An ASP Approach for the Valves Positioning Optimization in a Water Distribution System

Marco Gavanelli, Maddalena Nonato, Andrea Peano, Stefano Alvisi, and Marco Franchini

EnDiF, Università degli Studi di Ferrara
via G. Saragat 1 – 44122, Ferrara, Italy

Abstract. Positioning of valves is a real-life issue in Water Distribution System design and, currently, it is usually addressed by hand by hydraulic engineers, or by means of genetic algorithms, that give no assurance of optimality. Since a given valves placement identifies a sectorization of the WDS in several isolable portions, the valves positioning problem can be seen as a variant of the well known graph partitioning, which is a hard combinatorial problem. [2] showed recently that Computational Logic can provide technologies and techniques that can be exploited to model and achieve the optimal partition of the water network (i.e., the optimal positioning of valves). In particular, they tackled the optimization of the valves positioning through a two player game model, giving a Constraint Logic Programming formalization to solve it effectively. The aim of this paper, instead, is to investigate the potential of Answer Set Programming in this practical application; evaluation is in terms both of language expressivity and solving efficiency. Results are discussed for different ASP models and a comparison with the CLP(FD) technique shown by [2] will be given.

1 Introduction

During the design of a water distribution network, one of the choices is the design of the isolation system. It is a real-life problem for hydraulic engineers, and in recent years it has been studied through computational methods in the *hydroinformatics* literature [13].

A water distribution system has the main objective of providing water to homes and facilities that require it. The water distribution network can be thought as a labelled indirected graph, in which the edges represent the pipes in the network. There is at least one special node that represents the source of water (node 1 in Figure 1), and the users' homes are connected to the edges. For each edge, we assume to have knowledge about the average amount of water (in litres per second) that is drawn by the users insisting on that edge (during the day); such value is the label associated to the edge, and it is called the users' *demand*.

The isolation system is mainly used during repair operations: in case some pipe is damaged, it has to be fixed or substituted. However, no repair work can

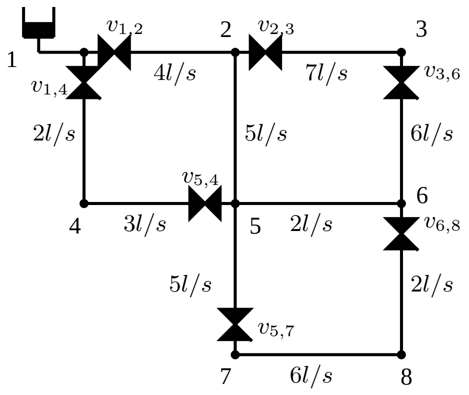

Fig. 1. A water distribution network with valves

be done while the water is flowing at high pressure in the pipe: first the part of the network containing the broken pipe should be de-watered, then workers can fix the pipe. The de-watering is performed by closing a set of *isolation valves*, that make up the so-called *isolation system* of the water distribution network. For example, in Figure 1, if the edge connecting nodes 2 and 3 (let us call it $e_{2,3}$) is broken, workers can close valves $v_{2,3}$ and $v_{3,6}$ and de-water the broken pipe. Of course, during this pipe substitution the users that take water from edge $e_{2,3}$ cannot be serviced. The usual measure of disruption is the *undelivered demand*: in this case, it corresponds to the demand of the users insisting on the broken pipe, namely $7l/s$.

However, we are not always this lucky: in case the damaged pipe is $e_{7,8}$, workers will have to close valves $v_{5,7}$ and $v_{6,8}$, de-watering pipes $e_{7,8}$ and $e_{6,8}$, with a total cost of $6 + 2 = 8l/s$. In fact, the minimum set of pipes that will be de-watered is that belonging to the so-called *sector* of the broken pipe, i.e., the set of pipes encircled by a same set of valves. But there can be even worse situations: if the broken pipe is $e_{2,5}$, workers have to close valves $v_{1,2}$ and $v_{5,4}$, which means disconnecting all the pipes except $e_{1,4}$ and $e_{4,5}$, with an undelivered demand of $4+5+7+6+2+5+6+2 = 37l/s$. Notice in particular that the edges $e_{2,3}$, $e_{7,8}$ and $e_{6,8}$ are disconnected in this way, although they do not belong to the same sector as the broken pipe. This effect is called *unintended isolation*, and usually means that the isolation system was poorly designed.

One common value used by hydraulic engineers [13] to measure the quality of the isolation system is the undelivered demand in the worst case. In the example of Figure 1, the worst case happens when the broken pipe is in the set $\{e_{1,2}, e_{2,5}, e_{5,6}, e_{5,7}\}$; in this case, as we have seen, the undelivered demand is $37l/s$.

In a previous work, [2] developed a system, based on Constraint Logic Programming [16] on Finite Domains (CLP(FD)), that finds the optimal positioning of a given number of valves in a water distribution network. The assignments found by [2] improved the state-of-the-art in hydraulic engineering for this problem, finding solutions with a (worst-case) undelivered demand lower than the best solutions known in the literature of hydraulic engineering [13], obtained through genetic algorithms.

