Math. Program., Ser. B (2010) 124:119–141
DOI 10.1007/s10107-010-0368-4

FULL LENGTH PAPER

Locating leak detecting sensors in a water distribution network by solving prize-collecting Steiner arborescence problems

Alain Prodon · Scott DeNegre · Thomas M. Liebling

Received: 3 June 2008 / Accepted: 10 August 2009 / Published online: 9 May 2010

Abstract We consider the problem of optimizing a novel acoustic leakage detection system for urban water distribution networks. The system is composed of a number of detectors and transponders to be placed in a choice of hydrants such as to provide a desired coverage under given budget restrictions. The problem is modeled as a particular Prize-Collecting Steiner Arborescence Problem. We present a branch-and-cut-and-bound approach taking advantage of the special structure at hand which performs well when compared to other approaches. Furthermore, using a suitable stopping criterion, we obtain approximations of provably excellent quality (in most cases actually optimal solutions). The test bed includes the real water distribution network from the Lausanne region, as well as carefully randomly generated realistic instances.

Keywords Prize-collecting Steiner problem · Branch and cut · Network leakage detection

Mathematics Subject Classification (2000) 90C27 · 90C57 · 90C90

1 Introduction

Leakages are well-known to be a major issue in all urban water distribution networks. At best, they will just cause sizeable water losses, while at worst, they can result in serious damage to people and buildings in the neighborhood where they occur.

A. Prodon (✉) · T. M. Liebling
EPFL-SB-IMA-ROSO, Station 8, 1015 Lausanne, Switzerland
e-mail: alain.prodon@epfl.ch

S. DeNegre
Industrial and Systems Engineering, Lehigh University, Bethlehem, PA 18015, USA

Therefore, network managers are keen on having a reliable monitoring system for early detection of water losses.

This paper considers the optimization of a network of acoustic water leakage sensors and accompanying radio relays, being tested by the city of Lausanne. Various such systems have been proposed in the past. The particular one underlying this paper is called *LORNO*, and is made of acoustic sensors placed at various hydrants and transponders that store and transmit the monitored and received information from other transponders to a central station. Each acoustic sensor hears problematic signals within a neighborhood defined by its placement and dependent on local network topology and geometry. Such a neighborhood must be estimated for each potential placement. Where there is a sensor, there is a transponder, but supplementary transponders may be needed to carry all information to the central station. For a given city, this gives rise to a family of combinatorial optimization problems. One example is: Given the set of hydrants and, for each of them, the neighborhood covered by a sensor placed there, find a minimum cost placement of sensors covering the whole network as well as of transponders enabling the corresponding information to be transmitted to the central station. Another version is: For a given budget, find a maximum utility placement of sensors and transponders, where utility is measured by the information that is transmitted and that can depend on the place from where it is collected. Some optimal placement problems in a water distribution network can simply be formulated as integer programs [see 2]. However, the problems at hand contain a hard constraint, as the transmission of the data to a central station requires that the solutions induce a connected subgraph of a given network. These connectivity constraints lead us to model the problems as variants of Steiner's problem, resp. Prize-Collecting Steiner's Problem, and we don't know any alternate way to model them directly in a Mixed Integer Program. These Steiner's problems are NP-hard, except on special graphs [e.g., 11], but polyhedral approaches like those described by [5–8] may help to find optimal or good approximate solutions.

In this paper, we present a novel branch-and-bound-and-cut approach for solving these problems and compare it with others from the literature [9]. We have tested our approach on real data from Lausanne water supply network. Furthermore, in order to provide better-founded empirical validation of our approach, we also tested it on specially constructed water supply systems, tailored to be realistic. We find that our approach is clearly competitive and, in many cases, the only one to solve those problems. Moreover, it can be implemented to provide approximate solutions with a guarantee on the optimality gap. The relevance of our study for decision makers is that it is possible to treat large size problems and obtain very good—and, in many cases, optimal—solutions. In particular, it makes it possible to carry out sensitivity analyses and variant studies that require repeated solutions of the underlying problem.

The remainder of the paper is structured as follows. First, we give a description of the *LORNO* system and mathematical formulations of the optimization problems. We next describe our approach for solving the Prize-Collecting Steiner Arborescence Problem. Finally, we present and discuss our computational results, both on real-world and realistically simulated models.

2 *LORNO* system and modeling

2.1 The problem

In the Lausanne water distribution network, water losses of approximately 20% have been reported. In other cities, this loss proportion can easily reach 50%.

Such losses may have many different causes, such as leaky or broken pipes or unregistered utilization (theft or even exercises performed by local fire departments). In a region where water supply is not a problem, these losses are not so important, with the exception of broken pipes, which may cause serious damage and related costs (traffic perturbation, floods, water distribution's break-downs). In a distribution system, leaks are often the best available sign that a pipe may break soon, making leak detection crucial for the manager.

The *LORNO* system has been developed to detect leaks in a water distribution network. It relies on recognizing the unusual noises that arise in the pipes due to the leaks. It consists of units placed in the hydrants and a central server connected to the manager's office. Thus, its installation does not require any heavy civil engineering work. Each unit consists of a listening device, integrated in the column inside a hydrant, and a radio transponder placed on the hydrant at the surface. The auditory function is realized by an acoustic sensor, which can hear the pipes around it up to a certain topology-dependent distance, coupled with electronic chips which analyze the sound signal, compare it with normal and historical curves, and send an alert or a periodic report. There is also a component that measures the amount of water drawn from the hydrant. These data are then transmitted through radio signals to the central server. In order to limit electricity consumption (energy is provided by a battery which should last for several years) low power transponders are used. These can only communicate with their neighbors located up to a certain distance—200–500 m, depending on the physical obstacles—and the signals are transmitted from hydrant to hydrant, through what we call the communication network, until the server is reached.

The engineers' thumb rule says that equipping half of the hydrants with full *LORNO* units (listening device and radio transponder) is enough for covering the pipe network for leak detection, and that one third of the hydrants equipped already provides a good coverage. In order to get a connected communication network, it is also possible to equip a hydrant with a radio transponder only. Thus, for each hydrant it should be decided whether to equip it with a full *LORNO* unit, a radio transponder only, or not to use it at all.

The following example (see Fig. 1) shows a middle size subnetwork of the Lausanne water distribution network. It contains 173 hydrants represented by stars. While edges in the first picture correspond to pipes, they indicate in the second picture that two hydrants can communicate.

The optimization problem we want to solve is then the following: Choose a subset of hydrants to equip with a full *LORNO* unit, and possibly a subset of hydrants to equip with a radio transponder only, in order to get a connected communication network, and maximize the expected benefit.

The cost of installing a *LORNO* system is composed of a fixed part for the central server, its software, and the overhead, plus a part proportional to the number of

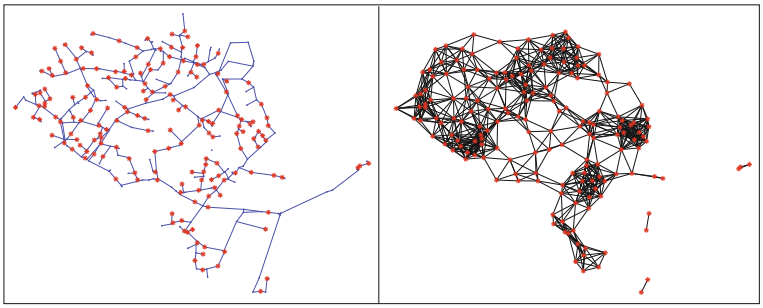

Fig. 1 Pipes and communication networks of Dailles

LORNO units that must be acquired. One could imagine that the cost of a full unit may vary according to the position of the hydrant, while we assume that the cost of a radio transponder is always the same. For our modeling purposes, we decompose the cost of a full unit into the cost of the radio transponder plus the cost of the rest of the installation.

