Accepted to *INFORMS Journal on Computing*
manuscript (Please, provide the mansucript number!)

Mathematical Programming Algorithms for Spatial Cloaking

Alberto Ceselli, Maria Luisa Damiani, Giovanni Righini, and Diego Valorsi
Università degli Studi di Milano, Dipartimento di Informatica

We consider a combinatorial optimization problem for spatial information cloaking. The problem requires to compute one or several disjoint arborescences on a graph from a predetermined root or subset of candidate roots, so that the number of vertices in the arborescences is minimized but a given threshold on the overall weight associated with the vertices in each arborescence is reached. For a single arborescence case, we solve the problem to optimality by designing a branch-and-cut exact algorithm. Then we adapt this algorithm for the purpose of pricing out columns in an exact branch-and-price algorithm for the multi-arborescence version. We also propose a branch-and-price-based heuristic algorithm, where branching and pricing respectively act as diversification and intensification mechanisms. The heuristic consistently finds optimal or near optimal solutions within a computing time which can be three to four orders of magnitude smaller than that required for exact optimization. From an application point of view, our computational results are useful to calibrate the values of relevant parameters, determining the obfuscation level that is achieved.

Key words: Steiner trees, branch-and-price, branch-and-cut
History:

1. Introduction

The widespread use of location-aware mobile applications raises important challenges for the protection of location privacy [15]. Popular applications include location-based services, offering information services in exchange of personal location [9]; geo-social networks, enabling the sharing of personal location data within a community [16]; participatory geospatial data collection, with individuals acting as mobile sensors gathering georeferenced data [17]. In all such applications the location is communicated to a third party. Whenever such a third party is not fully trusted, as for example when the service provider is *honest-but-curious*, the uncontrolled sharing of location information inevitably exposes individuals to possible abuses [10]. A main stream of research on location privacy concerns the development of techniques enabling the protection of true locations from untrusted parties. For example, *location cloaking* is a popular technique that replaces true locations with uncertain locations [14]. The gain in location privacy, however, is normally paid in terms

of location quality (e.g. location-based services are less accurate), therefore the challenge is to minimize the utility loss while providing strong privacy guarantees against possible threats. Location privacy preserving techniques address two major privacy goals, namely the protection of user anonymity from location-driven inferences and the protection of personal location in non anonymous applications, respectively [9, 13]. Especially the latter research stream has attracted great interest, following the discovery of increasingly complex privacy threats, such as those enabled by the availability of *background knowledge*, consisting of publicly available data that, combined with location information, can lead to the disclosure of sensitive information. For example, cloaked locations fully contained into point of interest generally considered sensitive such as hospitals and religious buildings put users' privacy at risk. This problem has been first addressed by the PROBE framework [2, 3] while a number of related threats and approaches can be found in literature [11, 12, 18]. The privacy issue can be more precisely formulated as follows. Given a map reporting the location and extent of both sensitive and non sensitive places, and given people distribution over the region, determine the cloaked regions of minimal size such that the probability of finding the users in a sensitive place, inside a cloaked region, is limited (i.e. upper bounded).

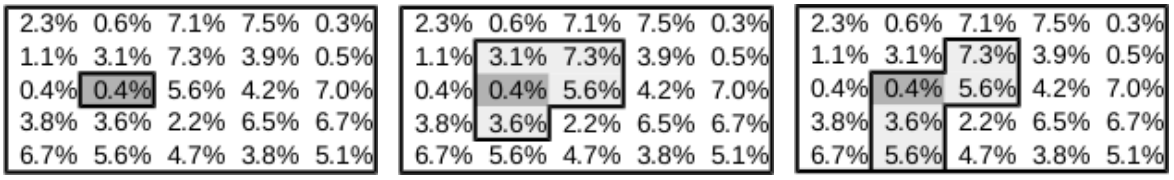
In this paper we provide the following major contributions. First, we propose a combinatorial optimization model faithfully representing the location cloaking problem (Section 2). Unfortunately, such a combinatorial optimization counterpart turns out to be hard from both a theoretical and a computational point of view. Therefore, we design ad hoc exact algorithms exploiting mathematical programming and graph optimization. In particular, we first consider the special case of a single sensitive location, we introduce a compact formulation, and we combine reduction procedures, valid cuts, heuristics and dedicated search strategies in a branch-and-cut framework (Section 3). Then, we tackle the general case with a decomposition approach: we identify some features on the structure of optimal solutions that help to manage a nonlinear objective function, and to reduce symmetries; then we provide an extended formulation that is solved by means of branch-and-price. Pricing is performed with both dedicated heuristics and a branch-and-cut exact algorithm adapted from the single location case (Section 4). Finally, we provide a computational assessment on the effectiveness of our techniques, also yielding insights on optimal calibration of parameters for real-world cloaking systems (Section 4.7).

2. Problem definition and modeling

The technique known as *cloaking* can be summarized as follows. The user selects some locations as *sensitive* and when he is close to a sensitive location, the information about his position is purposely made imprecise. Spatial cloaking is used, for instance, in the PROBE system [2] [4]: the area under study is represented by a grid, and every cell of the grid has an associated probability of the user being in it. The user is allowed to select some non overlapping and connected subsets of cells, called *base subsets*, as sensitive; when the user is located close to a base-cell, the information about his position is cloaked, i.e. only the information about a region around him is made available. The region includes the base-cell and the cell where the user is located but also other cells. The obfuscation level obtained in this way is measured by a sensitivity measure of the regions, as formally defined below, that estimates the probability of detecting the true position of the user.

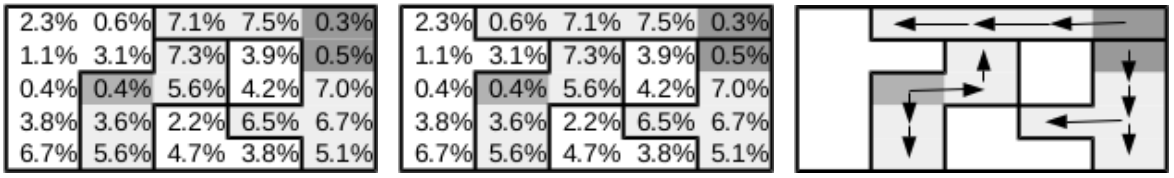
In its simplest version, cloaking is applied to each single base-cell independently, and the goal is to compute a region of limited size around it, whose sensitivity does not exceed a given threshold. In the most general version, instead, many base-cells are given at once, adjacent ones forming base subsets. Each base subset must be covered by properly selected regions, avoiding overlaps between regions. The size of the reference space and the overlaying grid, typically representing the urban context where the movement takes place, is an application-dependent parameter. Such parameter can be determined based on the region size, spatial resolution, system architecture. Just as an example the experiments reported in [2] and conducted on real mapping data, use a cell resolution of 20 meters: a grid whose side is composed by a few hundreds cells is enough to cover a metropolitan area like that of Milan.

In Figure 1a a sample 5x5 area is depicted, representing a single base-cell cloaking instance. The numbers on each cell represent the probability of a user being in it; a sensitive location is marked in dark gray. In Figures 1b and 1c two possible cloaking solutions are reported: when the user approaches the sensitive location, its position is obfuscated by returning only the region delimited with black borders. Intuitively, the solution in Figure 1c offers more protection, as the probability of a user being in the sensitive location, knowing he is in the region, is smaller. In Figures 2a and 2b, instead, a multi base-cell instance is depicted. Being adjacent, the two base-cells at top right form a single base subset. Possible cloaking solutions are represented by the regions identified by black borders: two and three regions are depicted in 2a and 2b, respectively.



(a) (b) (c)

Figure 1 Sample single base-cell cloaking instance.