In this work, we address the same problem in Answer Set Programming [1, 21, 10], and evaluate pros and cons of the two solutions.

The rest of the paper is organized as follows. In Section 2, we provide the formal definition of the problem, as given in [2]. Section 3 contains two ASP formulations of the valve placement problem. In Section 4 we present experimental results on a real-life network, taken from the hydraulic engineering literature [13]. We discuss related work in Section 5, and, finally, we conclude.

2 Problem Description

A water distribution network is modelled as a weighted indirected graph $G \equiv (N, E)$, where $N = \{1, \dots, n\}$ is a set of nodes and $E = \{e_{ij}\}$ is a set of edges. Each edge e_{ij} has an associated weight $w(e_{ij})$ called *demand*. In the network, there are some nodes identified by the set Σ that are called *sources*. Valves can be positioned near one of the ends of a pipe; we will refer to valve on edge e_{ij} near to node i as v_{ij}, while v_{ji} is a valve on the same edge, but close to node j.

Given a number N_v of valves, the objective is to position the valves in the network such that:

1. it is possible to isolate any pipe in the network. Formally, given an edge e_{ij}, it is possible to identify a minimal set of valves C to be closed such that there is no path from any source node $s \in \Sigma$ to the edge e_{ij} that does not contain a valve $v \in C$. Since the set C of valves to be closed depends on the damaged pipe e_{ij}, we will also write $C(e_{ij})$. Note that there is only one reasonable set $C(e_{ij})$ of valves to be closed given a broken edge e_{ij}: intuitively only the valves directly reachable from e_{ij} will be closed.
2. the objective is to minimize the maximum undelivered demand (UD). Formally, let $D(C)$ be the set of edges that do not receive water when the valves in C are closed, i.e., those edges for which there is no path from any source node to the edge: $D(C) = \{e_{ij} \in E | \forall s \in \Sigma, \nexists Path(s, e_{ij})\}$. The objective function to be minimized is

$$UD = \max_{e_{ij} \in E} \sum_{e_{kl} \in D(C(e_{ij}))} w(e_{kl}).$$

3 ASP Formulations for the Valves Positioning Problem

The ASP approach to treat a computational problem consists of defining a logic program that models the solutions of the said problem through its answer sets [11].

We present two different approaches to the Valve Positioning Problem (VPP). In one of the approaches, we explicitly define a concept of “sector”, while in the other the same concept is left implicit. We first define the common parts of the two approaches (Section 3.1), then we present the parts specific to the two ASP programs (Sections 3.2 and 3.3).

The input data consists of a set of facts that describe the graph of the water distribution network. Nodes are given as facts `node(X)`, while labelled edges are facts `edge(I,J,D)`, where `I` and `J` are nodes of the graph, and `D` is the demand associated to the edge. The sources of water are usually tanks, and they are given as nodes `tank(N)`, where `N` is the name of the node.

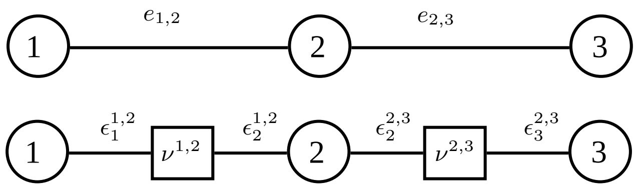

Fig. 2. Example of graph with the corresponding Extended Graph.

In order to simplify the definition of the ASP program, we rely on an Extended Graph. The extended graph is built by adding to each edge $e_{i,j}$ of the original graph a fictitious node $\nu^{i,j}$ that intuitively represents the demand from the users. The added node $\nu^{i,j}$ *"splits"* the edge $e_{i,j}$ into two parts: one connects node n_i with the the new node $\nu^{i,j}$, and the other connects the new node with node n_j, as shown in Figure 2.

Definition 1. *Given a graph* $G(N, E)$*, the* Extended Graph $\bar{G}(\bar{N}, \bar{E})$ *is defined as follows:*

– $\bar{N} = N \cup \{\nu^{i,j} | e_{i,j} \in E\}$
– $\bar{E} = \{\epsilon_i^{i,j}, \epsilon_j^{i,j} | i, j \in N, e_{i,j} \in E\}$

We will name ϵ-edge *each* $\epsilon_i^{i,j}$ *and* $\epsilon_j^{i,j}$*, and* ν-node *each* $\nu^{i,j}$*. We also add weights to the* ν*-nodes:* $w(\nu^{i,j}) := w(e_{i,j})$*, while the original nodes in the graph do not have a weight* ($w(n_i) := 0$)*.*

With this definition, each ϵ-edge can host up to one valve, so defining a partition through valves amounts to define a set of ϵ-edges to be removed from the Extended Graph. Moreover, the demand is only on the nodes, so it is easy to compute the demand of a partition by summing up the weights of the nodes inside that partition.