The benefit of installing a *LORNO* unit at a particular site depends on the probability of a pipe breaking there and the potential damage caused by such an event. These quantities may vary depending on material and age of the pipes and, of course, on the surrounding environment (presence of residences, industry, hospitals, electricity, or telecom cables, and so on). In the real data we had from Lausanne, these factors had not been evaluated, and we used only the natural assumption that the benefit of one unit is proportional to the length of the pipes that is heard.

We also assume that the pipes are discretized into small enough pieces such that each is "scanned" entirely or not at all from a given hydrant. Note that such a piece may be heard by more than one hydrant, but the corresponding benefit should be counted only once. Here, it is tacitly assumed that redundant detection is not necessary to protect against equipment failure, since it tests itself continually by sending around appropriate signals.

2.2 The model

We model the optimization problem as a rooted prize-collecting Steiner arborescence problem in a directed graph $G = (V, E)$. Recall that the prize-collecting Steiner arborescence problem consists in finding a minimum weight subarborescence in a directed graph with non negative arc weights and non positive node weights. The graph G is constructed as follows. Let H be the set of hydrants and R the set of pipe pieces. For each hydrant $h_i \in H$ we introduce two nodes $\overline{h_i}$ and $\underline{h_i}$ and for each pipe piece $r_i \in R$, a node denoted also r_i. Denote these node sets by $\overline{H}$, $\underline{H}$ and R, respectively. Then, $V = \{r_0\} \cup \overline{H} \cup \underline{H} \cup R$, where r_0 is a special node, the root. It is easier to draw these nodes at different levels: the root r_0 at level 0, $\overline{H}$ at level 1, $\underline{H}$ at level 2 and R at level 3. The arcs of graph G are decomposed into four types. There is an arc from root r_0 to every node in $\overline{H}$. There is a pair of opposite arcs $\{(\overline{h_i}, \overline{h_j}), (\overline{h_j}, \overline{h_i})\}$ for each pair of hydrants which can communicate by radio station (that is, for each edge of the

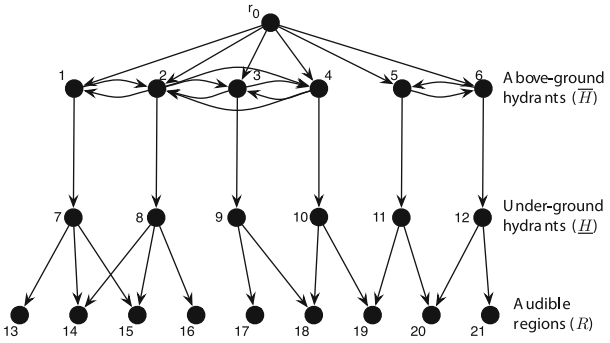

Fig. 2 A *LORNO* network

communication network), and an arc $(\overline{h_i}, \underline{h_i})$ for each hydrant. Finally, there is an arc $(\underline{h_i}, r_j)$ for each region r_j that can be heard by hydrant h_i. Figure 2 shows an example of such a network.

The elements of the network given by the graph and its weights have the following interpretation. Node r_0 represents the central station, which will actually be located in the vicinity of some hydrant. The choice of this hydrant may be free or restricted. This is modeled by an appropriate choice of arcs leaving r_0. Level 1 represents the communication network, and each node in $\overline{H}$ has the cost of a radio transponder. Level 2 represents the listening devices of the *LORNO* system, and each node in $\underline{H}$ has the cost of this installation (cost of the full *LORNO* unit minus cost of a radio transponder). Note that the only way to reach a node $\underline{h_i}$ from r_0 is by using $\overline{h_i}$. Finally, each node at level 3 has the benefit associated with this region. Note that this level may be further simplified by aggregating all nodes with the same set of predecessors in a single node with the sum of the benefits, so that the complexity does not depend on the discretization used, but only on the network topology. Now, one can see that each solution to the optimization problem corresponds to an arborescence in G, by choosing some spanning tree in the subgraph of the connection network induced by the nodes used and choosing arbitrarily which *LORNO* unit hears a region in case of multiple possibilities. Conversely, each rooted arborescence in G having at most one arc leaving r_0 defines such a solution.

Thus the problem can be formulated as that of finding an optimal r_0-rooted arborescence with the property of having exactly one arc leaving r_0—for simplicity we discard the theoretically possible solution of installing nothing—and, as an arborescence has exactly one arc entering each of its nodes, we report the cost of each node (positive or negative) on each of its entering arcs, thereby getting a standard prize-collecting Steiner arborescence problem.

3 Solving prize-collecting Steiner arborescence problems

3.1 Definitions and formulation

The prize-collecting Steiner problem has been originally defined on undirected graphs with non-negative benefits associated to the nodes and non-negative costs to the edges

as looking for an optimal connected subgraph. Under these conditions there is a tree among the optimal solutions. The rooted version ensures a given node r_0 will be part of the solution. This definition extends straightforwardly to a rooted directed graph, in which we are looking for an optimal Steiner arborescence, with the property that all costs and benefits can then be transferred with appropriate signs on the corresponding incoming arcs.

The most successful formulation for the prize-collecting Steiner arborescence (PCSA) so far is the classical cut formulation defined next, using 0/1 variables x_{ij} associated with the arcs and y_i associated with the nodes of the graph.

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{1}$$

$$\text{s.t.} \sum_{(j,i)\in E} x_{ji} = y_i, \quad \forall i \in V - \{r_0\} \tag{2}$$

$$x\left(\delta^-(S)\right) \geq y_k, \quad \forall k \in S,\ \forall S \subset V - \{r_0\} \tag{3}$$

$$\sum_{(r_0,i)\in E} x_{r_0 i} = 1 \tag{4}$$

$$x_{ij}, y_i \in \{0, 1\}, \quad \forall i \in V - \{r_0\},\ \forall (i, j) \in E \tag{5}$$

Here c_{ij} is the cost of including edge (i, j) in the solution, $\delta^-(S) = \{(i, j) \in E | i \in \overline{S}, j \in S\}$ and $x(A) = \sum\{x_e, e \in A\}$.

Constraints (2) enforce that each node i in the solution must have exactly one incoming arc, while constraint (4) implies that exactly one arc leaves r_0. The constraints (3), which we call *connectivity constraints*, ensure that, if the solution contains node k, it should also contain a path from the root r_0 to k and, thus, at least one arc in each cut induced by a node set S containing k and not r_0.

In a recent paper [9], this formulation is used as a starting point to solve PCSA problems. This paper motivated us to try to solve problems of the size we are dealing with here and we used their ideas extensively in our work. The difficulty inherent in this formulation is managing the connectivity constraints (3). There are exponentially many of these constraints, so only those truly necessary for a given instance should be used.