(a) Solution using 2 regions. (b) Solution using 3 regions. (c) Graph representation.

Figure 2 Sample multi base-cell cloaking instance.

2.1. Modeling

Let $G = (N, \mathcal{E})$ be a graph representing the area grid, that is having one vertex for each cell, and one edge between vertices corresponding to adjacent cells. Except for borders, each cell has four adjacent ones, and therefore the degree of each vertex in N is at most four. For each cell $i \in N$, a probability p_i is given for a user being in it. Without loss of generality we assume $\sum_{i \in N} p_i = 1$.

Sensitivity of the regions. We define the *sensitivity* $\sigma(R)$ of any region $R \subseteq N$ as the conditional probability of a user being in a base-cell, given he is in the region; that is:

$$\sigma(R) = \begin{cases} \frac{\sum_{i \in R} s_i p_i}{\sum_{i \in R} p_i} & \text{if } \sum_{i \in R} p_i \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

where the binary datum s_i indicates whether cell $i \in N$ is a base-cell or not. The sensitivity of a region R represents the cloaking level that obfuscates the base-cells in R. Smaller values of $\sigma(R)$ correspond to higher obfuscation levels. A region with no base-cells has null sensitivity.

We denote as $\tau \in (0, 1]$ the sensitivity threshold. For each region R containing at least one base-cell, we formulate a sensitivity constraint as follows:

$$\sigma(R) \leq \tau; \quad \frac{\sum_{i \in R} s_i p_i}{\sum_{i \in R} p_i} \leq \tau$$

which is well defined because $\sum_{i \in R} p_i$ is always strictly positive. Now we can rewrite the constraint as

$$\sum_{i \in R} (s_i - \tau) p_i \leq 0.$$

We associate a binary variable x_i with each cell $i \in N$, indicating whether cell i belongs to a selected region or not, and we rewrite the sensitivity constraint as

$$\sum_{i \in N} (s_i - \tau) p_i x_i \leq 0.$$

To further simplify the notation we introduce a coefficient $w_i = (\tau - s_i) p_i$ for each cell $i \in N$. The sensitivity constraint is now

$$\sum_{i \in N} w_i x_i \geq 0$$

for each region. We note that since $s_i \in \{0, 1\}$ and $\tau \in (0, 1]$, $w_i \leq 0$ for each base-cell, and $w_i > 0$ for any other cell. In the remainder we denote as "favorable" and "unfavorable" the cells with positive and negative weight respectively, because including them in R helps satisfying the sensitivity constraint or makes it more difficult.

In a feasible solution sensitivity constraints must be respected for each base-cell. For instance, by fixing $\tau = 1.99\%$, the solution in Figure 1b is infeasible, while that of 1c is feasible, having respectively $\sigma(R) = 2.00\%$ and $\sigma(R) = 1.78\%$.

Topology and cost of the regions. To guarantee that the selected regions are connected, we represent them as arborescences in $G = (N, \mathcal{E})$, rooted at base-cells. Regions must not overlap, and therefore arborescences must be disjoint. Furthermore, cells in the same base subsets can be split in different regions, but no base-cells of different base subsets can belong to the same region, and therefore no arborescence is allowed to connect vertices of different base sets. We also make the assumption that portions of the same base subset belonging to the same region must be adjacent. Our aim is to design regions that are as small as possible, and therefore we define as cost of each arborescence the number of its vertices, that is the number of cells in the corresponding region. A solution is therefore a collection of arborescences; appealing solutions optimize the trade-off between the number of regions and the sum of their costs. For instance, in Figure 2c, the arborescences corresponding to the solution of Figure 2b are depicted.

Related literature. Related problems deal with the computation of optimal arborescences in graphs. The Steiner Tree/Arborescence Problem (SP) is a well known NP-hard optimization problem, for which several exact (see [5, 6]) and heuristic algorithms have been devised: for a recent review we refer to [7].

A useful starting point for the design of an algorithm for the single base-cell problem is the paper by Ljubic et al. [8], that addresses the problem of computing a Steiner tree on a graph with costs on the edges and prizes on the vertices, where the objective function is to minimize the sum of the overall cost of the edges belonging to the tree and the overall prize of the vertices not belonging to the tree. This problem is called Prize-Collecting Steiner Tree Problem (PCSTP). Our single base-cell problem turns out to be a PCSTP with some special characteristics: (i) all edges have unit cost (and this induces the existence of a lot of equivalent solutions); (ii) all vertices of the graph are Steiner vertices; (iii) we do not have a mixed objective function, but we have an additional constraint on the sensitivity of the region (arborescence): edge costs are in the objective, while vertex prizes are in the sensitivity constraint; (iv) vertex prizes can be positive or negative, according to the contribution of the vertices to the sensitivity constraint: base-cells are unfavorable, i.e. their inclusion increases the sensitivity of the region; (v) the region is represented by an arborescence rooted at a base-cell. To the best of our knowledge, no previous attempt has been made to tackle this specific variant, neither from a heuristic nor from an exact optimization point of view.

3. The single base-cell problem

In the single base-cell problem a base-cell corresponding to a single node $b \in N$ is given and the goal is therefore to compute a minimum cost arborescence rooted at b, complying with the sensitivity constraint.

3.1. Formulation: a 0-1 ILP model

To obtain a provably optimal solution we use a mathematical programming model based on the following notation. We consider the general setting in which $\mathcal{G} = (N, \mathcal{E})$ is a graph, not necessarily a grid, with vertex set N and edge set $\mathcal{E}$. We transform the graph into an equivalent digraph by replacing each edge $\{u, v\} \in \mathcal{E}$ with a pair of arcs (u, v) and (v, u) and we indicate by $\mathcal{A}$ the resulting arc set. We indicate by $\delta^-(S)$ the set of arcs entering any vertex subset $S \subset \mathcal{V}$, i.e. $\delta^-(S) = \{(u, v) \in \mathcal{A} : u \in \bar{S}, v \in S\}$, where $\bar{S} = N \backslash S$. We indicate

the vertex corresponding to the base-cell by r and the weight of each vertex $v \in N$ by w_v. We use binary variables x_v associated with each vertex $v \in N$ to indicate whether the vertex belongs to the region or not; we also use binary variables y_{uv} associated with each arc $(u,v) \in \mathcal{A}$ to indicate whether u is the predecessor of v along the path from r to v.

$$\text{minimize } z = \sum_{(u,v)\in\mathcal{A}} y_{uv} \tag{1}$$

$$\text{subject to } \sum_{u\in N:(u,v)\in\mathcal{A}} y_{uv} = x_v \qquad \forall v \in N\backslash\{r\} \tag{2}$$

$$\sum_{v\in N} w_v x_v \geq 0 \tag{3}$$

$$\sum_{(u,v)\in\delta^-(S)} y_{uv} \geq x_s \qquad \forall S \subset N\backslash\{r\},\ \forall s \in S \tag{4}$$

$$x_r = 1 \tag{5}$$

$$y_{uv} \in \{0,1\} \qquad \forall (u,v) \in \mathcal{A} \tag{6}$$

$$x_v \in \{0,1\} \qquad \forall v \in N. \tag{7}$$

The objective function (1) requires to minimize the cardinality of the region; constraints (2) state the relationship between variables x and variables y: a vertex v belongs to the region (i.e. $x_v = 1$) if and only if it is the head of a selected arc (i.e. $y_{uv} = 1$ for some vertex u); constraints (3) impose that the overall weight of the region complies with the sensitivity constraint; constraints (4) impose that the region is connected; constraints (5) impose that the region includes the base-cell; finally constraints (6) and (7) require the integrality of the variables.