The Extended Graph is represented through predicates `nu` and `eps` that represent respectively its nodes and edges. Since each node $\nu^{i,j}$ of the extended graph corresponds to an edge of the original graph, we keep the same naming convention: for each $e_{i,j} \in E$, there exists a ν-node $\nu^{i,j}$:

```
nu(I,J) :- e(I,J,D).
```

The arguments of `nu` are the two extremes of the edge hosting the ν-node.

The ϵ-edges are represented with predicate `eps`. Each ϵ-edge connects one of the original nodes to one ν-node; there are two ϵ-edges for each edge of the original graph (or, equivalently, for each ν-node):

```
eps(I, nu(I,J)) :- nu(I,J).
eps(J, nu(I,J)) :- nu(I,J).
```

In this way, the graph partitions consist of weighted ν-nodes plus some weightless junction nodes (the nodes of the original graph), and the valve placement choice no longer lies on each vertex-side of each edge but it lies over the simple set of ϵ-edges.

3.1 A Basic ASP Program for Valve Positioning

As we said in the problem description (Section 2), each ν-node should be isolable by cutting a subset of ϵ-edges of the extended graph. The cut edges identify the set of valves of the hydraulic network which (when closed) isolate the broken pipe. We are interested in finding the set of valves in the network. As in Figure 1, we use the convention that $v_{i,j}$ means that the valve is on edge $e_{i,j}$ closer to edge i, while $v_{j,i}$ means that the valve is on the same edge, but closer to node j. We define a predicate `valve/2` with the same convention. We generate the possible valves depending on the edges of the network; the number of valves should be exactly N_v:

N_v `{ valve(A,B) : e(A,B), valve(B,A) : e(A,B) }` N_v`.`

The symbol ":" is a conditional operator and, in this case, it instantiates as many `valve` atoms as the possible groundings of facts `e`.

We are able to isolate network patches by closing the valves. However, the set of closed valves depends on where the damaged pipe is. We define a predicate `closed_valve/2`. The meaning that

`closed_valve(`$\epsilon_a^{a,b}$`,`$\nu^{x,y}$`)`

is true iff the valve that is on the ϵ-edge $\epsilon_a^{a,b}$ will be closed when the pipe $\nu^{x,y}$ is broken. The generation of the possible values for closed valves is as follows; given a (tentatively broken) ν-node $\nu^{X,Y}$, the number of possible valves that can be closed ranges from 1 to the maximum number of valves N_v:

```
1 {
  closed_valve(eps(A,nu(A,B)), nu(X,Y)) : nu(A,B),
  closed_valve(eps(B,nu(A,B)), nu(X,Y)) : nu(A,B)
} N_v :- nu(X,Y).
```

If a valve is closed (for at least one broken pipe $\nu^{X,Y}$), then there must be a valve in such position, so we link predicates `closed_valve/2` and `valve/2`:

```
valve(A,B) :- closed_valve(eps(A, nu(A,B)), nu(X,Y)).
valve(B,A) :- closed_valve(eps(B, nu(A,B)), nu(X,Y)).
```

Up to now, the ASP program assures that for each (damaged) ν-node there exists a subset of the installed valves that will be closed, but there is no knowledge of which users (or, which ν-nodes) will be reached by the water in each situation. We define a predicate `reached/2`, that explains which ν-nodes $\nu^{A,B}$ are reached by the water when node $\nu^{X,Y}$ is damaged:

```
reached(nu(A,B), nu(X,Y))
```

The ν-node $\nu^{A,B}$ is reached by the water if one of the endpoints of its edge is a tank, and between the two there is no valve, or there is a valve but it is not closed when the damaged node is $\nu^{X,Y}$ (Figure 3).

```
reached(nu(A,B), nu(X,Y)) :-
  nu(A,B), tank(A),
  not closed_valve(eps(A,nu(A,B)), nu(X,Y)).
```

Otherwise, $\nu^{A,B}$ is reached by the water if (at least) one of its adjacent ν-nodes is reached and no valves are closed between the two (again, when the damage is in ν-node $\nu^{X,Y}$). Figure 4 represents the reachability of a generic ν-node $\nu^{A,B}$, that is reachable if its adjacent $\nu^{Z,A}$ is, in turn, reached and if the hypothetical valves between them are not closed.

```
reached(nu(A,B), nu(X,Y)) :-
  nu(A,B), nu(X,Y), nu(Z,A),
  not closed_valve(eps(A,nu(Z,A)), nu(X,Y)),
  not closed_valve(eps(A,nu(A,B)), nu(X,Y)),
  reached(nu(Z,A), nu(X,Y)).
```

Finally, the broken pipe should not be reachable by water:

```
:- reached(nu(X,Y), nu(X,Y)).
```

Frequently, in the hydraulic networks some junction nodes link only two pipes. Of course, in such a case, there is no point in adding two valves on the two edges, since isolating a junction does not make sense, as the demand is only on edges. This also means that placing a valve at one side of such nodes rather than the other side leads to two equivalent solutions. This kind of symmetry can be avoided, obtaining a possible reduction of the search space, through the definition of a further integrity constraint, as follows:

```
symm_e(X,Y) :- e(X,Y,D).
symm_e(Y,X) :- e(X,Y,D).
:- node(X), not tank(X), symm_e(X,A), symm_e(X,B),
   2 { symm_e(X,Y) } 2, A>B, valve(X,A).
```

The above integrity constraint states that if a junction node `X` has degree 2 (i.e., the related set of symmetric edges has cardinality 2) and `X` is not a tank, then we can impose that in one given side there must be no valve.

