Solving Geometric Covering Problems by Data Reduction

Steffen Mecke and Dorothea Wagner*

Dep. of Computer Science, University of Karlsruhe, {mecke, wagner}@ira.uka.de

Abstract. We consider a scenario where stops are to be placed along an already existing public transportation network in order to improve its attractiveness for the customers. The core problem is a geometric set covering problem which is $\mathcal{NP}$-hard in general. However, if the corresponding covering matrix has the consecutive ones property, it is solvable in polynomial time. In this paper, we present data reduction techniques for set covering and report on an experimental study considering real world data from railway systems as well as generated instances. The results show that data reduction works very well on instances that are in some sense "close" to having the consecutive ones property. In fact, the real world instances considered could be reduced significantly, in most cases even to triviality. The study also confirms and explains findings on similar techniques for related problems.

1 Introduction

The continuous stop location problem consists of placing new stations or stops along a given public transportation network in order to reach all potential customers and thus to improve attractiveness of using public means of transport. In particular, this scenario occurs in local public transport where additional stops in a rapid transit system or bus system, e.g. close to a residential or a settlement may induce significant increase of customers. Of course, the placement of new stops has certain additional effects to be considered as well. New stations require investment and operational costs and the travel time for customers already using a bus or train on the affected line might increase. On the other hand, the total number of customers and the according gain is improved and the total door-to-door travel time might be decreased.

As part of a project with the largest German railway company, we focused on a scenario where only the reachability of the network for potential customers was considered. That is, we are given a geometric graph, a (possibly empty) set of stations placed on the graph, a set of demand points in the plane and a radius. Weights are assigned to all edges and vertices of the graph representing the increase of travel time caused by new stop placed there. Then a demand point is covered by a (possibly new) station if the station is within the given radius

* The authors acknowledge financial support from the EU for the RTN AMORE under grant HPRN-CT-1999-00104.

of that point. The problem consists in finding a set of stations with minimum total weight placed on the graph such that all demand points are covered. In [1] and [2] this problem is introduced as the continuous stop location problem and proved to be $\mathcal{NP}$-hard. It is also shown, that the problem can be formulated as a set covering problem by reducing the set of potential stations to a finite set of candidates.

Set covering problems have been studied widely, for example in the context of crew scheduling. See [3] for an overview. As shown for example in [4], the set covering problem is notoriously hard to solve or even to approximate. However, it is observed already in [1] that set covering is solvable in polynomial time by linear programming if the covering matrix has the so-called consecutive ones property. In the geometric setting, the special case that the network consists of only one single line satisfies the consecutive ones property. Polynomial solvability of some similar special cases of related problems is also shown in [5].

Experiments with railway data indicate that the size of such instances can be reduced significantly by applying a simple reduction technique mentioned already in [6]). Therefore we examine the power of data reduction for geometric set covering instances from the station location scenario as well as on generated instances. In addition to the known reduction techniques we apply a more advanced reduction rule and a refined enumeration algorithm to solve weighted set covering problems. It turns out that all real world instances could be reduced almost completely. Even the largest instances were reduced within one minute leaving an instance that could usually be solved within a few seconds.

One explanation for this nice behaviour is that the matrices occurring in our setting are close to having consecutive ones property or band diagonal form. In comparison, we conducted experiments on generated matrices (nearly) having consecutive ones property and found a similar behaviour. The computational results suggest that the effectiveness of data reduction highly depends on "closeness" of the matrix to having consecutive ones property. For the real world data considered, closeness to the consecutive ones property is obviously affected by the underlying geometry. The study also confirms and explains results reported in [7] about the power of data reduction for similar covering problems.

In the following section, well known reduction techniques for set covering are introduced and our new advanced reduction technique is presented. Efficient algorithms to implement these techniques are given and their behaviour is analyzed especially for instances with consecutive ones property. In Section 3 a refined enumeration algorithm for set covering is introduced and its running time is analyzed. Section 4 presents our experimental study and discusses the results.

2 Data Reduction for Set Covering

2.1 Basic Definitions

An *instance* $I := (P, F, A, w)$ of our problem is given by two finite sets P and F with $m := |P|$, $n := |F|$, a positive weight function w on F and a $m \times n$–

matrix $A = (a_{ij})_{i \in P, j \in F}$ over $\{0,1\}$. We want to find a *covering* of P with minimum weight, i.e. a subset $F' \subseteq F$ such that for each $p \in P$ there is $f \in F'$ such that $a_{pf} = 1$ and $\text{cost}(F') := \sum_{f \in F'} w_f$ is minimized.