This model resembles the one proposed by Ljubic et al. [8] for the prize-collecting Steiner tree problem; our adaptations cope with the five main differences outlined in the previous Section. The algorithm we have designed and tested is also based on a relaxation adapted from [8]: we initially relax constraints (4) and integrality restrictions (6) and (7), we solve the resulting linear programming (LP) problem and we detect violated constraints (4) by a separation algorithm; some violated constraints are then inserted into the LP model and the procedure is iterated until the optimal LP solution satisfies all constraints (4).

3.2. Relaxation: an LP model

The model of the linear relaxation we iteratively solve in our branch-and-cut algorithm is the following:

$$\text{minimize } z = \sum_{(u,v)\in\mathcal{A}} y_{uv} \tag{8}$$

$$\text{subject to } \sum_{u\in N:(u,v)\in\mathcal{A}} y_{uv} = x_v \qquad \forall v \in \mathcal{V}\backslash\{r\} \tag{9}$$

$$\sum_{v\in N} w_v x_v \geq 0 \tag{10}$$

$$\sum_{(u,v)\in\delta^-(S)} y_{uv} \geq x_s \qquad \forall S \in \mathcal{S}, \forall s \in S \tag{11}$$

$$x_r = 1 \tag{12}$$

$$x_u \geq y_{uv} \qquad \forall (u,v) \in \mathcal{A} \tag{13}$$

$$\sum_{(r,v)\in\mathcal{A}} y_{rv} \geq 1 \tag{14}$$

$$2y_{uv} + 2y_{vu} \leq x_u + x_v \qquad \forall (u,v) \in \mathcal{A} \tag{15}$$

$$0 \leq y_{uv} \leq 1 \qquad \forall (u,v) \in \mathcal{A} \tag{16}$$

$$0 \leq x_v \leq 1 \qquad \forall v \in N. \tag{17}$$

The set $\mathcal{S}$ of connectivity constraints is initially empty. This model is obtained from the linear relaxation of model (1)-(7) by relaxing connectivity constraints (4), which are exponential in number, and by inserting constraints (13), (14) and (15) to reinforce the formulation. Constraints (13) state that an arc can be used only if the node corresponding to its tail belongs to the region. Constraints (14) state that at least one arc must leave the root. Constraints (15) forbid cycles of cardinality 2.

Since connectivity constraints (4) have been relaxed, it is possible that the optimal solution of model (8)-(17) is disconnected and contains cycles. Violated connectivity constraints (11) are dynamically generated by an exact separation algorithm illustrated in the remainder. According to our notation, they are inserted in $\mathcal{S}$. However, owing to the relaxation of the integrality requirements (6) and (7), the optimal solution of the linear program with connectivity constraints is not guaranteed to be integer and acyclic. For this reason we finally resort to branching, when necessary.

3.3. Pre-processing

Before starting the branch-and-cut algorithm we run a pre-processing routine, whose goal is to identify and discard some vertices of the graph that cannot be part of any optimal solution, thus reducing its size. This goal is achieved by visiting the graph with a breadth-first-search algorithm, starting from the root vertex r. Every vertex u is labeled with the maximum weight of a path from r to u; the weight of a path is the sum of the weights of the vertices along the path. The weight of r is negative, because r corresponds to a base-cell; as soon as the weight of a path P emanating from r becomes non-negative, the algorithm is stopped: a feasible solution has been detected and the number of arcs along P is a valid upper bound. Therefore all vertices whose distance from r is larger than P cannot belong to any optimal solution and are discarded.

3.4. Separation

The separation problem is a min-cut problem, which can be solved to optimality in polynomial time. We indicate with x^* and y^* the values of the x and y variables in the current optimal solution of model (8)-(17). For each vertex $u \in N$ with $x^*_u > 0$ we search for a minimum cut separating r from u in the support digraph where each arc $(u, v) \in \mathcal{A}$ has capacity equal to y^*_{uv}. We indicate by C_S the capacity of a (r, u)-cut separating $S \subset N$ from its complement $\overline{S} \subset N$, with $u \in S$ and $r \in \overline{S}$. Any cut S with $C(S) < x^*_u$ corresponds to a violated connectivity constraint; with a little abuse of terminology, we call it *violated cut* in the remainder.

To compute minimum cuts we used the efficient algorithm by Cherkassky and Golberg [1]. However it is important to remark that there are several possible strategies that can be used in the cut generation process. It is common experience that adding more than one constraint per iteration usually speeds up cutting planes algorithms. Therefore, for each destination vertex $u \in N$, it is convenient to find multiple violated connectivity constraints, i.e. multiple violated (r, u)-cuts. For this purpose we also generated *nested cuts* and *back cuts* [8] and we made computational tests to suitably tune the separation strategy.

Nested cuts are obtained as follows. Assume a violate cut has been found; then all arcs belonging to the cut are given capacity equal to 1, so that they cannot belong to a bottleneck cut anymore; then the max-flow min-cut algorithm is run again and another violated cut is possibly found. The procedure is iterated and it generates a sequence of

violated cuts, initially close to r and then closer and closer to u. When the same technique is executed sending the flow from u to r, *back (nested) cuts* are generated.

We compared four separation strategies, all generating nested cuts. In strategies A and B the choice between forward and backward nested cuts is made at random with the same probability at each iteration and for each (r, u) pair; on the contrary, in strategies C and D nested cuts are always generated both forward and backward. In strategies A and C the generation of cuts stops as soon as the connected component of the target vertex is reached, while in strategies B and D the cut generation procedure stops only when the target vertex is reached (by target vertex we mean u for forward cuts and r for backward cuts). The goal is to reach a (possibly fractional) optimal solution in which all connectivity constraints are satisfied as early as possible. Besides tightening the lower bound, this also allows to run heuristics at each node of the branch-and-bound tree and in turn this allows to early compute good feasible solutions and to effectively prune the search. Fine tuning and strategy selection issues are discussed in Section 3.7.

3.5. Heuristics

At every node of the branch-and-bound tree we run a fast heuristic algorithm, when the cut generation procedure is over. The algorithm works on the subgraph $(N^*, \mathcal{A}^*)$ defined by vertices $u \in N$ with $x^*_u > 0$ and the arcs $(u, v) \in \mathcal{A}$ with $y^*_{uv} > 0$. This support graph is guaranteed to be connected but it may not correspond to an arborescence. If any node in N^* turns out to be the head of more than one arc in $\mathcal{A}^*$, then we arbitrarily choose a path from r to u in the support graph and we delete from $\mathcal{A}^*$ all arcs entering u and not belonging to the path. After this pre-processing every non-root vertex u has a unique predecessor $\pi(u)$ and the support graph is an r-arborescence. Since the region formed by cells in N^* satisfies the sensitivity constraint, we are guaranteed that $\sum_{u \in N} w_u x^*_u \geq 0$ and then $\sum_{u \in N^*} w_u \geq 0$ as shown in Subsection 2.1. We formulate the problem of selecting a minimum cardinality r-arborescence in $(N^*, \mathcal{A}^*)$ still complying with the sensitivity constraint. The corresponding ILP model requires a binary variable t_{uv} for each arc in $\mathcal{A}^*$ and reads as follows.

$$\text{minimize} \sum_{(u,v) \in \mathcal{A}^*} t_{uv}$$

$$\text{subject to } w_r + \sum_{(u,v) \in \mathcal{A}^*} w_v t_{uv} \geq 0$$

$$t_{uv} \leq t_{\pi(u)u} \qquad \forall (u, v) \in \mathcal{A}^* : u \neq r$$

$$t_{uv} \in \{0,1\} \qquad \forall (u,v) \in \mathcal{A}^*.$$

Preliminary experiments verified that the solution of this model with state-of-the-art general purpose solvers takes negligible computing time.