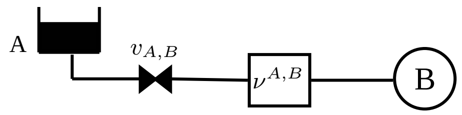

Fig. 3. Reachability of a node through a tank

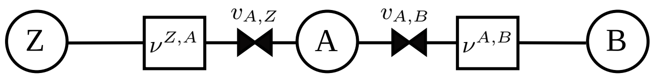

Fig. 4. Reachability of a node through an adjacent node

3.2 Optimal Placement Specialization Based on Paths

As explained earlier, the objective is to maximize the satisfied demand (or, equivalently, minimize the unsatisfied demand) in the worst case.

As a first attempt, we might use the ♯sum aggregator operator [5], and define the satisfied water demand (when some ν-node $\nu^{X,Y}$ is broken) as the sum of the demands of the ν-nodes reached by water:

```
sat(Dsat, nu(X,Y)) :- nu(X,Y),
  Dsat = #sum[reached(nu(A,B), nu(X,Y))=D : e(A,B,D)].
```

Unfortunately, this type of aggregation leads to an explosion of the ground program, especially if the single demands `D` can take large integer values. Another way to find the minimum (total) satisfied demand is by means of pairwise comparisons amongst `reached/2` atoms, varying the "broken" ν-nodes, using the aggregator ♯`sum` as a conditional operator. Predicate `cmp(`$\nu^{X,Y}$`,`$\nu^{W,Z}$`)` is true if the satisfied demand is lower in case the edge $\nu^{X,Y}$ is broken, than it is when the broken edge is $\nu^{W,Z}$. It is defined as follows:

```
cmp(nu(X,Y), nu(W,Z)) :- nu(X,Y), nu(W,Z),
     #sum[ reached(nu(A,B), nu(X,Y)))=Dn : e(A,B,Dn),
           reached(nu(C,D), nu(W,Z)))=-Dm : e(C,D,Dm)]0.
```

In this case, the ♯`sum` aggregator sums with a positive sign the satisfied demand when the broken edge is the first ($\nu^{X,Y}$), and with a negative sign the satisfied demand when the broken edge is the second ($\nu^{W,Z}$). The body evaluates to true if this algebraic sum is less than or equal to 0. In this case, the grounder instantiates rule `cmp/2` for each pair of ν-nodes, so that we move to the solver the task to check if the sum is really less or equal than 0 (i.e., if the first term of `cmp/2` –a ν-node isolation– determines a satisfied demand less or equal than the second one) and it will return answer sets containing only the actual comparisons.

We can now compute the minimum of the satisfied demands, varying the broken node: the minimum is the one that is less than or equal to all the other ones.

```
min(nu(X,Y)) :- nu(X,Y),
  cmp(nu(X,Y),nu(W,Z)) : nu(W,Z).
```

At this stage, we know which damaged pipe gives the minimum satisfied demand; one could think to compute the total satisfied demand and use it as an objective function for maximization, however it is not possible to compute it by this intuitive formula:

```
sat_min(nu(A,B), min(nu(X,Y)), D):-
  reached(nu(A,B), nu(X,Y)), min(nu(X,Y)), e(A,B,D).
#maximize[ sat_min(nu(A,B),min(nu(X,Y)),D)=D ].
```

because the minimum is not unique, as there can be two or more sectors that, when isolated, provide the same delivered demand. Also, the sector corresponding to minimal satisfied demand may contain more than one pipe: if one of those pipes is broken, they will all provide the same delivered demand. When the minimum is not unique, with the above program we would sum one contribution for each of the equally good minima. In order to compute the correct delivered demand, we select one of the minima. To select a unique minimum, we use a lexicographic comparison: after comparing the delivered demand, we compare the names of the nodes inside a sector, obtaining only one node, given by the atom `unique`, that (when broken) provides the minimum satisfied demand. Thus, we can define the satisfied ν-nodes of the unique minimum, as follows:

```
min_sat(nu(X,Y), D) :-
 reached(nu(X,Y), nu(A,B)),
 unique(min(nu(A,B))), e(X,Y,D).
```

Finally, summing up the weights of the argument of any `min_sat/2` and maximizing it through the operator `♯maximize`, we find out the answer set that represents the optimal valves positioning, i.e. that one for which the satisfied water demand of the worst ν-node isolation is maximized, as follows:

```
#maximize [ min_sat(nu(X,Y), D)=D ].
```

3.3 Optimal Placement Specialization Based on Extended Sectors

The paths-based specialization for the optimization of the valve positioning, described in the previous section, does not explicitly use sectors, but it maximizes the satisfied water demand of the worst ν-node isolation.