Relaxing all but some connectivity constraints (3), that is selecting a subset, denoted $\mathcal{L}$, of valid pairs (S, k), yields the following formulation we call current program (CP):

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{6}$$

$$\text{s.t.} \sum_{(j,i)\in E} x_{ji} = y_i, \qquad \forall i \in V - \{r_0\} \tag{7}$$

$$\sum_{(r_0,i)\in E} x_{r_0 i} = 1 \tag{8}$$

$$x\left(\delta^-(S)\right) \geq y_k, \quad \forall (S, k) \in \mathcal{L} \tag{9}$$

$$\sum_{(j,i)\in E} x_{ji} \leq \sum_{(i,j)\in E} x_{ij}, \quad \forall i \notin R \cup \{r_0\} \tag{10}$$

$$y_i \leq 1 - x_{r_0 j}, \quad \forall i < j, \{i, j\} \subset \overline{H} \tag{11}$$

$$x_{ij} + x_{ji} \leq y_i, \quad (x_{ij} \leq y_i), \quad \forall (i, j) \in E, \ i \in V - \{r_0\} \tag{12}$$

$$x_{ij}, y_i \in \{0, 1\}, \quad \forall i \in V - \{r_0\}, \ \forall (i, j) \in E \tag{13}$$

Due to the symmetric structure of the communication network and to the symmetries in the cost function, many equivalent solutions exist. To fight against this symmetry, we have added some symmetry breaking constraints (11), which force a connection between the root and the node of the communication network in the solution with smallest index.

We have also added constraints (10) and (12) for strengthening the relaxed linear programming formulation. The constraints (10) make sure that there are at least as many arcs leaving as there are arcs entering an internal node, which is valid for any arborescence. We also tried both forms of constraints (12), the stronger form, $x_{ij} + x_{ji} \leq y_i$ avoiding cycles of length 2, and the weaker form $x_{ij} \leq y_i$, forcing use of both end nodes with each choice of edge.

The connectivity constraints , i.e., the pairs (S, k) in $\mathcal{L}$, introduced at the root node are obtained as follows:

1. For each node $r \in R$, we consider the set $p(r)$ of its (direct) predecessors, (thus $p(r) \subset \underline{H}$), form the set $S_1 = \{r\} \cup p(r)$ and add the pair (S_1, r) to $\mathcal{L}$.
2. For each node $r \in R$, we consider the set $p(p(r))$ of predecessors of its predecessors, (thus $p(p(r)) \subset \overline{H}$), form the set $S_2 = \{r\} \cup p(r) \cup p(p(r))$ and add the pair (S_2, r) to $\mathcal{L}$.
3. If the connection network is not connected, then for each connected component C, we form the set $S_3 = \left\{\overline{h} | \overline{h} \in C \cap \overline{H}\right\}$ of nodes in that component and add a pair (S_3, k) to $\mathcal{L}$ for each $k \in S_3$.

For example, in figure (2), one would have for $r = 15$, $S_1 = \{15, 7, 8\}$ for type 1 and $S_2 = \{15, 7, 8, 1, 2\}$ for type 2. We would also have $S_3 = \{1, 2, 3, 4\}$ for type 3. The motivation for this choice is that these are constraints whose associated dual variables may have a positive value.

3.2 Branch-and-bound algorithm

The approach used in [9] in order to solve PCSA is to solve the linear relaxation of the current program, find violated connectivity constraints, introduce them in the current program, and iterate until all connectivity constraints are satisfied by the current solution. Finally, branch and bound is used if the optimal solution found so far is not integer. Finding the most violated connectivity constraint for a given terminal node r_i can be done efficiently by solving a max flow problem. If the maximum $r_0 - r_i$ flow value is less than y_i, the corresponding minimum cut produces such a violated constraint.

The difficulties we encountered with this approach come from two special properties of our instances. First, the number of terminal nodes is very large, about one half

of the nodes in the network. This means we find a lot of violated constraints, usually with the same amount of violation, and have no good criteria to select some among them, while introducing all of them in the current problem quickly brings the system out of memory. Secondly, the convergence of the approach is very poor due to the fact that all arcs in the communication network have the same cost.

In order to overcome these difficulties we used an approach based on finding integer solutions to the current program, using standard branch-and-bound methods. If the integer solution found is an arborescence we are done. Otherwise, the special structure of our instance allows either to find an arborescence with the same value, and thus we are done, or to find effective connectivity cuts, which we add to the current problem and iterate in the same way. Though it may seem foolish to solve a series of integer programs, this worked well for us, specially due to the following property.

Let $G_{sol} = (V_{sol}, A_{sol})$ denote the graph associated with the solution to CP, G the *LORNO* network, and $G(S)$ the subgraph of G induced by the nodes in S.

Recall the special form of the networks (see Fig. 2) we consider:

- All arcs are either directed from level i to level $i+1, i = 0, \ldots, 2$, or have both end nodes in $\overline{H}$ (at level 1), thus all circuits are entirely contained in $G(\overline{H})$.
- All arcs having both end nodes in $\overline{H}$ have the same cost.

Proposition 1 *If G_{sol} is not an arborescence but $G(V_{sol} \cap \overline{H})$ is a connected graph, then there exists an arborescence with the same value as the current solution to CP.*

Proof From (7) and (8) it follows $|V_{sol}| = \sum_{i \in V} y_i = 1 + \sum_{i \in V - r_0} y_i = 1 + \sum_{i \in V - r_0} \sum_{(j,i) \in E} x_{ji} = 1 + |A_{sol}|$. Thus, if G_{sol} is not an arborescence, then it is disconnected. From (7) and (12) we have that if the solution contains a node $i \neq r_0$, it also contains exactly one arc (j, i) and also node j. Thus it contains a path going (backward) from i either to r_0 or to a node k contained in a circuit which is, from the precedent property, entirely contained in $G(\overline{H})$. We also have $|V_{sol} \cap \overline{H}| = \sum_{i \in \overline{H}} \sum_{(j,i) \in E} x_{ji} = 1 + \sum_{i \in \overline{H}} \sum_{j \in \overline{H}} x_{ji}$, that is the solution has in $G(\overline{H})$ a number of arcs equal to its number of nodes minus one. If $G(V_{sol} \cap \overline{H})$ is connected, it contains a spanning tree which has the same number of arcs as the solution in $G(\overline{H})$. Replacing the arcs of the solution in $G(\overline{H})$ by such a spanning tree (properly oriented) gives an arborescence with the same value as G_{sol}. □

Figure 3 illustrates our algorithm. On the left, a solution of the current problem associated with the network in Fig. 2 is shown. This solution is not connected, but can be transformed into a connected solution with the same cost, as shown on the right.

This leads to the following algorithm

Step 0 Solve the current problem CP (using branch and bound)

Step 1 Find the connected components $C_1, \ldots, C_l$ of G_{sol}. If G_{sol} is connected, STOP, else go to Step 2.

Step 2 Find the connected components of $G(V_{sol} \cap \overline{H})$. If $G(V_{sol} \cap \overline{H})$ is connected, go to Step 3, else go to Step 4.

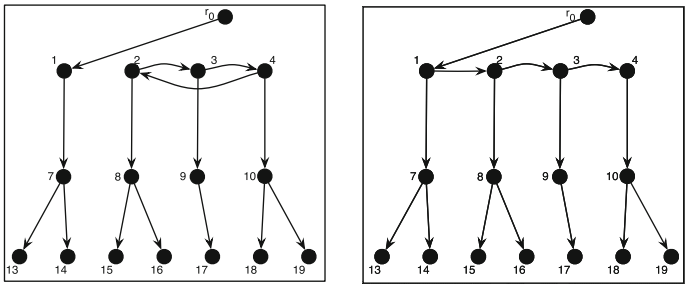

Fig. 3 Getting a connected network from a solution

Step 3 Find a spanning tree T of $G(V_{sol} \cap \overline{H})$, replace the arcs of A_{sol} contained in $G(\overline{H})$ by those of T properly oriented, if necessary prune the leaves which are not in R and terminate with this solution.