3.6. Policies for branching and searching

We included a binary branching policy: we consider the last optimal solution of the linear program obtained when the cut generation procedure is over and we select the x variable whose current optimal value is closest to $1/2$. Then we create two subproblems by fixing the variable respectively to 0 and to 1.

We compared four tree exploration policies: best-first, depth-first, breadth-first and a mixed strategy. This latter one is motivated by the observation that fixing a variable to 1 is more effective than fixing it to 0. Therefore subproblems generated by fixings to 1 (1-nodes) are given precedence with respect to those originated from fixings to 0 (0-nodes): every time a branch occurs the 1-node is inserted in the first position in the queue of open nodes while the 0-node is appended in the last position. From our preliminary tests, we observed that best-first tends to be the most effective strategy and this was selected for the final extensive set of computational tests.

3.7. Computational results

We separately performed a training phase, whose aim was to understand the contribution of each component of the algorithm to its overall performance, and the best parameters settings, and a test phase, whose aim was to assess the performance of the algorithm on realistic instances. All tests have been performed on a PC equipped with an i7 2.6 GHz CPU and 16 GB of RAM; each test was restricted to use a single core. The algorithms were coded in C++, compiled with the GNU C++ compiler version 4.3.1 with full optimization flags. Maximum flow computations were performed using the implementation of [19], that is made publicly available by its authors; LP subproblems were optimized with the commercial solver CPLEX 12.6. The R environment [22] was used to generate problem instances; R is a GNU project, released open source. An implementation of our instance generator is available online [21].

For the sake of comparison, we have implemented a benchmark branch-and-cut algorithm fully using CPLEX, including constraints (4) as lazy cuts. The largest instance we could solve with that approach involved a grid of 5x4, that was too small to produce any

significant insight. In fact, from grids of 5x5 onwards, each run went out of memory. At the same time, optimizing 5x5 instances is trivial for our methods. Having verified that the simple lazy cuts approach is largely outperformed by our branch-and-cut, we did not proceed to any further testing with it.

3.7.1. Training phase. For the training phase we generated a random dataset (T in the remainder), creating instances without specific structure as follows.

- The grid is a square with size ranging from 10×10 to 25×25 (that is, graphs with 100 to 625 vertices).
- 30% of its cells are set to have zero probability; Intuitively, these cells represent zones of the map whose access is either restricted or simply impossible.
- the remaining cell values are initially set to a random value, drawn as an integer from a uniform distribution between 1 and 10;
- the sum of these values is then normalized to one, in such a way that the value in each cell represents the probability of a user being in it.
- The base-cell is chosen at random among the cells with non zero probability.
- Each instance is tested with three values for τ, namely 0.1, 0.3 and 0.5.

Five instances have been created for each combination of grid size and τ values.

Fine tuning of the cut generation procedure. We tested all strategies on a set of instances corresponding to 25×25 grids in this way: a node u is selected at random; cuts are generated with one of the four strategies and inserted in the relaxed model; the linear program is re-optimized; the procedure is repeated until a timeout of 200 seconds expires. A typical outcome is represented in Figures 3 and 4, and summarized in Table 1. In particular, the chart in Figure 3 (resp. 4) includes one line for each strategy, and reports execution time on the x axis and value of LP relaxations (resp. number of generated cuts) on the y axis. Table 1, instead, is composed by one column for each strategy, and two rows, reporting the average lower bound achieved and the average number of constraints generated by each strategy within the timeout.

Strategy C turns out to be the most effective in tightening the lower bound. From further computational experiments on larger instances we could observe that the best trade-off between the number of LP iterations and the number of constraints generated depends on the instances. When instances are easier (i.e. smaller), strategies B and D which generate a larger number of cuts, are faster. When the instance is more difficult, it

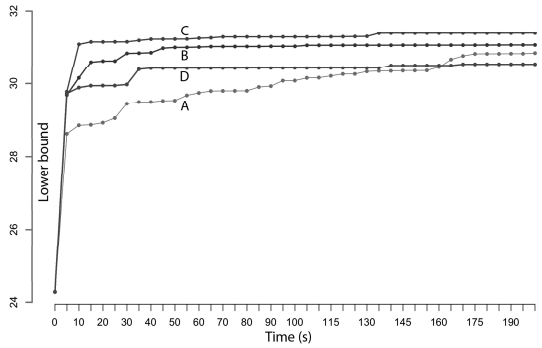


Figure 3 **Comparison between the four separation strategies: lower bound as a function of computing time.**

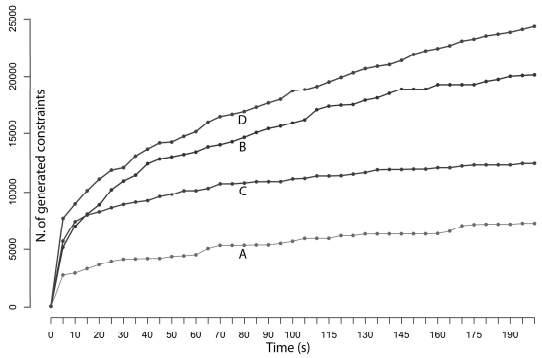


Figure 4 **Comparison between the four separation strategies: number of generated constraints as a function of computing time.**

	A	B	C	D
Lower bound	30.86	30.95	31.57	30.82
N. cuts	7188	20073	12364	24256

Table 1 **Final lower bound and number of cuts with different separation strategies.**

is convenient to generate less constraints and to optimize the LP more often, with strategy C. Computational results reported in the remainder have been obtained with strategy C.

Effectiveness evaluation. Then, we performed a detailed analysis of the effect of the different components of the algorithm with two aims: searching for the best configuration, and understanding the relative contribution of each component to the overall performance of the algorithms. As an overall result, we could verify that indeed, including pre-processing, heuristics, generation of valid cuts, and selecting a best-bound-first search policy yields best overall performances. In order to provide a synthetic, yet meaningful, report of our findings, we performed the following a posteriori experiment: starting from the best configuration, we de-activated each component of the algorithm, and measured the corresponding performance worsening. A timeout of two hours was set for this experiment. Our results are summarized in Tables 2, 3, 4, 5, 6. Each table contains one row for each combination

of threshold value (τ) and grid border number of cells (size), and one column for the following three versions of the algorithm: using depth-first instead of best-first as tree search policy (DFS), de-activating pre-processing (No Prep.), de-activating primal heuristics (No Heur.); we have also tried to de-activate the generation of cuts, but the lower bounds produced by our formulations were too weak to yield meaningful results. Each cell of the tables reports the average worsening on the instance class indicated in its row, when the versions of the algorithm indicated in the leading column is used in place of the full one. In details, table 2 reports the number of additional instances hitting the timeout, while tables 3, 4, 5, 6 report worsening factors in computing time, number of explored nodes in the branching tree, number of max flow subproblems solved, and number of generated cuts, respectively; these are computed as the ratio between the performance of either DFS, No Prep. or No Heur., and the performance of the full algorithm, restricting to those instances solved within the timeout. It can be observed that both the tree search policy and the ad hoc heuristics have a strong impact in the overall performance of the algorithm. By turning them off a strong increase in average computing time is observed, and a substantial number of instances could not be solved within the timeout anymore. Heuristics seem to be effective in reducing the number of search tree nodes to explore, as better upper bounds allow for early pruning. Best-fist search, in turn, avoids to optimize nodes that require the generation of several cuts, and the solution of several flow subproblems; intuitively, these may correspond to nodes lying deeper in the search tree, in which many branching decisions have been taken. An overall increase in computing time of about 55% and one more timed-out instance are observed by turning pre-processing off, even if the number of visited nodes is lower. Indeed, each node requires more computational effort, especially in terms of CPU time for solving subproblems.