In this section, instead, we give a further specialization of the program shown in Section 3.1, where sectors are explicitly defined by means of two steps: in the first, we generate the possible sectors, and then we state that any ν-node must belong at least to a sector. It is important to notice that each valve can increase the number of sectors of at most *one* unit. This bound is strict, and it happens, e.g., if the network graph is actually a tree. Accordingly, the maximum number of sectors is limited to the number of valves.

In the standard conception of sector, we should limit their number per ν-node to 1, since one ν-node belongs to exactly one sector; here we refer as *extended sector* to the set of unreachable ν-nodes given a ν-node isolation. In fact, a ν-node isolation could be due to a direct or a indirect effect (the effect of *unintended* *isolation* explained in the Introduction). E.g., for the hydraulic network shown in Figure 1 the worst case of unsatisfied water demand is due to the isolation of one among the possibly broken ν-nodes $\{\nu^{1,2}, \nu^{2,5}, \nu^{5,6}, \nu^{5,7}\}$; such isolation determines the unintended disservice for $\{\nu^{2,3}, \nu^{3,6}, \nu^{6,8}, \nu^{7,8}\}$. Accordingly, the related extended sector is the union of these two sets, whereas the extended sector of the broken node $\nu^{7,8}$ is merely composed of $\{\nu^{6,8}, \nu^{7,8}\}$.

```
s(1..N_v).
1 { sector(nu(A,B),S) : s(S) } N_v :- nu(A,B).
```

The predicate `sector/2` says that the ν-node $\nu^{A,B}$ (argument 1) belongs to the (extended) sector `S` (argument 2). Two ν-nodes belong to the same extended sector if whenever one is unreachable, the other one is unreachable as well:

```
sector(nu(A,B),S) :- nu(A,B), sector(nu(C,D),S),
  not reached(nu(A,B),nu(C,D)).
```

In this case, if $\nu^{A,B}$ is not reachable for an indirect side effect of the isolation of $\nu^{C,D}$, then $\nu^{A,B}$ belongs to two or more different extended sectors and at least one of which is in common with $\nu^{C,D}$.

In order to find out the sector that determines the maximum service disruption if isolated, we proceed with pairwise comparisons among all sectors, similarly as in Section 3.2. More precisely, the two following rules state that a sector is empty if no ν-nodes are assigned to it and that a sector `S1` is greater than `S2` if it is not empty and the sum of the weights of its ν-nodes is greater than the sum of weights of `S2`:

```
empty(S) :- s(S), not sector(nu(A,B),S) : nu(A,B).
cmp(S1,S2) :- s(S1), s(S2), S1!=S2, not empty(S1),
  0 #sum[sector(nu(A,B),S1)=Dn : nu(A,B): e(A,B,Dn),
         sector(nu(C,D),S2)=-Dm : nu(C,D) : e(C,D,Dm)].
```

The worst disservice is determined by the sector `S1` for which the predicate `cmp(S1,S2)` is true for each other `S2`, with `S1`$\neq$`S2`, and we identify it by the predicate `maxSect/1`, where the argument is the sector name, as follows:

```
maxSect(S1):- s(S1), not empty(S1),
  cmp(S1,S2) : S1!=S2 : s(S2).
```

As stated in Section 3.2, two or more extended sectors may determine the same worst disservice, so we select the existing `maxSect/1` with the greatest name value; finally, we minimize the sum of those ν-node weights belonging to the sector which gives rise to the maximum service disruption (and that has the greatest name value):

```
bestMax(S) :- S=#max[ maxSect(S1)=S1 ].
maxUnsatDem(nu(A,B),D) :-
  sector(nu(A,B),S), bestMax(S), e(A,B,D).
#minimize [ maxUnsatDem(nu(A,B),D)=D ] .
```

4 Computational Results

As above mentioned, the two ASP programs, described in Section 3, are implemented with the native syntax of the Potsdam Answer Set Solving Collection (Potassco) [9]. Such collection of tools includes, among all, the grounder *Gringo* and the conflict-driven ASP solver *Clasp*. The solver *Clasp* can be "finely" tuned, by working on parameters related to the preprocessing and solving processes [9]. Beside the default configuration, the Potassco team suggests specific configurations for many types of combinatorial problems, used during the ASP Competition 2009[1]. Among such *Clasp* configurations, one is tuned to treat Graph Partitioning (GP) problems and, perhaps due to the similarity with the valve positioning problem, we will show that the GP configuration can improve the solving computation time.