If $a_{pf} = 1$ for a row p and a column f of A, then p is *covered* by f. For subsets $F' \subseteq F$ and $P' \subseteq P$ denote

$$\text{cov}(F') := \{p \in P |\ f \in F',\ a_{pf} = 1\}$$
$$\text{precov}(P') := \{f \in F |\ p \in P',\ a_{pf} = 1\}\ .$$

A set $F' \subseteq F$ such that $\text{cov}(F') = P$ is called a *feasible solution* for I. It is called an *optimal solution* for I if $\text{cost}(F')$ is minimum. The set of all optimal solutions will be denoted by $\mathcal{OPT}(I)$ and the weight of an optimal solution by $\text{cost}(I)$. An instance has a feasible solution if and only if every row of A contains at least one 1. We will assume in the following that all instances are solvable. As an instance of Set Covering is fully described by its covering matrix A and the weight function w, we also write $I = (A, w)$ as an abbreviation. In the following let $N := |\{a_{ij} = 1\}|$.

In the context of stop location P corresponds to the demand points in the plane (population areas) and F to the set of facilities (stations). Usually, an instance of the stop location problem is given by the geometric graph, the set of stations, the set of demand points in the plane and a radius. It is obvious that the covering matrix A can be easily derived from this. Moreover, we consider cases where no set of stations is given explicitly. Instead, a station can be placed on any (or given) positions on the graph. As already mentioned in the introduction, such an instance can be transformed into an equivalent instance with a finite set of stations.

Definition 2.1. *A matrix over* $\{0,1\}$ *has the* strong consecutive ones property*, if the ones in every row are consecutive. It has the (simple)* consecutive ones property *(C1P) if its columns can be permuted such that the resulting matrix has the strong consecutive ones property.*

For a matrix with C1P, a permutation of its columns to induce a matrix with strong consecutive ones property can be found in linear time using PQ–trees (cf. [8]).

2.2 Reduction

Definition 2.2. *For columns f and g of A, f is* dominated *by g if either* $\text{cov}(f) \subseteq \text{cov}(g)$ *and* $w_f \geq w_g$ *or* $\text{cov}(f) = \emptyset$*. For rows p and q of A, p is* dominated *by q if* $\text{precov}(q) \subseteq \text{precov}(p)$*.*

For an instance (P, F, w, A) we use the following terminology:

- If there are rows $p \neq q$ such that row p is dominated by q, we call the instance *reducible by rows*. If there are columns $f \neq g$ such that column f is dominated by g, we call the instance *reducible by columns*. An instance that is neither reducible by rows nor by columns is called *completely reduced*.

– A *reduction step* is denoted by a triple (M, i, k), where $M \in \{row, col\}$, i, k denote two rows (resp. columns) and i is dominated by k.
– Let $\sigma = (row, i, k)$ (resp. $\tau = (col, i, k)$) denote a reduction step. By σA (resp. τA) we denote the matrix resulting from deleting row (resp. column) i. The according instance is denoted by σI (resp. τI).

It is a well known fact (see e.g. [6]) that dominated rows and columns can be deleted without changing solvability. Even more, the value of an optimal solution is maintained. More precisely:

Lemma 2.3. *Let σ be a reduction step for an instance I. Then we have*

$$\text{cost}(\sigma I) = \text{cost}(I) \text{ and } \mathcal{OPT}(\sigma I) \subseteq \mathcal{OPT}(I) \ .$$

Accordingly, in order to solve an instance of Set Covering one can first apply a sequence of reduction steps and then solve the (completely) reduced instance. It can be even shown that a completely reduced instance is in a certain sense unique:

Theorem 2.4. *Let $\mathcal{I} = (\mathcal{A}, \gamma)$ an instance and $I = (A, c)$ and $J = (B, d)$ two completely reduced instances derived from $\mathcal{I}$. Then we have, for a suitable permutation π of the rows and columns of A*

$$\pi A = B \text{ and } \pi c = d \tag{1}$$

and thus

$$\text{cost}(I) = \text{cost}(\mathcal{I}) = \text{cost}(J) \tag{2}$$
$$\mathcal{OPT}(I) \subseteq \mathcal{OPT}(\mathcal{I}) \supseteq \mathcal{OPT}(J) \tag{3}$$
$$|OPT(I)| = |OPT(J)| \ . \tag{4}$$

We omit a formal proof here. Note that (2) already follows from Lemma 2.3. Equation (1) follows from the fact that, for two different reduction steps σ and τ for a matrix A, either $\tau A = \sigma A$ or τ is still a valid reduction step for σA. Using this observation one can derive a bijection between the sequences of reduction steps leading to I and J respectively. □