3.7.2. Test phase. Subsequently, we defined two realistic datasets, identified by A and B in the remainder, reproducing areas with a single sensitive location or multiple sensitive locations respectively. Data were generated at random as follows.

- The grid is a square with size of either 15×15, 20×20 or 25×25; Intuitively, we assume that points of interest exist on the region, and that people tend to cluster in their neighborhoods. Therefore its cells are tentatively populated with the probability values p given by a bivariate Gaussian distribution, centered in a cell chosen at random with

τ	size	DFS	No Prep.	No Heur.
0.10	10	0	0	0
	15	-2	0	-2
	20	-2	0	-3
	25	-2	0	-2
0.30	10	0	0	0
	15	-2	0	-2
	20	-3	0	-4
	25	-1	0	-1
0.50	10	0	0	0
	15	0	0	-1
	20	-2	0	-2
	25	-4	-1	-3

Table 2 Worsening as different components of the algorithm are de-activated - number of solved instances within time limit.

τ	size	DFS	No Prep.	No Heur.
0.10	10	1904.98	0.79	1142.57
	15	8.40	1.23	3534.76
	20	795.97	2.89	392.30
	25		2.31	
0.30	10	6638.40	0.15	4081.28
	15	147.50	1.08	73.00
	20	11410.70	1.12	49814.50
	25		1.07	
0.50	10	181.02	0.58	294.60
	15	69778.50	2.96	114393.48
	20	8247.61	2.88	0.65
	25	1.80	1.75	0.92
Avg.		13807.66	1.55	18201.80

Table 3 Worsening as different components of the algorithm are de-activated - computing time.

τ	size	DFS	No Prep.	No Heur.
0.10	10	2.03	0.94	24.60
	15	9.00	0.96	37.67
	20	2.00	0.83	2.33
	25		1.00	
0.30	10	0.92	0.47	15.51
	15	1.00	0.78	2.33
	20	3.00	0.87	7.00
	25		1.00	
0.50	10	2.36	0.73	7.80
	15	1.27	0.60	7.83
	20	1.44	0.47	1.00
	25	0.67	0.33	1.00
Avg.		2.37	0.72	13.35

Table 4 Worsening as different components of the algorithm are de-activated - number of explored branching tree nodes.

uniform probability distribution in the grid, and having as covariance matrix a diagonal one, whose terms are half the grid side.

τ	size	DFS	No Prep.	No Heur.
0.10	10	127.15	0.81	125.78
	15	7.45	1.00	69.88
	20	112.89	1.58	188.00
	25		0.96	
0.30	10	900.47	0.94	206.90
	15	42.67	0.85	61.67
	20	285.07	0.88	588.33
	25		1.20	
0.50	10	42.84	0.96	44.12
	15	722.48	0.83	566.04
	20	1322.17	1.10	1.00
	25	2.00	1.00	1.00
Avg.		431.07	0.99	173.36

Table 5 Worsening as different components of the algorithm are de-activated - number of calls to the max flow algorithm.

τ	size	DFS	No Prep.	No Heur.
0.10	10	16.90	1.02	18.73
	15	3.01	1.23	8.86
	20	1414.64	1.12	1810.00
	25		1.45	
0.30	10	1199.43	1.10	659.30
	15	414.00	1.08	500.00
	20	5051.40	1.01	8795.00
	25		0.95	
0.50	10	203.83	0.99	260.92
	15	3378.31	0.94	3824.76
	20	2614.23	1.35	1.00
	25	2.00	1.00	1.00
Avg.		1412.26	1.10	1037.68

Table 6 Worsening as different components of the algorithm are de-activated - number of generated cuts.

• In dataset A, 30% of the grid cell values are subsequently reset to 0.0; the sum of the remaining ones is then normalized to 1. The base-cell is chosen at random among the cells with non zero probability. That is, we do not assume that sensitive locations are always main points of interest, as otherwise obfuscation is not likely be an issue.

• In dataset B we generate at random a rectangular *base subset* with size 2×2 (dataset B1) or 4×4 (dataset B2); then 30% of the grid cells are selected at random, among those that do not belong to the base subset, and their value is reset to 0.0; the sum of the remaining values is normalized to 1, and the base-cell is chosen at random among those in the base subset. Also in this case, the base subset may or may not include points of interest.

• Three instances are created for each grid size in dataset A, and each of them is tested with three values for τ, namely 0.3, 0.5 and 0.7. In datasets B, instead, two instances are

Type	size	τ	root time (s)	root cuts	root gap (%)	B&B time (s)	B&B nodes
A	15	0.3	0.64	3087.33	3.55	2.98	8.00
		0.5	0.61	4456.00	0.00	1.34	0.00
		0.7	0.70	5731.00	0.56	1.44	0.67
	20	0.3	15.36	11404.3	2.46	224.24	15.33
		0.5	7.50	18183.6	0.00	14.83	0.00
		0.7	6.11	25971.3	0.98	15.13	1.33
	25	0.3	18.22	24737.3	1.91	317.32	11.33
		0.5	4.22	28722.6	0.00	28.52	0.00
		0.7	8.53	55327.3	0.49	60.88	2.67
B1	15	0.05	4.44	4336.50	24.31	43.48	6.00
		0.1	0.06	1381.50	24.29	1.51	25.00
		0.2	0.08	1448.00	32.50	28.24	56.00
		0.4	0.08	1423.00	25.00	160.29	74.00
	20	0.05	22.47	13419.5	4.17	104.66	2.00
		0.1	18.06	7252.50	0.00	236.23	0.00
		0.2	0.19	566.50	0.00	0.32	0.00
		0.4	0.14	1001.00	0.00	0.60	0.00
	25	0.05	9.77	17375.5	6.58	2513.11	152.00
		0.1	2.52	6301.00	0.00	5.72	0.00
		0.2	0.50	3686.00	0.00	2.80	0.00
		0.4	0.52	3233.50	0.00	2.58	0.00
B2	15	0.05	7.95	6678.00	25.00	464.10	47.00
		0.1	1.16	2672.50	25.00	16.24	15.00
		0.2	0.22	1683.00	41.67	61.44	71.00
		0.4	0.13	1566.00	32.50	92.05	54.00
	20	0.05	7.17	9564.50	4.55	47.55	2.00
		0.1	7.21	5382.50	8.34	120.20	11.00
		0.2	0.20	719.00	0.00	7.05	0.00
		0.4	6.02	236.00	0.00	3.99	0.00
	25	0.05	14.55	20895.0	18.99	645.61	47.00
		0.1	7.65	8847.50	16.41	330.73	70.00
		0.2	2.75	5281.00	12.50	165.16	42.00
		0.4	0.66	3096.00	0.00	7.98	0.00

Table 7 Aggregated results on single base-cell instances.

created for each grid size, and each of them is tested with four values for τ, i.e. 0.05, 0.1, 0.2 and 0.4.

As a result, dataset A is characterized by vertices with non-negative weights $w_i = p_i$ only, while vertices in the base subsets of dataset B can have negative weights, being $w_i = (\tau - s_i)p_i$ with $s_i = 1$ for the cells of the base subset, 0 otherwise. The sensitivity constraint for dataset A requires $\sum_{i \in N} p_i x_i \geq \tau$, while that for datasets B requires $\sum_{i \in N} w_i x_i \geq 0$.

Our results are summarized in Table 7, that includes in turn the time spent, the number of generated cuts and the final gap at the root node, the time spent in exploring the branching tree, and the number of explored nodes; each row reports average results over instances having identical class, grid size and threshold value, as indicated in the first three columns. Detailed results on each instance are reported in the Online Supplement.