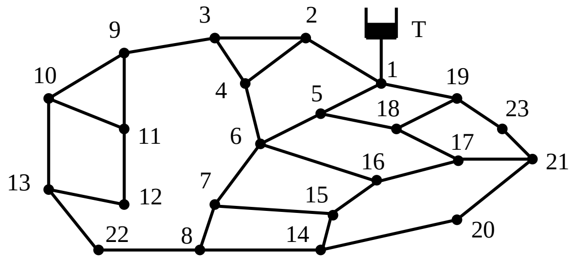

Fig. 5. The apulian water distribution network

We tested the two logic programs on the water distribution system in [12, 13], that represents the distribution network of the Apulia Italian region, depicted in Figure 5. It is worth noting that 4 nodes, namely $\{12, 20, 22, 23\}$, are junction nodes of degree 2 (exactly two pipes meet in each of these nodes) and, as said in Section 3.2, adding a symmetry breaking integrity constraint helps reducing the search space.

The experiments have been performed on a *Intel* dual core architecture based on P8400 CPUs, 2.26 GHz and 4GB of RAM; however, although Potassco provides a parallel ASP solver, we used only one core.

First, we show in Figure 6 the solving times of the two ASP programs (the sectors-based one and the paths-based one) when symmetry breaking is either used or not. In general, it is clear that, for the Apulian instance of the valve positioning problem, the sectors based program reaches the optimality in a computation time lower than the one based on paths. Moreover, for 6 valves the gap between the two models is really huge. In particular, the symmetry breaking leads to a performance improvement for the sectors based ASP program and to a worsening for the paths based one.

[1] http://dtai.cs.kuleuven.be/events/ASP-competition/Teams/Potassco.shtml

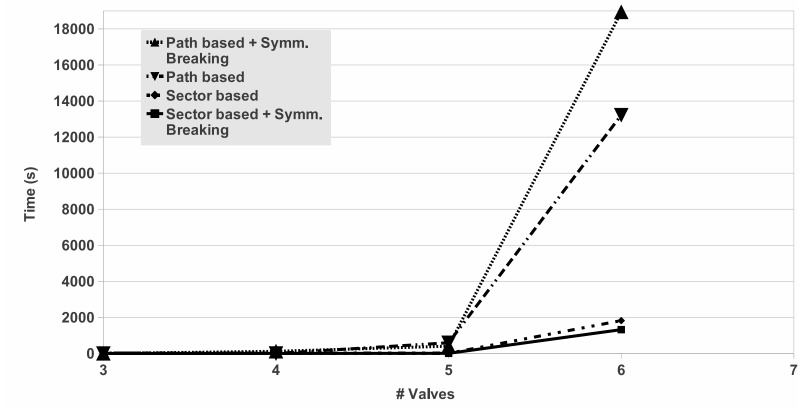

Fig. 6. Computing times of programs solving if symmetry breaking is either used or not

To give a general view about the incidence of the two different *Clasp* configurations (the default and the GP one) and of the symmetry breaking constraint, we plot in Figure 7 the performance of four different solving runs, obtained by using the sectors based program and by mixing the above described customizations. It is clear that both the symmetry breaking and the *Clasp* configuration for graph partitioning problems enhance the performance of the optimization process; nevertheless, the latter seems to have a higher incidence. The paths based program solving processes present the same behaviour, but with much higher computational times.