Step 4 Insert the cut

$$x(\delta^-(C_i)) \geq y_k, \quad k \in C_i$$

into CP, for all $i = 1, \ldots, l$, and return to Step 0.

It should be stressed that one of the reasons the problems at hand are hard to solve is the fact that all arcs in the communication network have the same weight. Indeed, this causes the linear programming relaxation to run astray without finding solutions that are connected graphs. Our approach aims at overcoming this difficulty by searching for trees with the same weight as the current solution. Note, furthermore, that this approach also allows us to easily find approximations, i.e., feasible solutions whose value is guaranteed to be at most α times the optimal one. Indeed it suffices to stop the branch-and-bound procedure as soon as the gap first hits α. This becomes particularly important when finding an optimal solution is no longer possible within reasonable computing time.

4 Computational experiments

With our algorithms, we were able to successfully process three real-world instances, stemming from the water distribution network of the city of Lausanne and surrounding region, the largest instance comprising 606 hydrants. This encouraged us to proceed to further testing of our approach on random instances. Below we give results from 50 such instances.

4.1 Problem generation

As described above, the PCSA instances dealt with in the present study have a special structure. We were therefore led to develop a procedure enabling us to generate random problem instances having the required characteristics. We proceed as follows. First, we generate a planar representation of a planar graph representing the pipe

network. Then, we choose a number of hydrants and their locations, and compute which hydrants if *LORNO* equipped would be able to listen to which pipe portions. Next, we generate a communication network and, finally, build the corresponding *LORNO* network (i.e., where our Steiner arborescence lives). In order to do this, we first generate n uniformly distributed points in a square of side length proportional to $\sqrt{n}$. Then, the Delaunay triangulation of this set of points (a sparse planar connected graph, containing the optimal spanning tree and easy to compute, see e.g., [1]) is computed, yielding a graph G_D on n nodes, in which a minimum Euclidean length spanning tree S_D is determined. For each edge e in $E(G_D)$, a probability p_e proportional to its length is assigned. Then, edges $e \in E(G_D) - E(S_D)$ are randomly eliminated with probability p_e, until a desired average node degree d_a is achieved. This process yields a connected planar graph G'_D with n nodes and average degree d_a. We then add $|H|$ hydrants to randomly chosen edges of this graph. The position of each hydrant on the edge is also determined randomly, at a location close to the center of the edge. For each hydrant, a node is added to the graph and the associated edge is split, creating two new edges and one new node. Then, the communication network is generated by adding an edge between hydrants $h_1, h_2 \in H$ with probability

$$p_{h_1,h_2} = \begin{cases} 0 & d(h_1, h_2) > r_2 \\ p_2 & r_1 \leq d(h_1, h_2) \leq r_2 \\ p_1 & r_1 > d(h_1, h_2). \end{cases}$$

Finally, we determine for each edge, the set of hydrants from which it can be heard. An edge $e = (e_1, e_2)$ is audible by hydrant h if $d(h, e_1) < r_L$ and $d(h, e_2) < r_L$. Results are reported below for instances generated with $|H| = n/2$, and

$$\begin{aligned} d_a &= 2.3 \\ r_1 &= 250 \\ r_2 &= 400 \\ p_1 &= .8 \\ p_2 &= .5 \end{aligned}$$

The name of each instance in the following tables resumes the main data: $r1 - 10050$ means that it is a random instance, the seed of the random generator was 1,100 points have been generated in the plane and 50 hydrants have been placed in the resulting graph.

4.2 Algorithms

The branch-and-cut algorithm, denoted CONN, was implemented in C/C++, using the libraries available from the Computational Infrastructure for Operations Research (COIN-OR) repository [10]. The Open Solver Interface (OSI) was used to interface with the integer and linear programming solvers. All results reported here reflect the use of OSI CPLEX interface, where CPLEX 9.1 was used to solve the

integer and linear programming instances generated throughout the course of the algorithm. The algorithm was tested on an Intel Xeon 2.4 GHz processor with 4 GB of memory.

After carefully examining our preliminary results, we realized that a significant amount of running time was being spent on proving the optimality of solutions to CP that would eventually be discarded because of constraint violation. As mentioned previously, the design of our algorithm allows to change the optimality requirements of CP solutions without altering the overall structure of the algorithm. Aside from the ability to get good solutions quickly, this also allows us to change our optimality gap dynamically within the algorithm. The cut generation routine requires only a feasible solution to produce valid cuts. Thus, it is possible to widen the optimality gap in the early iterations of the algorithm and generate several rounds of cuts in a much smaller amount of CPU time. In order to test the benefit of this procedure, we experimented with a slight variant of the branch-and-cut algorithm denoted GAPCONN, where we systematically modify the optimality requirement during the course of the algorithm, i.e., **Step 0** of the algorithm becomes "Solve the current problem CP with an optimality gap set to δ" and the parameter δ is decreased during the course of the algorithm. We refer to algorithms of this type as *dynamic*, while the standard algorithms can be described as *static*. As already described, the objective function comprises an easily quantifiable component, namely the investment costs, and one which is less so corresponding to the drawn benefits. The former are quantized by the cost of a full *LORNO* unit, resp. that of a transponder unit. Rather than using a relative gap measure for stopping criteria it seems more sensible to use an absolute gap. We chose to bound the gap successively by the cost of a full *LORNO* unit, that of a transponder unit and by ε, a sufficiently small parameter to prove optimality. Note that for our numerical examples these values make good sense, indeed they add up to at most a fraction of a percent of the objective function value.

To evaluate the effectiveness of our separation routine, we implemented an alternate algorithm similar to that described in [9]. In this algorithm, denoted FLOW, a modification of Goldberg's maximum flow algorithm [3] is used to find violated constraints with respect to solutions of the LP-relaxation of CP. In the case that such a solution does not violate any constraints, an integer programming solver is called to find an integer solution to CP. If the resulting integer solution is also feasible for the original problem, it must be optimal. Else, new violated inequalities are added to CP, and the algorithm continues. This algorithm was also implemented in C/C++, using COIN-OR's libraries to interface with CPLEX 9.1.

As mentioned previously, in addition to the comparison of separation routines, we also experimented with the form of the constraints (12). The algorithms included in our experiments are summarized in Table 1.

4.3 Results

We present here results from two classes of experiments. The first consists of a series of tests where we compare the algorithmic variants on data instances of increasing size. In each table, the set of columns labeled *Iterations* gives the number of algorithmic

Table 1 The algorithm variants used in the computational study

	CONN	FLOW	GAPCONN
$x_{ij} + x_{ji} \leq y_i$	CONN2	FLOW2	GAPCONN2
$x_{ij} \leq y_i$	CONN4	–	GAPCONN4

Table 2 Summary results for all algorithms

	Success ratio	Avg no. Iter.	Avg CPU sec
CONN2	0.78	7.85	249.15
CONN4	0.80	7.38	118.49
FLOW2	0.54	66.30	154.52
GAPCONN2	0.82	15.54	331.41
GAPCONN4	0.88	20.43	318.15

loops necessary to find the optimal solution. The group of columns labeled *CPU sec* gives the running time of the algorithms on the platform described above. Unless otherwise noted, all results presented represent a CPLEX optimality gap of 1×10^{-4}. For the algorithms in which the gap is changed dynamically, this corresponds to a choice of $\varepsilon = 1 \times 10^{-4}$. For this study, a maximum running time of 5000 CPU seconds was allotted. In the tables, instances that were not solved within this time limit are indicated with a dash in the corresponding row of the table. Note that the set of unsolved instances includes both those instances that exceeded the time limit, as well as those whose memory requirements were too large for the resources available. We do not differentiate between these two types of unsolved instances in our presentation of results. However, we do note that the FLOW algorithm frequently fails due to memory requirements. This suggests that if this algorithm is used, unnecessary cuts should be removed dynamically throughout the course of the algorithm.