2.3 Advanced Reduction

In many cases, a column is dominated in the sense that a combination of two or more other columns would dominate it. Consider a column f and a set of columns G such that $\text{cov}(f) \subseteq \text{cov}(G)$ and

$$w_f \geq \sum_{g \in G} w_g \ .$$

Then f is *dominated by* G. If a column f is dominated by a set of columns G, we call its deletion *advanced reduction step*. However, it seems computationally

infeasible to consider for a column f all combinations of other columns. Instead we apply the following approach. For every row r and column f denote $f_{\min}(r)$ a column in precov(r) with minimal weight. Then column f is dominated by g together with the set of columns $\{f_{\min}(r) | r \in \text{cov}(f) - \text{cov}(g)\}$ if $\text{cov}(f) \cap \text{cov}(g) \neq \emptyset$ and

$$w_f \geq w_g + \sum_{r \in \text{cov}(f) - \text{cov}(g)} w_{f_{\min}(r)} \ .$$

For this advanced reduction, analogous results to Lemma 2.3 and Theorem 2.4 can be proved. Note however, that by applying advanced reduction certain types of solutions might be lost. Consider for example the covering matrix $\left(\begin{smallmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{smallmatrix}\right)$ with weights $(2, 1, 1)$. It is not reducible by simple reduction, optimal solutions are $S_1 = \{1\}$ and $S_2 = \{2, 3\}$. By advanced reduction it can be reduced to $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, with only a two element optimal. In the next section we will show that the advanced reduction can be implemented without increasing the asymptotic worst-case running time of the reduction.

2.4 Reduction Algorithm

We shortly review the algorithmic realization of reduction. In the following we consider a matrix obtained by reductions consisting of reduction steps as well as advanced reduction steps.

Theorem 2.5. *A matrix can be reduced by columns in $O(nN)$ and by rows in $O(mN)$. Furthermore a matrix can be reduced completely in $O(Nmn)$.*

Proof. Compare each pair of columns and delete dominated columns. If the matrix is represented as n adjacency lists, comparing one column to all others has cost $O(N)$, which leads to $O(nN)$ steps. Note that for advanced reductions the $f_{\min}(r)$ have to be computed only once. Obviously, the cost for computing the $f_{\min}(r)$ is as well within $O(nN)$. Reduction by rows is analogous. For complete reduction of a matrix, reductions by rows and reductions by columns are applied alternating until no reduction step at all is applicable. In each reduction step at least one row or one column is deleted. Accordingly, the procedure is finished after $O(\min(m, n))$ alternations. This induces cost $O(Nmn)$ for complete reduction. □

Remark 2.6. If the matrix is very sparse the analysis can be improved further. Let the number of ones in every row and every column be smaller than a constant c, then every column has to be compared with at most c^2 other columns, where each comparison has cost $O(c)$. Then the matrix is reduced by columns within $O(c^3 n)$ reduction steps. As before it follows that complete reduction can be done in $O(nm)$.

Remark 2.7. Theorem 2.5 can be slightly improved using results of [9] and [10]: These papers present a lower bound and algorithms for the problem of finding extremal sets, which is equivalent to reduction by column (resp. rows). In [9] a

lower bound of $\Omega(N^2/\log^2(N))$ is proved, and an algorithm with time complexity $O(N^2/\log(N))$ is given in [10] which is only slightly better than the running time of our much simpler approach.

2.5 Instances with Consecutive Ones Property

Instances with C1P can be reduced very effectively. A matrix with C1P maintains this property after every reduction step. If the weights are all equal it can be even reduced to the unit matrix. For the weighted case, reduction can still be done very efficiently by establishing strong C1P and then sorting the rows lexicographically.

Remark 2.8. Note that a reduced matrix with C1P for rows has C1P for columns.

Set Covering for matrices with C1P can be solved by applying a shortest path algorithm in a graph with n nodes as described in [11]. In our experiments, we used Algorithm 1 which is exponential for general instances but polynomial on instances with C1P.

3 A Refined Enumerative Algorithm

For solving a completely reduced set covering problem, it is favorable to apply an algorithm that takes into account the (observed) special structure of the real world instances. More precisely, we aim at an algorithm that is especially effective for instances that are "close" to having C1P. Although the well-established approaches for solving integer linear programs as e.g. branch and bound are very successful in practice, they are often disadvantageous for nicely structured instances.

Alternatively, we designed the following simple, however refined enumerative solution algorithm. The basic idea is simply to enumerate all solutions by combining the covering columns for each row. This leads to partial solutions for all the rows considered so far and finally to a solution for the entire instance. This procedure is described in Algorithm 1.