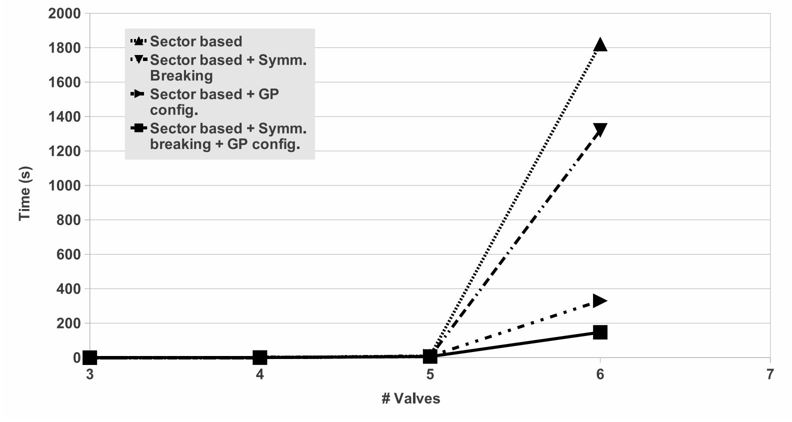

Fig. 7. Computation times of the sectors based program

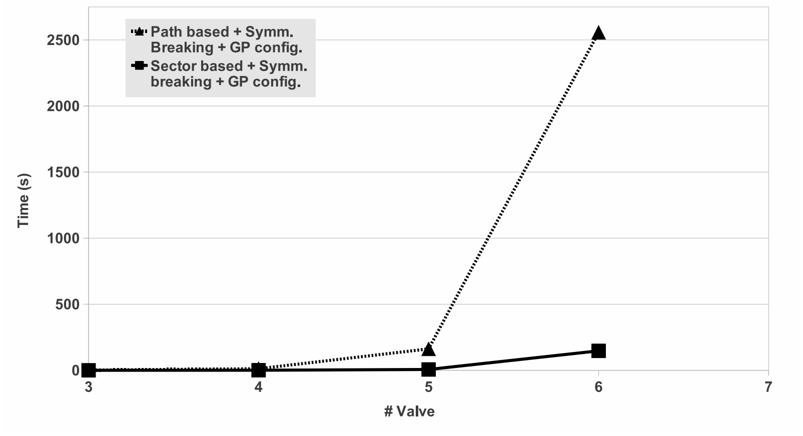

Fig. 8. Computing times of the two ASP models if both customizations are used

The comparison between the two different ASP models' computation times, if both symmetry breaking constraint and the GP configuration of *Clasp* are tuned, is shown in Figure 8. Based on the chart in Figure 8, we can state that the sectors based ASP model finds out more quickly the optimal valves positioning. Even so, computational results achieved with the CLP(FD) formulation discussed in [2] is still better than the ones obtained with our ASP programs; in fact, while the optimal solution with 7 valves is achieved in about 4 hours by our best ASP program, the above mentioned CLP(FD) program computes the optimum in a few seconds.

We must underline that both of the ASP programs presented here consist of respectively 20 and 25 clauses, against the hundreds needed to solve the same problem in CLP(FD). Hence, the ASP approach usually permits rapid prototyping of combinatorial problems, with lower implementation cost and time; in fact, in a real-life infrastructure design context like this, there is no need for real-time solving and the costs of person-hour are much more valuable than the computation time of a machine.

5 Related Work

In the literature of hydraulic engineering, two main problems related to the isolation valves in a pipe network have been faced, that is a) the identification of the segments and undesired disconnections that occur after a set of isolation valves has been closed and b) the (near) optimal location of the set of isolation valves. As far as the first topic is concerned, in the literature there are a number of studies regarding segment identification and the undesired disconnections that occur following the closure of a set of isolation valves. In particular, the methods proposed by [17] and [18] are based on a dual representation of the network, with segments treated as nodes and valves as links. The methods proposed by [4] and

[13] use topological incidence matrices to identify the segments. As far as the second topic is concerned, recently, [13] and [4] have proposed two different multi-objective optimization approaches, both based on genetic algorithms; the first minimizes the number of valves ensuring a fair compromise between the costs of the valves and the system reliability in the event of routine and non-routine maintenance, while the second one minimizes the costs and the undelivered water demand given a number of available valves. All of these works use incomplete algorithms, that cannot ensure that the found solution is the real optimum. To the best of our knowledge, [2] present the first complete algorithm to address the valve placement problem.

The valve placement problem has some similarities with the graph partitioning problem, in which the goal is to partition a graph into (almost) equal-size parts. In general, graph partitioning is NP-hard [8]. Most works in the literature deal with heuristics or approximation algorithms and one of the first works in the area is by [20], that propose a greedy algorithm which outputs a graph bisection. [6] improve the algorithm so that the asymptotic behaviour of the algorithm is linear rather than quadratic. A different approach is based on the *spectral* analysis of the graph, discussed in [3, 7, 14, 24]. In comparison with other heuristics, spectral methods provide good quality partitions at an increased computational cost (necessary to compute the matrix eigenvalues). Moreover, various kinds of heuristics can be used if multilevel schemes are exploited, as described in [15] and [19].

The special case of *planar* graphs (i.e. graphs which can be drawn without intersecting edges) is of particular interest for our application since it is often the case for water supply networks. Finding the optimal solution is NP-hard also for the planar case, however the *planar separator theorem* [22] states that a bisection in which the biggest set contains at most two thirds of the vertices and whose separator contains $O(\sqrt{n})$ vertices can be found in linear time.

Other related problems are the multicut problems [23], in which the aim is to find the minimal set of edges (or nodes) such that given pairs of nodes are no longer connected. In our case, instead, the aim is to disconnect a possibly small part of the network while keeping connected all the rest.

The algorithms for graph partitioning or solving multicut problems are clearly not directly applicable to the valve placement problem, also because of the issue of unintended isolation mentioned in Section 1.

6 Conclusions

In this work, we presented two ASP formulations for the valve placement problem, a problem taken from the literature of hydraulic engineering [13], and for which a CLP(FD) model was proposed in [2]. The two ASP formulations were mainly developed by a first-year PhD student that was not an ASP expert (his main background was on Operations Research, although he had some knowledge of Prolog and CLP) in about one week. This shows that ASP is very intuitive and easy to understand even for non experts, that it is indeed very declarative, and that it can be used to address real-life problems taken from subject areas apparently very distant from Logic Programming. The two ASP programs consist of respectively about 20 and 25 clauses, which shows that ASP is a very interesting technology for rapid prototyping.

The experiments show that the developed models take more computation time than a CLP(FD) approach. However, we must say that the CLP(FD) model was developed by two CLP experts, during some person-months and was trimmed for efficiency. Since person-months are largely more costly than CPU time, and since the given application does not require results in strict real-time (as it is to be executed during the design of the hydraulic network), ASP could be an effective solution for this type of applications. Another advantage of ASP stands in the fact that existing solvers are improved all the time, and new solvers are developed every year, so the efficiency of an ASP program improves every year, requiring little (if any) modifications to be adapted to new solvers.