Remarkably, many instances were solved at the root node. Instances in datasets B are harder to solve to optimality; this can be seen for instance from the values in the "Gap" columns. The gap also tends to be larger when the base subset is larger (dataset B2). The reason is that the fractional optimal solution of the relaxed model tends to combine a lot of variables; besides enlarging the primal-dual gap this also requires to generate more cutting planes. This is particularly evident when τ is small (i.e. $\tau = 0.05$) and the probability of the root vertex is relatively large, because in this case it is necessary to include many vertices in the arborescence to satisfy the sensitivity constraint.

4. The multi base-cell problem

As discussed in the introduction, in the most general version of cloaking many base-cells can be defined in the area, forming base subsets; appealing solutions cover all base-cells with many regions of limited size. However, maximizing the number of regions and minimizing their overall size are two conflicting objectives. Therefore, we tackle this generalization with a bi-objective optimization approach. For each suitable number of regions k, we search for the minimum number of cells $\mathcal{P}_k$ to be included in the regions to cover all base-cells and respect sensitivity constraints. Once all $\mathcal{P}_k$ values are computed, different synthetic trade-off measures can be selected to choose a particular solution, like the average region size $\mathcal{P}_k/k$.

The number of solution values $\mathcal{P}_k$ to compute can be in principle large, but can be substantially reduced by exploiting the following two observations.

Observation 4.1. There is no feasible solution using less regions than the number of base subsets ℓ, as no region can cover base-cells of different subsets.

Observation 4.2. When an instance is feasible, an optimal solution always exists in which the root of each arborescence lies in the border of a base subset.

In fact, suppose to have an optimal solution in which an arborescence is rooted in an inner cell of a base subset. In order to satisfy the sensitivity constraint, the root must be connected to non-base outer cells; therefore at least one directed path exists from the root to outer cells, thereby crossing the border of the base subset in a particular base-cell. Hence, an equivalent optimal solution can be obtained by arbitrarily selecting one of those paths, considering the subpath from the root to the base-cell on the border, and reversing its arcs.

Observation 4.3. For each suitable number of regions $k > \ell$, $\mathcal{P}_{k-1} \leq \mathcal{P}_k$.

In fact, one can always obtain a solution in which $k-1$ regions are defined by a solution with k regions, by simply merging two regions with adjacent base-cells. Since sensitivity constraints hold for each of them independently, they hold also after merging, being actually replaced by a single surrogate constraint. That solution might, of course, be suboptimal.

Let $\bar{\ell}$ be the number of base-cells on the border of base subsets. Summarizing, we compute the Pareto-optimal frontier of the bi-objective problem by searching for all optimal $\mathcal{P}_k$ values with a bisection process: starting with $k' = \ell$ and $k'' = \bar{\ell}$, we compute $\mathcal{P}_{k'}$ and $\mathcal{P}_{k''}$; if $\mathcal{P}_{k'} = \mathcal{P}_{k''}$ then all values $\mathcal{P}_k$ with $k' \leq k \leq k''$ are equal to $\mathcal{P}'_k$, and therefore we stop;

otherwise we select $k = \lfloor (k'' + k')/2 \rfloor$ and we proceed recursively on the ranges $[k', k]$ and $[k, k'']$, stopping when $k = k'$.

A key issue remains on how to minimize $\mathcal{P}_k$ for a fixed k. Adapting the branch-and-cut designed for the single base-cell problem turns out to be hard, as many observations that are at the basis of it do not hold anymore. Instead, the combinatorial structure of the problem lends itself to a very natural decomposition, since the only way in which the trees interact is the no-overlaps constraint. This can be easily exploited by either Lagrangian relaxation or Dantzig-Wolfe reformulation methods; in both cases, the complexity of the full problem is managed by iteratively solving a set of minimum cardinality arborescences complying with the sensitivity constraint, for which the algorithm we have presented in the previous section for the single base-cell problem proved to be successful. Therefore, in the following we detail our branch-and-price approach, yielding an exact algorithm for the multi base-cell case.

4.1. Mathematical formulation

As in the previous section we indicate as $G = (N, \mathcal{A})$ the graph corresponding to the grid, and as w_i the weight associated with each node $i \in \mathcal{N}$. The set of base subsets is indicated by $\mathcal{R}$; base subsets are disjoint. For each base subset $r \in \mathcal{R}$ we indicate by $\mathcal{B}_r \subseteq N$ the corresponding subset of base-cells; the set of all the base-cells is indicated by $\mathcal{B} = \cup_{r \in \mathcal{R}} \mathcal{B}_r$. For each base-cell $b \in \mathcal{B}$ we indicate by Θ_b the set of all feasible arborescences rooted at b, that are those satisfying the sensitivity constraints without including cells of other base sets.

The union of all the subsets of columns is indicated by Θ. The required number of arborescences is indicated by k.

A binary variable λ_l is associated with each column (arborescence) $l \in \Theta$. A binary coefficient x_{ibl} indicates whether vertex $i \in N$ belongs to the arborescence rooted at vertex $b \in B$ in column $l \in \Theta_b$. With this notation the master problem reads as follows:

$$\text{minimize } z = \sum_{r \in \mathcal{R}} \sum_{b \in \mathcal{B}_r} \sum_{l \in \Theta_b} \sum_{i \in N} x_{ibl} \lambda_l \tag{18}$$

$$\text{s.t. } \sum_{r \in \mathcal{R}} \sum_{b \in \mathcal{B}_r} \sum_{l \in \Theta_b} x_{ibl} \lambda_l \leq 1 \qquad \forall i \in N \tag{19}$$

$$\sum_{r \in \mathcal{R}} \sum_{b \in \mathcal{B}_r} \sum_{l \in \Theta_b} x_{ibl} \lambda_l \geq 1 \qquad \forall i \in \mathcal{B} \tag{20}$$

$$\sum_{r\in\mathcal{R}}\sum_{b\in\mathcal{B}_r}\sum_{l\in\Theta_b}\lambda_l = k \tag{21}$$

$$\lambda_l \text{ binary} \qquad\qquad \forall r\in R\ \forall b\in B_r\ \forall l\in\Theta_b \tag{22}$$

We consider the linear relaxation of this master problem, where the integrality conditions (22) are replaced by constraints $0\le\lambda_l\le 1\ \ \forall r\in R\ \forall b\in B_r\ \forall l\in\Theta_b$. Because of constraints (19) it is redundant to impose $\lambda_l\le 1$. Hence constraints (22) are simply replaced by non-negativity conditions.

We indicate the (non-positive) dual variables of constraints (19) with μ_i, the (non-negative) dual variables of constraints (20) with γ_i and the (free) dual variable of constraint (21) with ξ. Hence the reduced cost of a column $l\in\Theta_b$ for any $b\in\mathcal{B}$ is:

$$\overline{c}_{bl} = \sum_{i\in N} x_{ibl} - \sum_{i\in N}\mu_i x_{ibl} - \sum_{i\in\mathcal{B}}\gamma_i x_{ibl} - \xi$$

that is

$$\overline{c}_{bl} = \sum_{i\in N} x_{ibl}(1-\mu_i-\gamma_i) - \xi$$

with $\gamma_i = 0$ for all non-base-cells.

Further, since the number of arborescences is exponential in the size of the graph, only a subset of them is actually inserted in a restricted master problem. In order to guarantee that the linear relaxation of the restricted master problem is always feasible, we insert a dummy column with a cost equal to $|N|+1$ and such that it satisfies all constraints of the master problem, i.e. it covers all base-cells in constraints (20) and it counts as k arborescences in constraint (21). To accelerate the convergence of the column generation algorithm we also insert columns corresponding to "empty arborescences" made by a single base-cell. Since they are infeasible (a root alone cannot satisfy the sensitivity constraint), they also have a very large cost like the dummy column.