Throughout Algorithm 1 the cover of partial solutions is maintained in matrix B, the weights of the columns in array w', and the partial solutions in array S. It always terminates, because in every iteration at least one row is removed from C. Before each iteration the following invariants hold by induction:

- For every column f of B
  - the partial solution S_f covers all rows deleted in step 2 up to this iteration and the rows with a one in column f of B.
  - S_f has weight w'_f.
- S contains a subset of an optimal solution for I.

The key argument is that step 1 adds all solutions covering the first row. The correctness of the returned solution follows if $C = (1)$.

Algorithm 1: Combi solver

Input : Instance $I = (P, F, w, \mathsf{A})$
Output : Optimal solution
Start
  $\mathsf{B} := (0) \in \{0,1\}^{|P| \times 1}$, weights $w' := (0)$, solutions $S := (\emptyset)$;
  $C := \left(\begin{array}{c|c} \mathsf{A} & \mathsf{B} \\ \hline 0 & 1 \end{array}\right)$;
  while $C \neq (1)$ **do**
    forall the *columns f of B with 0 in first row* **do**
      forall the *columns g of A with 1 in first row* **do**
1   Add a new column $A_g + B_f$ to B, a weight $w_g + w'_f$ to w' and solution $S_f \cup \{g\}$ to S;
      Remove column f;
2   Remove first row of A and B ;
    Reduce $C := \left(\begin{array}{c|c} \mathsf{A} & \mathsf{B} \\ \hline 0 & 1 \end{array}\right)$ and remove corresponding entries of w' and S;
  return S_1 *and* w'_1;
End

In order to keep the effort for computing the partial solutions reasonable, we apply again our reduction technique throughout the algorithm. Of course, the running time of Algorithm 1 is exponential in general. The size of matrix B, which contains all the partial solutions obtained so far can be still exponentially large. For certain kinds of instances, however, we can prove some nice properties:

Theorem 3.1. *If the covering matrix A has strong C1P, is reduced, the rows are sorted in lexicographic order, and c is the maximum number of ones per row and per column. Then Algorithm 1 has time complexity $O(c^3 n)$.*

Proof. Let c be the maximum number of ones in a column or row of A. If the covering matrix A has strong C1P, is completely reduced, and the rows are sorted in lexicographic order it has, as noted in remark 2.8, also strong C1P for columns. This implies that after each iteration, the first column of B contains only zeroes, all other columns have strong C1P, contain a one in the first row, and no two columns are equal. Thus, B has at most c columns. So during Algorithm 1 for each column of A a new column is added to B at most c times. Each reduction is in $O(c^3)$ as stated in Theorem 2.5. □

Even for a weaker property than C1P Algorithm 1 has polynomial running time:

Theorem 3.2. *If the distance between the first and last entry in each column of the covering matrix is at most k, then at any time throughout Algorithm 1 the number of columns in matrix B is in $O(2^k n)$. The time complexity is in $O(mn^2 2^{2k} k)$.*

Proof. We prove by induction that after each iteration, the ones in B are contained in rows 1 to $k-1$. As B is completely reduced, no two columns are equal, so there are at most 2^k different columns. In every iteration only those columns of A with a one in the first row are considered. The ones of these columns are again contained in rows 1 to k. Therefore for every column of B at most n new columns are added, having all their ones in rows 1 to k. After addition of all new columns there are at most $2^k n$ columns in B and there are at most $2^k nk$ ones in B. So by Theorem 2.5 reduction by columns is in $O((2^k n)2^k nk)$. There are at most m iterations which leads to the claimed upper bound. □

As Theorem 3.2 indicates, the ordering of the rows is crucial for the running time of the algorithm. We have made good experiences with the Cuthill-McKee ordering (cf. [12]). This means that the rows are chosen in their order of appearance in a breadth first search on the *covering graph* with adjacency matrix $\left(\begin{smallmatrix} 0 & A \\ A^T & 0 \end{smallmatrix}\right)$. A simple on-line-heuristic which always tries to minimize the size of B in the next step leads to even better running times.

4 Experimental Results

The implementation is in `C++` and runs on a i686-Linux platform. The experiments were performed on a Xeon(TM) CPU with 2.80GHz and 2GB RAM.

4.1 Instances

Real World Instances. We tested our algorithm with different data sets. The original motivation of this work was a project in collaboration with the largest German railway company. Its goal was to optimize the accessibility of customers to local train service. We therefore had access to the data on the German railway network (circa 8200 nodes and 8700 edges) and German settlements (circa 30000). We tested different radii. According to the railway company, radii of two to five kilometers are the most realistic.