In future work, we plan to continue the development of new ASP models, and to experiment them with other available ASP solvers. We are also interested in trying to integrate the ASP models with a CLP approach, to take advantage of the strengths of the two approaches. Finally, we plan to submit the problem instances to the next ASP competitions, so that new ASP models can be developed and solvers can be improved also to solve these types of applications.

Acknowledgements We wish to thank Roland Kaminski for suggesting how to reduce the explosion of the ground program.

This work was partially supported by EU project *ePolicy*, FP7-ICT-2011-7, grant agreement 288147. Possible inaccuracies of information are under the responsibility of the project team. The text reflects solely the views of its authors. The European Commission is not liable for any use that may be made of the information contained in this paper.

References

1. Baral, C.: Knowledge representation, reasoning and declarative problem solving. Cambridge University Press (2003)
2. Cattafi, M., Gavanelli, M., Nonato, M., Alvisi, S., Franchini, M.: Optimal placement of valves in a water distribution network with CLP(FD). Theory and Practice of Logic Programming 11(4-5), 731–747 (2011), http://arxiv.org/abs/1109.1248
3. Chung, F.R.K.: Spectral Graph Theory (1994)
4. Creaco, E., Franchini, M., Alvisi, S.: Optimal placement of isolation valves in water distribution systems based on valve cost and weighted average demand shortfall. Journal of Water Resources Planning and Management 24(15) (2010)
5. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In: Alferes, J., Leite, J. (eds.) JELIA 2004. pp. 200–212. No. 3229 in Lecture Notes in Artificial Intelligence, Springer Verlag (2004)
6. Fiduccia, C.M., Mattheyses, R.M.: A linear-time heuristic for improving network partitions. In: Proceedings of the 19th Design Automation Conference. pp. 175–181. DAC '82, IEEE Press, Piscataway, NJ, USA (1982), http://portal.acm.org/citation.cfm?id=800263.809204

7. Fiedler, M.: Algebraic connectivity of graphs. Czechoslovak Mathematical Journal 23(98), 298–305 (1973)
8. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1990)
9. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The Potsdam answer set solving collection. AI Communications 24(2), 105–124 (2011)
10. Gelfond, M.: Answer sets. In: Handbook of Knowledge Representation, chap. 7. Elsevier (2007)
11. Gelfond, M., Lifschitz, V.: Compiling circumscriptive theories into logic programs. In: Shrobe, H.E., Mitchell, T.M., Smith, R.G. (eds.) AAAI. pp. 455–449. AAAI Press / The MIT Press (1988)
12. Giustolisi, O., Savić, D.A.: Optimal design of isolation valve system for water distribution networks. In: Van Zyl, J., Ilemobade, A., Jacobs, H. (eds.) Proceedings of the 10th Annual Water Distribution Systems Analysis Conference WDSA2008 (2008)
13. Giustolisi, O., Savić, D.A.: Identification of segments and optimal isolation valve system design in water distribution networks. Urban Water Journal 7(1), 1–15 (2010)
14. Hendrickson, B., Leland, R.: An improved spectral graph partitioning algorithm for mapping parallel computations. SIAM Journal on Scientific Computing 16, 452–469 (March 1995), http://portal.acm.org/citation.cfm?id=203046.203060
15. Hendrickson, B., Leland, R.: A multilevel algorithm for partitioning graphs. In: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM). Supercomputing '95, ACM, New York, NY, USA (1995), http://doi.acm.org/10.1145/224170.224228
16. Jaffar, J., Maher, M.J.: Constraint logic programming: A survey. J. Log. Program. 19/20, 503–581 (1994)
17. Jun, H., Loganathan, G.V.: Valve-controlled segments in water distribution systems. Journal of Water Resources Planning and Management 133(2), 145–155 (March/April 2007)
18. Kao, J.J., Li, P.H.: A segment-based optimization model for water pipeline replacement. J. Am. Water Works Assoc. 99(7), 83–95 (2007)
19. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. 20, 359–392 (December 1998), http://dx.doi.org/10.1137/S1064827595287997
20. Kernighan, B., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Systems Technical Journal 49, 291–307 (1970)
21. Leone, N.: Logic programming and nonmonotonic reasoning: From theory to systems and applications. In: Baral, C., Brewka, G., Schlipf, J. (eds.) Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07), Lecture Notes in Computer Science, vol. 4483. Springer (2007)
22. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM Journal on Applied Mathematics 36(2), 177–189 (1979)
23. Pichler, R., Rümmele, S., Woltran, S.: Multicut algorithms via tree decompositions. In: Calamoneri, T., Díaz, J. (eds.) Algorithms and Complexity, 7th International Conference, CIAC 2010. Lecture Notes in Computer Science, vol. 6078, pp. 167–179. Springer (2010)
24. Spielman, A., Teng, S.H.: Spectral partitioning works: Planar graphs and finite element meshes. Technical Report. University of Berkeley (1996)