4.2. Pricing

The linear relaxation of the master problem is solved via column generation. When a fractional optimal solution is reached we resort to branching. The pricing subproblem is solved with the branch-and-cut algorithm illustrated for the single base-cell problem. The only difference is that in the pricing subproblem of the multi base-cell problem there are also penalties on the vertices: they are the current values of the dual variables μ and γ. The pricing algorithm is run for each base-cell and the corresponding optimal solutions are inserted in the restricted master problem if their reduced cost is negative.

4.3. Symmetry breaking

The pre-processing procedure outlined for the single base-cell problem is no longer applicable in the multi base-cell case, because several arborescences may compete for the same favorable cells. Hence a feasible solution with disjoint arborescences is not guaranteed not to use paths longer than the minimum path found by the breadth-first-search algorithm.

However we pre-process the instances in a different way, to reduce the symmetries due to the different base-cells in the same base subset. First, Observation 4.2 allows us to directly avoid pricing for cells in the interior of a base set. Furthermore, since we also made the assumption that no disjoint subsets of base-cells belong to the same arborescence, it is sufficient to keep the arcs of a cycle along the borders of each base subset, deleting those in the opposite direction.

4.4. Greedy pricing

Before running the branch-and-cut algorithm for exact pricing, we run a greedy heuristic pricing routine: starting from the base-cell the arborescence is iteratively extended with the reachable cell of minimum weight until the sensitivity constraint is satisfied. After that, the arborescence is possibly further extended with negative reduced cost cells. To speed up the search for the minimum weight cell among those that can be appended to the arborescence, we use a heap data-structure. The effectiveness of the greedy pricing algorithm in producing good solutions is well documented by the computational results of the branch-and-price based heuristics described in subsection 4.6.

4.5. Branching

When the column generation algorithm is over and the optimal solution of the linear relaxation of the restricted master problem is fractional, we resort to branching in three different ways.

Branching 1: roots. We consider all the candidate roots, i.e. the cells along the borders of the base subsets. For each candidate root b we compute $\sum_{l \in \Theta_b} \lambda_l$ and we select the value that is closest to $1/2$. Then we generate two subproblems with a binary branching operation. In a subproblem we constrain b to be a root: this is achieved by deleting all arcs entering b from other base-cells of the same base subset. In this way b is forbidden to be included in any arborescence rooted elsewhere and therefore it is constrained to be a

root. In the other subproblem we forbid b to be a root: all columns in Θ_b are deleted from the master problem and the pricing algorithm is no longer executed from cell b in that branch.

Branching 2: base-cells. When all values of $\sum_{l\in\Theta_b}\lambda_l$ are integer, then we resort to a second branching strategy. We consider all pairs of base-cells i and b in the same base subset $\mathcal{B}_r$ and we compute $\sum_{l\in\Theta_b} x_{ibl}\lambda_l$. We select the value that is closest to $1/2$ and we generate two subproblems with a binary branching operation. In a subproblem we forbid i to belong to an arborescence rooted at b by fixing $x_{ib}=0$ in the pricing subproblem for root b. In the other subproblem we constrain i to belong to an arborescence rooted at b: this is achieved by fixing $x_{ib'}=0$ in the pricing subproblems corresponding to all roots $b'\neq b$.

Branching 3: non-base-cells. When the values of $\sum_{l\in\Theta_b} x_{ibl}\lambda_l$ are integer for all base-cells i, then we resort to a third branching strategy, which is identical to the second one but considers non-base-cells. For each non-base-cell i and each base-cell b we compute $\sum_{l\in\Theta_b} x_{ibl}\lambda_l$. We select the value that is closest to $1/2$ and we generate two subproblems with a binary branching operation. In a subproblem we forbid i to belong to an arborescence rooted at b by fixing $x_{ib}=0$ in the pricing subproblem for root b. In the other subproblem we forbid i to belong to any arborescence not rooted at b: this is achieved by fixing $x_{ib'}=0$ in the pricing subproblems corresponding to all roots $b'\neq b$. This third branching strategy does not partition the solution space of the current subproblem: solutions in which i is not part of any arborescence remain feasible in both branches.

4.6. A branch-and-price-based heuristics

To cope with the combinatorial explosion and the exponential increase in computing time, we also devised a heuristic algorithm to compute good solutions quickly, optimizing also instances that are out of reach for the exact optimization branch-and-price. The heuristic algorithm relies upon the structure of the branch-and-price algorithm where the pricing sub-problem is solved heuristically. The price is of course to lose any optimality guarantee, including that the final solution of the linear restricted master problem is a valid lower bound. Nevertheless the algorithm produces a well diversified set of "good" columns and often achieves the optimal solution. Branching acts as a diversification strategy, forcing the

search to explore all the solution space, while heuristic pricing acts as an intensification mechanism, producing feasible solutions of small cardinality. We tested two versions of this branch-and-price-based heuristics: in one case we only use the greedy pricing algorithm described in subsection 4.4; in the other case we also use the branch-and-cut algorithm described in subsection 3 truncated at the root node. When the root node is solved to optimality, the heuristic procedure described in 3.5 produces a feasible arborescence.

4.7. Computational results

Our algorithms were implemented in C++ and compiled with the GNU C++ compiler version 4.3.1 with full optimization flags, using CPLEX 12.6 as LP solver and SCIP 3.1 [20] as framework for branch-and-cut-and-price algorithms. The latter framework is released open source for academic research. Pricing problems were solved with an adaptation of the code used in subsection 3.7. As before, tests were run on a PC equipped with a i7 2.6 GHz CPU and 16 GB of RAM, using a single core. An implementation of our instance generator is also available online [21]. As in the single base-cell problem, we have experimented with a benchmark implementation of a branch-and-cut directly using CPLEX on a suitable variant of formulation (1) – (7), setting the counterpart of constraints (4) as lazy cuts, verifying this approach not to be viable.

The instances for testing the branch-and-price algorithm were generated with the same technique described in subsection 3.7 for datasets B. The size of the grid was set to 15×15 when searching for optimal solutions. We used one or two base subsets of size 2×2 or one base subset of size 3×3. When generating instances with more than one base subset, the random generation is repeated until no pair of base subsets overlaps. To offer a clear view on how selecting the number of regions affects the computing time of branch-and-price, we run a test for all suitable values of the number of arborescences k, ranging from the number of base subsets (1 or 2) to the number of base-cells on the borders of the base subsets (4 or 8). We also used different values of the sensitivity threshold τ.

Tests on larger grids were also carried out with the aim of producing heuristic solutions by the branch-and-price based heuristic algorithm. In these cases we used grid of size up to 30×30, up to 5 base subsets of size 2×2 or 3×3, resulting into a number of arborescences up to 24.

We present aggregate results in Table 8 and detailed results in the tables collected in the Online Supplement. The first column in Table 8 defines how data have been aggregated.

Aggregation criterion	Time (exact)	Time (trunc.)	N.cells (exact)	N.cells (trunc.)
$\tau = 0.05$	4850.13	75.53	27.14	27.29
$\tau = 0.10$	6571.68	78.63	19.57	19.64
$\tau = 0.20$	1838.51	15.63	15.36	15.36
$\tau = 0.40$	754.54	12.45	12.71	12.71
One base subset 2×2	548.15	18.77	12.31	12.38
Two base subsets 2×2	6126.46	24.68	22.25	22.25
One base subset 3×3	2525.16	103.67	19.75	19.88
$k = \lvert\mathcal{R}\rvert$	408.63	8.40	9.75	9.75
$k \approx \lvert\mathcal{R}\rvert$	2738.36	14.05	14.12	14.12
$\lvert\mathcal{R}\rvert < k < \lvert\overline{\mathcal{B}}\rvert$	4202.64	86.24	15.31	15.37
$k \approx \lvert\overline{\mathcal{B}}\rvert$	3345.59	33.09	21.54	21.60
$k = \lvert\overline{\mathcal{B}}\rvert$	1278.18	24.64	23.09	23.18

Table 8 Aggregated results.