The complete output for the experimental study is shown in Table 3. From the table, we can see that the dynamic variants of the branch-and-cut algorithm (GAPCONN2 and GAPCONN4) clearly dominate their static counterparts. Further, there is no problem that GAPCONN2 is able to solve that GAPCONN4 cannot. Thus, we can say that GAPCONN4 is the most robust, with respect to number of problems solved, of all the branch-and-cut variants. Additionally, in Sect. 4.4, we compare the performance of GAPCONN2 and GAPCONN4 across different optimality requirements, and see that when the optimality gap is equal to the cost of a full *LORNO* installation, GAPCONN4 is able to solve all but two problems in our test set.

It is not immediately obvious, however, how the branch-and-cut variants compare to FLOW2, since there are problems that FLOW2 is able to solve where all branch-and-cut algorithms fail. However, a comparison between FLOW2 and GAPCONN4 shows that this occurs only twice in the entire test set. Additionally, the total number of problems solved by each of the branch-and-cut algorithms is significantly higher than that by FLOW2.

The results are summarized in Table 2.

In this table, we report the average number of instances solved, the average required iterations and average CPU time required for each algorithm. Table 2 gives further

Table 3 Results from all variants on the full test set

Instance	Iterations					CPU sec				
	CONN2	CONN4	FLOW2	GCONN2	GCONN4	CONN2	CONN4	FLOW2	GCONN2	GCONN4
r1-10050	1	3	1	2	3	0.780	0.432	0.540	1.320	0.492
r2-10050	1	2	4	1	2	0.428	1.044	0.712	0.468	1.352
r3-10050	6	8	47	6	7	8.005	16.313	3.804	8.165	16.585
r4-10050	4	5	–	41	6	2.356	2.192	–	31.550	3.216
r5-10050	2	1	28	2	1	1.160	0.720	7.708	1.360	0.988
r6-10050	1	1	19	1	1	0.340	0.324	1.832	0.428	0.408
r7-10050	20	13	168	11	15	38.994	15.877	19.881	10.553	16.281
r8-10050	2	1	3	2	1	1.000	0.300	0.524	1.080	0.360
r9-10050	–	–	146	–	–	–	–	82.513	–	–
r10-10050	4	4	4	5	3	3.748	2.692	0.620	5.568	3.560
r1-200100	–	–	–	51	22	–	–	–	991.814	110.323
r2-200100	1	5	53	1	5	8.028	10.229	66.240	11.601	9.845
r3-200100	–	–	–	–	–	–	–	–	–	–
r4-200100	–	–	–	–	110	–	–	–	–	1128.250
r5-200100	1	1	29	1	1	3.852	4.300	14.301	4.748	4.752
r6-200100	2	3	7	2	3	5.616	10.097	4.412	6.712	11.145
r7-200100	3	1	73	3	1	9.253	5.228	75.293	11.697	5.728
r8-200100	3	2	103	3	2	6.756	5.368	129.632	8.129	7.176
r9-200100	3	3	9	4	4	5.400	11.109	4.240	7.828	16.285
r10-200100	1	4	–	1	4	5.784	19.433	–	8.065	19.973
r1-300150	3	9	–	3	9	13.197	52.631	–	19.201	61.420

Table 3 continued

Instance	Iterations					CPU sec				
	CONN2	CONN4	FLOW2	GCONN2	GCONN4	CONN2	CONN4	FLOW2	GCONN2	GCONN4
r2-300150	1	6	48	1	6	17.081	51.199	54.579	27.918	57.000
r3-300150	7	7	33	18	31	43.075	40.911	26.370	106.323	152.254
r4-300150	2	4	–	2	6	38.286	106.547	–	47.807	107.479
r5-300150	54	81	–	76	362	381.604	903.728	–	560.043	3850.540
r6-300150	2	5	73	4	7	25.490	44.879	128.984	41.183	55.572
r7-300150	–	–	–	21	23	–	–	–	565.779	846.493
r8-300150	3	10	–	3	7	25.774	162.306	–	34.490	76.277
r9-300150	–	–	70	–	–	–	–	56.452	–	–
r10-300150	2	9	–	5	8	22.445	100.690	–	45.923	73.241
r1-400200	3	1	31	6	1	47.439	27.070	58.040	67.664	33.910
r2-400200	1	6	–	2	6	46.519	59.192	–	51.271	87.137
r3-400200	1	2	35	1	4	32.934	63.120	80.733	32.546	91.018
r4-400200	–	–	–	–	–	–	–	–	–	–
r5-400200	79	2	–	–	2	2586.150	29.058	–	–	45.847
r6-400200	8	22	186	20	37	106.175	229.622	535.125	238.467	516.076
r7-400200	5	7	–	6	9	79.577	215.217	–	99.922	265.701
r8-400200	9	6	89	11	15	98.942	82.037	238.915	66.496	263.248
r9-400200	–	10	–	116	9	–	94.446	–	2284.760	134.308
r10-400200	–	–	–	–	–	–	–	–	–	–
r1-500250	25	12	–	29	17	2727.330	605.638	–	2084.730	610.334
r2-500250	2	6	–	3	10	97.542	251.568	–	139.333	310.511
r3-500250	15	10	186	44	23	2181.300	772.440	1099.330	1304.160	692.931

Table 3 continued

Instance	Iterations					CPU sec				
	CONN2	CONN4	FLOW2	GCONN2	GCONN4	CONN2	CONN4	FLOW2	GCONN2	GCONN4
r4-500250	15	5	64	113	5	605.138	133.924	202.733	4192.560	222.622
r5-500250	3	2	128	3	9	73.285	80.161	474.438	90.942	160.966
r6-500250	–	–	–	–	–	–	–	–	–	–
r7-500250	1	2	–	2	3	69.864	77.853	–	75.961	86.357
r8-500250	–	–	–	–	70	–	–	–	–	2909.050
r9-500250	6	5	–	7	7	190.108	193.848	–	186.352	293.374
r10-500250	4	9	153	4	22	106.075	255.872	803.998	113.095	638.072

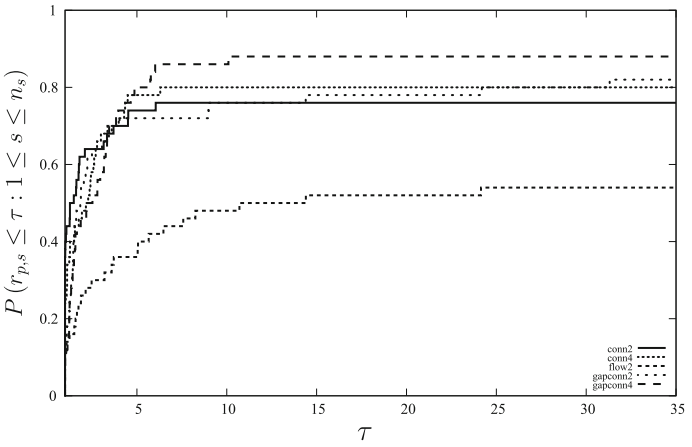

Fig. 4 Performance profiles of each of the solvers described

evidence that GAPCONN4 is the most robust of all algorithm variants, solving almost 90% of the problems in the test set, but also shows that it requires the second highest average CPU time. FLOW2 is the fastest algorithm, on average, but solves only 54% of the problems. CONN4 seems to yield the best balance between speed and robustness, solving 80% of the test problems, with the second lowest average CPU time. This table also suggests that, for the problems we study here, the weaker form of constraint (12) is preferable to the stronger form since, for both the static and dynamic variants, using this form yielded a higher success rate and a lower average CPU time. It should be noted, however, that the results in Table 2 may be somewhat misleading, since the averages do not account for those instances that remain unsolved by each algorithm. As noted in [4], using the averages tends to punish the more robust solvers and reward those that solve a few instances quickly but fail to solve the more difficult ones in the test set.

