.* **14** (2), 310–323.

Bragalli, C., D'Ambrosio, C., Lee, J., Lodi, A. & Toth, P. 2012 On the optimal design of water distribution networks: a practical MINLP approach. *Optim. Eng.* **13**, 219–246.

Burger, G. 2014 Parallel computing in urban water management: A Model-Based Parallel Computing Approach to Reduce the Runtime of Applications in Urban Water Management. Doctoral dissertation, University of Innsbruck, Austria.

Burger, G., Sitzenfrei, R., Kleidorfer, M. & Rauch, W. 2013 Parallel flow routing in SWMM 5. *Environ. Modell. Softw.* **53**, 27–34.

Craeco, E. & Franchini, M. 2012 Fast multi-objective design algorithm combined with an a posteriori procedure for reliability evaluation under various operational scenarios. *Urban Water J.* **9**, 385–399.

Craeco, E. & Franchini, M. 2014 Comparison of Newton-Raphson global and loop algorithms for water distribution network resolution. *J. Hydraul. Eng.* **140**, 313–321.

Cross, H. 1936 Analysis of flow in networks of conduits or conductors. Bulletin University of Illinois No. 286.

Crous, P., van Zyl, J. & Roodt, Y. 2012 The potential of graphical processing units to solve hydraulic network equations. *J. Hydroinform.* **14**, 603.

Dandy, G. C., Simpson, A. R. & Murphy, L. J. 1996 An improved genetic algorithm for pipe network optimization. *Water Resour. Res.* **32**, 449–458.

Di, P., Berardi, L., Khu, S. T. & Savic, D. 2009 Efficient multi-objective optimal design of water distribution networks on a budget of simulations using hybrid algorithms. *Environ. Modell. Softw.* **24**, 202–213.

Diao, K. & Rauch, W. 2013 Controllability analysis as a pre-selection method for sensor placement in water distribution systems. *Water Res.* **47** (16), 6097–6108.

Diao, K., Wang, Z., Burger, G., Chen, C.-H., Rauch, W. & Zhou, Y. 2013 Speedup of water distribution simulation by domain decomposition. *Environ. Modell. Softw.* **52**, 253–263.

Hoffman, J. D. 2001 *Numerical Methods for Engineers and Scientists*, 2nd edn. Marcel Dekker, New York, USA.

Holland, J. H. 1992 Genetic algorithms. *Scientific American* **267**, 66–72.

Ivetić, D., Vasilić, Ž., Prodanović, D. & Stanić, M. 2014 Implementing ΔQ method to accelerate the optimization of pressurized pipe networks. In: *Proceedings of the 16th Annual Water Distribution Systems Analysis Conference, WDSA, 14–17 July*, Bari, Italy, Procedia Engineering, Elsevier.

Jha, K. 2007 Automatic minimal loop extraction and initialisation for water pipe network analysis. *Int. J. Simul. Syst. Sci. Technol.* **8**, 8–19.

Jungnickel, D. 2005 *Graphs, Networks and Algorithms*, 2nd edn. Springer-Verlag, Berlin, Heidelberg, Germany.

Larock, B. E., Jeppson, R. W. & Watters, G. Z. 2000 *Hydraulics of Pipeline Systems*. CRC Press LLC, Boca Raton, Florida, USA.

Marques, J., Cunha, M. C., Sousa, J. & Savić, D. 2012 Robust optimization methodologies for water supply systems design. *Drinking Water Eng. Sci.* **5**, 31–37.

Montesinos, P., Guzman, A. G. & Ayuso, J. L. 1999 Water distribution network optimization using a modified genetic algorithm. *Water Resour. Res.* **35** (11), 3467–3473.

Neelakantan, T. R. & Suribabu, C. R. 2005 Optimal design of water distribution networks by a modified genetic algorithm. *J. Civil Environ. Eng.* **1** (1), 20–34.