The we report the average computing time to solve the full problem to proven optimality, the average computing time to solve the full problem with a heuritic pricing algorithm, in which only the root node of each branch-and-cut tree is optimized, the average cardinality of the optimal solution (overall number of cells in the arborescences) and the average cardinality of the heuristic solution.

The first section of Table 8 shows the same effect already observed with the single base-cell problem: as the threshold τ becomes large, the average size of the solutions tends to decrease, because few non-base-cells are sufficient to meet the sensitivity requirement. As a consequence it is less likely that arborescence "compete" for using the same favorable cells and the instances become easier to solve; for $\tau = 0.20$ or larger, heuristic solutions are often optimal.

The second section of Table 8 shows the effect of the number of base subsets $|\mathcal{R}|$. When $|\mathcal{R}|$ increases there are more base-cells in $\overline{\mathcal{B}}$ and, as expected, this increases the number of runs of the pricing algorithm and the computing time required by the branch-and-price algorithm. The same effect is produced by different sizes of the base subset.

The third section of Table 8 shows the effect of the number of arborescences k. We indicate by $|\mathcal{R}|$ the cardinality of the set of base subsets; this is a lower bound for k because at least one arborescence is needed for each base subset. We indicate by $|\overline{\mathcal{B}}|$ the cardinality of subset $\overline{\mathcal{B}}$ that includes the base-cells on the borders of their regions; this is an upper bound for k, because w.l.o.g. roots are not allowed to be in the interior of the base subsets. We indicate as $k \approx \alpha$ when $|k - \alpha|/\alpha < 0.5$. It is apparent that extreme values of k, i.e. $k = |\mathcal{R}|$ and $k = \overline{\mathcal{B}}$ correspond to easier instances. This is because the number of feasible choices for the subset of roots is larger when k is far from the extreme values. This yields

a remarkable increase in the number of negative reduced cost columns to be considered, a larger number of pricing iterations and a larger number of fractional values in the optimal solution of the linear restricted master problem, which in turn requires a larger number of branching levels in the branch-and-price tree.

Restricting to heuristic pricing allows to trade a limited worsening in solutions quality for a strong CPU time reduction.

5. Conclusions

The exact optimization algorithm we have devised for the single base-cell and the multi base-cell problems allowed to study the trade-off between the obfuscation level, represented by the value of the parameter τ, and the computational hardness of the resulting combinatorial optimization instance. Our tests confirmed that the value of the threshold τ strongly affects the cardinality of the resulting obfuscated regions and the computing time, as expected.

In the multi-arborescence problem the user can suitably calibrate the number of required arborescences which results in fine-grained or coarse-grained representation of the sensitive subsets of cells to be obfuscated. We observed a significant relationship between the number of arborescences k and the computing time. When k is close to the maximum or minimum values, instances turn out to be easier to solve to optimality. However the effect of the combinatorial explosion is quite evident, as expected. A research question remains open, concerning whether alternative exact optimization paradigms might yield better performances in some particular cases.

To cope with large instances we have devised a branch-and-price based heuristic algorithm which is able to consistently find optimal or near optimal solutions in a computing time which is reduced by three to four orders of magnitude with respect to that required for exact optimization.

Acknowledgments

The authors wish to thank two anonymous referees, whose valuable comments helped to improve the paper.

References

[1] B.V. Cherkassky, A.V. Goldberg, *On implementing push-relabel method for the maximum flow problem*, Algorithmica 19 (1997) 390-410.

[2] M.L. Damiani, E. Bertino, C. Silvestri, *The PROBE Framework for the Personalized Cloaking of Private Locations*, Transactions on Data Privacy 3(2) (2010) 123-148.

[3] M.L. Damiani, E. Bertino, C, Silvestri, *Protecting Location Privacy through Semantics-aware Obfuscation Techniques*, Trust Management II - Proceedings of IFIPTM 2008: Joint iTrust and PST Conferences on Privacy, Trust Management and Security, Trondheim, Norway (2008).

[4] M.L. Damiani, C. Silvestri, E. Bertino, *Fine-Grained Cloaking of Sensitive Positions in Location-Sharing Applications*, IEEE Pervasive Computing 10(4) (2011) 64-72.

[5] S. Chopra, E. Gorres, M.R. Rao, *Solving a Steiner tree problem on a graph using branch and cut*, ORSA Journal on Computing 4 (1992) 320-335.

[6] T. Koch, A. Martin, *Solving Steiner tree problems in graphs to optimality*, Networks 32 (1998) 207-232.

[7] J.W. van Laarhoven, *Exact and heuristic algorithms for the Euclidean Steiner tree problem*, Thesis, University of Iowa, 2010.

[8] I. Ljubic, R. Weiskircher, U. Pferschy, G.W. Klau, P. Mutzel, M. Fischetti, *An Algorithmic Framework for the Exact Solution of the Prize-Collecting Steiner Tree Problem* Mathematical Programming 105 (2006) 427-449

[9] G. Ghinita, *Privacy for Location-Based Services,Morgan* Claypool Publishers (2013)

[10] M.L. Damiani, *Location privacy models in mobile applications: conceptual view and research directions*, GeoInformatica 18(4) (2014) 819-842

[11] B. Lee, J. Oh, H. Yu, and J. Kim *Protecting location privacy using location semantics* proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data miningÂ (2011). ACM, New York, NY, USA, 1289-1297.

[12] S. Song, Z. Zou and K. Liu, *Semantic-aware Location Privacy Preservation on Road Networks* Proceeding of the International Conference on Database Systems for Advanced Applications (2016)

[13] R. Shokri, G. Theodorakopoulos, J.-Y. Le Boudec, J.P. Hubaux, *Quantifying Location Privacy* IEEE Symposium on Security and Privacy (2011) 247-262

[14] C. A. Ardagna, M. Cremonini, S. De Capitani di Vimercati and P. Samarati, *An Obfuscation-Based Approach for Protecting Location Privacy*, IEEE Transactions on Dependable and Secure Computing 8(1) (2011), 13-27.

[15] J. Krumm *A survey of computational location privacy* Personal Ubiquitous Comput. 13(6) (2009): 391-399

[16] K. P. N. Puttaswamy et al., *Preserving Location Privacy in Geosocial Applications*, IEEE Transactions on Mobile Computing, 13(1) (2014): 159-173

[17] D. Christin, A. Reinhardt, S.l S. Kanhere, M. Hollick, *A survey on privacy in mobile participatory sensing applications*, Journal of Systems and Software, 84 (11) (2011) 1928-1946

[18] X. Pan, W. Chen L. Wu , C. Piao, Z. Hu *Protecting personalized privacy against sensitivity homogeneity attacks over road networks in mobile services*, Frontiers Computer Science 10(2) (2016) 370-386

[19] Y. Boykov, V. Kolmogorov, *An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision*, proceedings of IEEE Transactions on Pattern Analysis and Machine Intelligence (2004).

[20] T. Achterberg, *SCIP: solving constraint integer programs*, Mathematical Programming Computation 1(1) (2009) 1-41.

[21] *Mathematical Programming Algorithms for Spatial Cloaking*, companion website, `http://homes.di.unimi.it/ceselli/GSTP` (2016)

[22] *R framework website*, *https://www.r-project.org*, last access July 2016.