In an attempt to overcome this shortcoming, we use the presentation described by [4], where we measure performance of an algorithm relative to performance of other algorithms on the same instance. Formally, for each algorithm $s \in \mathcal{S}$, we consider

$$\rho_s(\tau) = \frac{1}{n_p} \text{size}\left\{p \in \mathcal{P} : r_{p,s} \leq \tau\right\}, \tag{14}$$

where n_p is the size of the test set $\mathcal{P}$ and $r_{p,s}$ is the ratio of the running time of s on p to the minimum running time for p over $s \in \mathcal{S}$. Note that (14) is the cumulative distribution function for the performance ratio $r_{p,s}$ and yields the probability that $r_{p,s}$ is within a factor of τ from the best ratio. Thus, $\rho_s(1)$ gives the probability that solver s beats the other solvers in $\mathcal{S}$. We refer to the function ρ_s as the *performance profile*. Note that there is slight abuse in terminology here. All definitions from [4] are given in terms of solvers, rather than algorithms. Of course, in order to compare algorithms, we are truly comparing their implementations by a given solver. As the solvers we

Table 4 Results from budget study

dail							r1-300150						
GAPCONN2				GAPCONN4			GAPCONN2				GAPCONN4		
B	Iter	CPU sec	Obj	Iter	CPU sec	Obj	*B*	Iter	CPU sec	Obj	Iter	CPU sec	Obj
45	–	–	–	–	–	–	55	–	–	–	–	–	–
47	15	5166.830	−2083350	23	2862.860	−2083350	57	47	999.510	−3064700	64	3058.000	−3064700
49	12	628.147	−2117540	8	213.957	−2117540	59	18	342.893	−3099980	9	383.148	−3100240
51	4	174.463	−2149400	2	81.645	−2150150	61	13	176.339	−3131050	21	978.061	−3131320
53	1	65.984	−2180740	1	66.388	−2180740	63	19	166.990	−3157350	34	715.033	−3157610
55	1	46.515	−2206060	1	70.976	−2206060	65	24	202.717	−3181040	7	117.647	−3181040
57	1	47.531	−2224240	1	73.185	−2225680	67	8	33.002	−3199500	7	108.987	−3200220
59	1	42.639	−2242660	1	63.808	−2243540	69	7	24.373	−3216620	5	62.208	−3216620
61	1	42.079	−2258840	1	69.864	−2258840	71	3	20.285	−3228980	6	69.136	−3228980
63	1	57.576	−2271700	1	67.428	−2271700	73	3	9.397	−3240300	3	39.331	−3240300
65	1	51.567	−2281670	1	68.800	−2281670	75	3	12.941	−3249670	2	35.974	−3249670
67	1	59.312	−2286980	1	60.852	−2286980	77	5	17.605	−3256390	2	44.183	−3256390
69	1	52.735	−2287640	1	55.947	−2287640	79	5	15.209	−3256390	3	52.935	−3256390

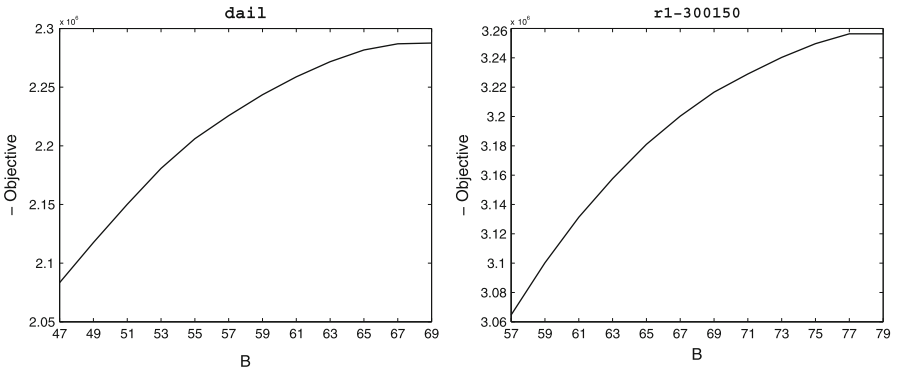

Fig. 5 Tradeoff between the installation budget and the objective value

wish to compare are identical except for the difference explicit in the algorithms, we use these terms interchangeably. Figure 4 shows the performance profiles of the solvers on the full test set (see also Table 3), where n_s is the number of solvers, whereas n_p is the number of problems being tested, each solver is given a number between 1 and n_s, i.e., $\mathcal{S} = 1, 2, \ldots, n_s$. From the figure, we can see that CONN2 achieved the minimum solution time on the largest number of problems (roughly 40%). Thus, in a loose sense, we can say this algorithm is the fastest. Finding points of intersection within the plot allows us to determine those values of τ for which a subset of the algorithms is equivalent with respect to running time. From the plot, we can see that CONN2, CONN4, GAPCONN2, GAPCONN4 will all solve a given problem within a factor of 4 of the fastest algorithm roughly 70% of the time.

In fact, for a range of values of τ between 3.5 and 4, the algorithms CONN2 and CONN4 are almost undistinguishable with respect to running time. This is not so surprising, since these two solvers differ only in the form of the constraints (12). From the larger values of τ, we can see the probability that the solvers solve a problem within our test set. GAPCONN4 is the most likely to solve a random instance, solving approximately 90% of the problems tested. GAPCONN2 and CONN4 both solve roughly 80% of the instances, and CONN2 is successful 75% of the time. From the plots, we can also see that the probability of success does not significantly increase for $\tau > 5$ for any of our solvers, except GAPCONN4. Thus, if GAPCONN2, CONN2 or CONN4 is able to solve an instance, it is likely that it will solve the instance within five times the speed of the fastest solver. Figure 4 confirms our earlier assertion that the flow algorithm is dominated by all variants of the branch-and-cut algorithm. FLOW2 is the fastest solver only 15% of the time and solves only half of the instances in the test set. Conversely, for each solver, we can consider the fraction of problems that the solver fails to solve within a factor of τ of the best solver. This value is given by the metric $1 - \rho_s(\tau)$, and includes those problems for which the solver fails to solve altogether. Using this metric, we can analyze the performance of the solvers, neither giving undue credit to solvers that lack robustness nor allowing difficult instances to dictate our results.