Ostfeld, A., Uber, J. G., Salomons, E., Berry, J. W., Hart, W. E., Philips, C. A., Watson, J., Dorini, G., Jonkergouw, P., Kapelan, Z., di Pierro, F., Khu, S., Savic, D., Eliades, D., Polycarpou, M., Ghimire, S. R., Barkdoll, B. D., Gueli, R., Huang, J. J., McBean, E. A., James, W., Krause, A., Leskovec, J., Isovitsch, S., Xu, J., Guestrin, C., Van Briesen, J., Small, M., Fischbeck, P., Preis, A., Propato, M., Piller, O., Trachtman, G., Wu, Z. Y. & Walski, T. 2008 The battle of the water sensor networks (BWSN): A design challenge for engineers and algorithms. *J. Water Resour. Plann. Manage.* **134** (6), 556–568.

Rossman, L. A. 2007 Discussion of 'solution for water distribution systems under pressure-deficient conditions' by Wah Khim Ang and Paul W. Jowitt. *J. Water Resour. Plann. Manage.* **133** (6), 566–567.

Savic, D. A. & Walters, G. A. 1997 Genetic algorithms for least cost design of water distribution networks. *J. Water Resour. Plann. Manage.* **123** (2), 67–77.

Smith, L., Liang, Q. & Quinn, P. 2013 A flexible hydrodynamic modelling framework for GPUs and CPUs: Application to the Carlisle 2005 floods. In: *Proc. International Conference on Flood Resilience: Experiences in Asia and Europe, 5–7 September*, Centre for Water Systems, Exeter, UK.

Stanić, M., Avakumović, D. & Kapelan, Z. 1998 Evolutionary algorithm for determining optimal tree layout of water distribution networks. In: *Hydroinformatics 98* (V. Babovic & L. C. Larsen, eds). Balkema, Rotterdam, The Netherlands, pp. 901–910.

Stanić, M., Ivetić, D., Vasilić, Ž. & Prodanović, D. 2012 Improvement of application of genetic algorithms in optimization of pressurized pipe networks. In: *Proc. 16th Conference of Society of Serbian Hydraulic Engineers*, Donji Milanovac, Serbia, University of Belgrade – Faculty of Civil Engineering (In Serbian).

Suribabu, C. R. 2010 Differential evolution algorithm for optimal design of water distribution networks. *J. Hydroinform.* **12** (1), 66–82.

Todini, E. & Pilati, S. 1987 A gradient method for the analysis of pipe networks. In: *International Conference on Computer Applications for Water Supply and Distribution*. Leicester Polytechnic, Leicester, UK.

Todini, E. & Rossman, L. A. 2013 Unified framework for deriving simultaneous equation algorithms for water distribution networks. *J. Hydraul. Eng.* **139**, 511–526.

Van de Geijn, R. A. 1997 *Using PLAPACK: Parallel Linear Algebra Package*. MIT Press, Cambridge, MA, USA.

von zur Gathen, J. 1993 Parallel linear algebra. In: *Synthesis of parallel algorithms* (J. H. Reif, ed.). Morgan and Kaufmann, Los Altos, CA, USA, pp. 574–617.

Wang, Q., Craeco, E., Franchini, M., Savić, D. & Kapelan, Z. 2015a Comparing low and high-level hybrid algorithms on the two-objective optimal design of water distribution systems. *Water Resour. Manage.* **29** (1), 1–16.

Wang, Q., Guidolin, M., Savic, D. & Kapelan, Z. 2015b Two-objective design of benchmark problems of a water distribution system via MOEAs: Towards the best-known approximation of the true Pareto front. *J. Water Resour. Plann. Manage.* **141** (3). doi:10.1061/(ASCE)WR.1943-5452.0000460.

Zecchin, A., Thum, P., Simpson, A. & Tischendorf, C. 2012 Steady-state behavior of large water distribution systems: Algebraic multigrid method for the fast solution of the linear step. *J. Water Resour. Plann. Manage.* **138** (6), 639–650.

First received 31 January 2015; accepted in revised form 16 July 2015. Available online 13 August 2015