Table 5 Comparing the cost of optimality for GAPCONN2

Instance	LORNO			TRANS			EPS		
	Iterations	CPU sec	Objective	Iterations	CPU sec	Objective	Iterations	CPU sec	Objective
r1-10050	1	0.848	−960341	1	0.876	−960341	2	1.320	−961061
r2-10050	1	0.464	−890994	1	0.468	−890994	1	0.468	−890994
r3-10050	3	0.764	−1062120	3	0.792	−1062120	6	8.165	−1062120
r4-10050	37	30.074	−1139590	40	31.478	−1140310	41	31.550	−1141030
r5-10050	2	1.344	−957468	2	1.356	−957468	2	1.360	−957468
r6-10050	1	0.420	−1113020	1	0.428	−1113020	1	0.428	−1113020
r7-10050	10	7.152	−813227	10	7.176	−813227	11	10.553	−813227
r8-10050	2	1.072	−1023130	2	1.080	−1023130	2	1.080	−1023130
r9-10050	8	6.296	−969108	8	6.316	−969108	–	–	–
r10-10050	5	4.772	−987484	5	4.820	−987484	5	5.568	−987484
r1-200100	7	27.214	−2209600	51	967.125	−2210320	51	991.814	−2210320
r2-200100	1	9.421	−2080110	1	9.505	−2080110	1	11.601	−2080110
r3-200100	14	78.121	−1910670	–	–	–	–	–	–
r4-200100	57	274.581	−2027370	–	–	–	–	–	–
r5-200100	1	4.708	−2166220	1	4.732	−2166220	1	4.748	−2166220
r6-200100	2	6.676	−2017630	2	6.700	−2017630	2	6.712	−2017630
r7-200100	3	11.665	−2008300	3	11.689	−2008300	3	11.697	−2008300
r8-200100	3	5.836	−2010260	3	6.416	−2010260	3	8.129	−2010260
r9-200100	3	3.548	−2176660	4	3.892	−2177380	4	7.828	−2177380
r10-200100	1	6.552	−2086780	1	8.053	−2086780	1	8.065	−2086780
r1-300150	3	13.077	−3256390	3	13.233	−3256390	3	19.201	−3256390
r2-300150	1	19.161	−3131920	1	27.606	−3131920	1	27.918	−3131920
r3-300150	17	105.683	−3133680	17	105.807	−3133680	18	106.323	−3134400
r4-300150	2	47.331	−2725630	2	47.487	−2725630	2	47.807	−2725630
r5-300150	76	551.646	−3048320	76	551.802	−3048320	76	560.043	−3048320
r6-300150	3	36.258	−3098540	4	37.302	−3099260	4	41.183	−3099260
r7-300150	21	458.725	−3134060	21	499.796	−3134060	21	565.780	−3134060
r8-300150	3	16.913	−3114700	3	17.121	−3114700	3	34.490	−3114700
r9-300150	12	97.430	−3029640	12	97.586	−3029640	–	–	–
r10-300150	4	33.390	−3161470	5	40.239	−3162190	5	45.923	−3162190
r1-400200	5	66.032	−4290830	5	66.236	−4290830	6	67.664	−4291550
r2-400200	1	45.059	−4273880	2	51.003	−4274600	2	51.271	−4274600
r3-400200	1	32.438	−4105300	1	32.514	−4105300	1	32.546	−4105300
r4-400200	–	–	–	–	–	–	–	–	–
r5-400200	68	658.465	−4178710	187	2132.825	−4179430	–	–	–
r6-400200	15	184.216	−3880780	15	184.440	−3880780	20	238.467	−3881500
r7-400200	5	79.665	−4269060	6	89.642	−4269780	6	99.922	−4269780
r8-400200	11	66.416	−4324840	11	66.468	−4324840	11	66.496	−4324840
r9-400200	29	303.971	−4091240	116	2278.291	−4091960	116	2284.763	−4091960
r10-400200	5	70.220	−3987600	7	108.671	−3987600	–	–	–

Table 5 continued

Instance	LORNO			TRANS			EPS		
	Iterations	CPU sec	Objective	Iterations	CPU sec	Objective	Iterations	CPU sec	Objective
r1-500250	15	350.346	−5196930	16	468.133	−5196930	29	2084.723	−5197650
r2-500250	2	90.938	−5257910	2	91.294	−5257910	3	139.333	−5257910
r3-500250	31	685.811	−5040440	44	1278.000	−5041160	44	1304.158	−5041160
r4-500250	100	3346.910	−5420330	113	4192.203	−5421050	113	4192.555	−5421050
r5-500250	3	66.344	−5236380	3	72.665	−5236380	3	90.942	−5236380
r6-500250	–	–	–	–	–	–	–	–	–
r7-500250	1	64.088	−5088330	2	75.521	−5089050	2	75.961	−5089050
r8-500250	18	399.773	−5030270	–	–	–	–	–	–
r9-500250	3	101.246	−5154660	7	185.963	−5155070	7	186.351	−5155070
r10-500250	4	110.307	−5401620	4	112.843	−5401620	4	113.095	−5401620

In the formulation described in this paper, we have assumed that we are free to install as many full *LORNO* installations as desired. However, this may not be a realistic assumption, since this number may be limited by physical or financial constraints. In order to test the sensitivity of our algorithm to this assumption, a second computational test was performed. We add the constraint

$$\sum_{i \in \underline{H}} y_i \leq B \tag{15}$$

to the formulation and apply the solution algorithms for varying bounds B on the number of installed auditory components. Due to the increased difficulty of these restricted problems, we chose to relax our optimality requirements. As mentioned previously, the design of our algorithm allows the user to change the desired optimality gap without alteration of the algorithm. For this experiment, we used an optimality gap equal to the cost of one *LORNO* installation. These results are also presented in tabular format, as before. Here, we have two additional columns, labeled B and *Obj*, which indicate the limit placed on the number of full *LORNO* installations and the resulting optimal objective value, respectively. The data used for the experiment consisted of both a real water network from Lausanne, as well as a randomly generated instance with similar characteristics. The full results are shown in Table 4.

Aside from testing the sensitivity of our algorithm, this study allows us to examine the inherent tradeoff that exists between the installation limit and the resulting benefit. Figure 5 illustrates this relationship for both data instances. The portions of the tradeoff curves in Fig. 5 we are most interested in are those with a steep slope. These areas represent critical points, where small increases in B yield substantial increases in the optimal benefit. Assuming the budget constraint (15) is somewhat flexible, these critical points indicate where it is worthwhile to increase the installation limit.

Table 6 Comparing the cost of optimality for GAPCONN4

Instance	LORNO			TRANS			EPS		
	Iterations	CPU sec	Objective	Iterations	CPU sec	Objective	Iterations	CPU sec	Objective
r1-10050	3	0.488	−961061	3	0.492	−961061	3	0.492	−961061
r2-10050	2	1.348	−890994	2	1.352	−890994	2	1.352	−890994
r3-10050	4	2.056	−1062120	4	2.080	−1062120	7	16.585	−1062120
r4-10050	2	0.668	−1140310	2	0.696	−1140310	6	3.216	−1141030
r5-10050	1	0.972	−957468	1	0.980	−957468	1	0.988	−957468
r6-10050	1	0.400	−1113020	1	0.404	−1113020	1	0.408	−1113020
r7-10050	14	11.037	−813227	14	11.061	−813227	15	16.281	−813227
r8-10050	1	0.356	−1023130	1	0.356	−1023130	1	0.360	−1023130
r9-10050	15	10.957	−968388	25	16.057	−969108	–	–	–
r10-10050	3	2.348	−987484	3	2.372	−987484	3	3.560	−987484
r1-200100	22	86.185	−2210320	22	89.534	−2210320	22	110.323	−2210320
r2-200100	5	7.020	−2080110	5	7.140	−2080110	5	9.845	−2080110
r3-200100	34	253.704	−1910670	–	–	–	–	–	–
r4-200100	110	1121.570	−2028090	110	1121.666	−2028090	110	1128.246	−2028090
r5-200100	1	4.732	−2166220	1	4.744	−2166220	1	4.752	−2166220
r6-200100	3	11.125	−2017630	3	11.137	−2017630	3	11.145	−2017630
r7-200100	1	5.700	−2008300	1	5.720	−2008300	1	5.728	−2008300
r8-200100	2	4.764	−2010260	2	5.216	−2010260	2	7.176	−2010260
r9-200100	4	12.429	−2177380	4	12.541	−2177380	4	16.285	−2177380
r10-200100	4	18.497	−2086790	4	19.965	−2086790	4	19.973	−2086790
r1-300150	9	52.667	−3256390	9	52.867	−3256390	9	61.420	−3256390
r2-300150	6	50.579	−3131920	6	56.616	−3131920	6	57.000	−3131920
r3-300150	19	83.625	−3133680	19	83.833	−3133680	31	152.254	−3134400
r4-300150	3	44.267	−2724910	6	107.263	−2725630	6	107.479	−2725630
r5-300150	357	3757.180	−3047600	361	3839.161	−3047600	362	3850.542	−3048320
r6-300150	5	43.099	−3098540	7	51.791	−3099260	7	55.571	−3099260
r7-300150	23	357.806	−3134060	23	667.617	−3134060	23	846.492	−3134060
r8-300150	7	55.731	−3114700	7	55.944	−3114700	7	76.277	−3114700
r9-300150	6	35.250	−3028920	40	347.438	−3029640	–	–	–
r10-300150	8	64.168	−3162190	8	64.380	−3162190	8	73.241	−3162190
r1-400200	1	33.786	−4291550	1	33.870	−4291550	1	33.910	−4291550
r2-400200	6	86.541	−4274600	6	86.857	−4274600	6	87.137	−4274600
r3-400200	2	67.876	−4104580	3	90.242	−4104580	4	91.018	−4105300
r4-400200	–	–	–	–	–	–	–	–	–
r5-400200	2	34.474	−4180150	2	34.730	−4180150	2	45.847	−4180150
r6-400200	37	515.456	−3881500	37	515.764	−3881500	37	516.076	−3881500
r7-400200	7	212.777	−4269060	9	244.911	−4269780	9	265.700	−4269780
r8-400200	4	84.905	−4323400	11	153.734	−4324120	15	263.249	−4324840
r9-400200	9	121.748	−4091960	9	122.028	−4091960	9	134.309	−4091960
r10-400200	3	56.243	−3987600	5	101.106	−3987600	–	–	–

Table 6 continued

Instance	LORNO			TRANS			EPS		
	Iterations	CPU sec	Objective	Iterations	CPU sec	Objective	Iterations	CPU sec	Objective
r1-500250	17	580.528	−5197650	17	609.634	−5197650	17	610.334	−5197650
r2-500250	5	173.587	−5257190	10	279.890	−5257910	10	310.512	−5257910
r3-500250	22	482.546	−5041160	22	482.962	−5041160	23	692.931	−5041160
r4-500250	5	197.368	−5421050	5	222.194	−5421050	5	222.622	−5421050
r5-500250	9	138.053	−5236380	9	138.509	−5236380	9	160.966	−5236380
r6-500250	–	–	–	–	–	–	–	–	–
r7-500250	2	63.268	−5088330	3	83.061	−5089050	3	86.357	−5089050
r8-500250	68	2442.070	−5030270	70	2505.382	−5030990	70	2909.051	−5030990
r9-500250	2	146.837	−5154660	7	277.413	−5155070	7	293.374	−5155070
r10-500250	19	527.873	−5400900	20	573.680	−5400900	22	638.072	−5401620

4.4 The cost of optimality

In this section, we compare the cost of proving optimality for GAPCONN2 and GAPCONN4. Tables 5 and 6 show the full sets of results for the two algorithms. In each table, we see the required iterations and CPU time, as well as the objective value for each value of optimality gap discussed in the previous section. We denote these values by *LORNO*, the cost of a full *LORNO* installation, *TRANS*, the cost of installing a radio transponder only, and *EPS*, a sufficiently small parameter that yields "true" optimality. For GAPCONN2, the results indicate that, on average, moving from *LORNO*-optimality to TRANS-optimality requires approximately 107% more CPU time, while improving the objective by only 0.007%. Further, to achieve EPS-optimality, GAPCONN2 requires approximately 162% more CPU time than for *LORNO*-optimality, and yields only a 0.012% objective improvement. The results for GAPCONN4 are less dramatic, but show a similar tendency. For GAPCONN4, the average difference in runtime between *LORNO*- and TRANS-optimality is approximately 37% with a 0.005% objective improvement, and obtaining EPS-optimality requires a 59% increase in CPU time and yields a 0.007% objective improvement.

5 Conclusion

In this paper we presented a decision aid methodology for the evaluation of *LORNO*, a new acoustic water leakage detection system in urban water distribution networks. The following aspects were stressed:

- modeling the operation of *LORNO* in a way capturing its main features;
- describing the placement problem as a well-defined optimization problem, which turned out to be a particular type of Prize-Collecting Steiner Arborescence Problem;

– developing suitable methods for finding both exact and approximate solutions with a given performance guarantee;
– collecting real data from a middle-sized city and creating a test-bed with realistic looking randomly generated instances.

The tool has been tested on real Lausanne data and has produced convincing results, furthermore it allows to perform various types of sensitivity analyses. As a bottom line it can be said that the present approach is not only well suited to solve the practical problems at hand, but also yields efficient procedures to solve difficult instances of the prize-collecting Steiner arborescence problem.

Acknowledgments We are grateful to Sébastien Apothéloz, Head of the Water Supply Services of the City of Lausanne, for having proposed the problem discussed in this paper and for having provided the data used. We also gratefully acknowledge the assistance of Dr. Gautier Stauffer, and of former students Sarah Betschart, Julia Dias and Louis Pellaux, who were active at various phases of this project.

References

1. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry Algorithms and Applications. Springer, Berlin (2000)
2. Carr, R.D., Greenberg, H.J., Hart, W.E., Konjevod, G., Lauer, E., Lin, H., Morrison, T., Phillips, C.A.: Robust optimization of contaminant sensor placement for community water systems. Math. Program. **107**(1–2), 337–356 (2006)
3. Cherkassky, B.V.: On implementing the push-relabel method for the maximum flow problem. Algorithmica **19**(4), 390–410 (1997)
4. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2), 201–213 (2002)
5. Fischetti, M.: Facets of two steiner arborescence polyhedra. Math. Program. **51**(3), 401–419 (1991)
6. Goemans, M., Williamson, D.: A general approximation technique for constrained forest problems. SIAM J. Comput. **24**(2), 296–317 (1995)
7. Goemans, M., Williamson, D.: The primal-dual method for approximation algorithms and its application to network design problems. Hochbaum D. (ed.) Approximation Algorithms for NP-hard Problems, pp. 144–191, PWS Publishing Company, Boston (1997)
8. Johnson, D.S., Minkoff, M., Phillips, S.: The prize collecting steiner tree problem: theory and practice. In: Proceedings of the 11th symposium on discrete algorithms, pp. 760–769 (2000)
9. Ljubic, I., Weiskircher, R., Pferschy, U., Klau, G., Mutzel, P., Fischetti, M.: An algorithmic framework for the exact solution of the prize-collecting steiner tree problem. Math. Program. Ser. B **105**(2–3), 427–449 (2006)
10. Lougee-Heimer, R.: The common optimization interface for operations research. IBM J. Res. Dev. **47**(1), 57–66 (2003)
11. Margot, F., Prodon, A., Liebling, T.M.: Tree polytope on 2-trees. Math. Program. **63**(2), 183–191 (1994)