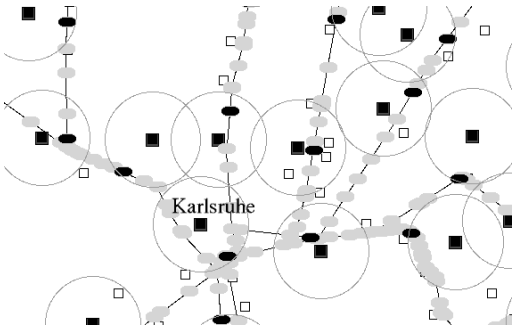
We also had two different versions of the problems. The first version contains all settlements that are within the given radius of the railway network, the second version contains only those settlements that were not already covered by one of the (approx. 6800) existing German railway stations. The edges and nodes were weighted by the number of customers traveling through them.

Generated Instances. As a second data set we considered randomly generated instances with C1P and suitably modified instances having almost C1P. We experimented with unweighted instances (all weights equal to 1) and instances where the weights were chosen randomly (equally distributed) between 1 and a maximum weight of 10 or 100. The matrices consist of 50 to 50000 rows with 10 to 200 non-zero entries per row. Up to 20% of the ones were randomly flipped to zero resulting in perturbed instances.

4.2 Performance of Reduction

Real World Instances. The original problem consisted in solving instances that contain only those settlements that are not already covered by existing railway stations. Our experiments on practical data showed that these are always reduced to almost trivial instances in very short time.

Moreover these instances tend to become easier to solve with increasing radius. For large radii many settlements are already covered by old stations and the instance decomposes into many, small components.



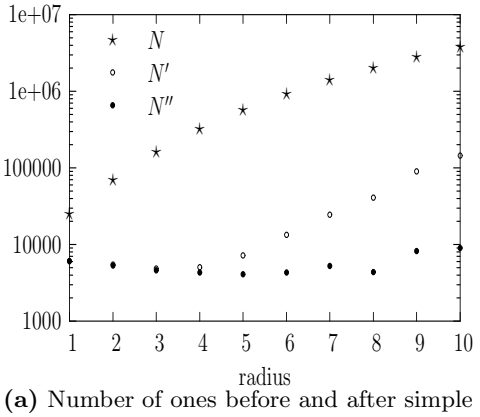
The situation is different for instances without existing stations. As an illustrative example for the effect of reduction see the figure on the left. It shows a section of the railway graph near the city of Karlsruhe. The covering radius was 2.5km. The settlements are squares, potential candidates for stops are ellipses. The original instance is colored grey, the instance remaining after reduction is black. Obviously the solution for the shown part of the remaining instance is already trivial. The largest remaining component of the covering matrix after reduction contained 8 rows and 9 columns.

With increasing radius, however, these instances get quite complex in terms of total matrix size and number and size of independent components. However, reduction is still very effective here. Figure 1 summarizes reduction rates for the German railway graph and radii between 1 and 10km. In these examples the advanced reduction technique is much more effective, especially for large radius. It even seems that the size of the reduced matrix does not grow with increasing radius.

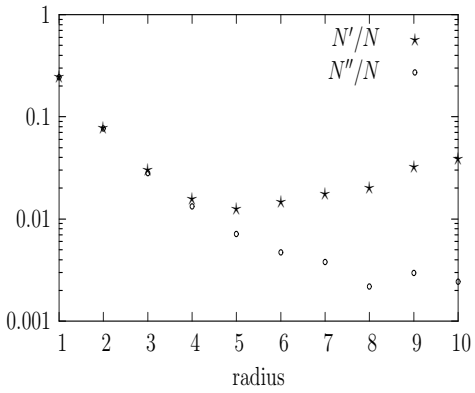
Generated Instances. When looking at the C1P matrices, we first observe that the unweighted instances can be reduced completely, that is, to the trivial unit matrix. The weighted matrices could not be reduced completely by the first step, but almost completely by the improved reduction.

Perturbed instances are harder to reduce. For weighted, perturbed instances simple reduction is not very effective but advanced reduction yields very good results as Figure 2 illustrates. Figure 3(a) shows that unweighted instances with 10% perturbation could usually be reduced by a factor of 10, instances with 20% only by a factor of around 2.

Increasing the Radius. Of course, N grows with increasing radius, but also the combinatorial structure of the instances changes. This affects the effectiveness of reduction. As r increases the reduction ratio decreases first, increases for moderate radii and finally drops to almost zero for very large radii (The left side

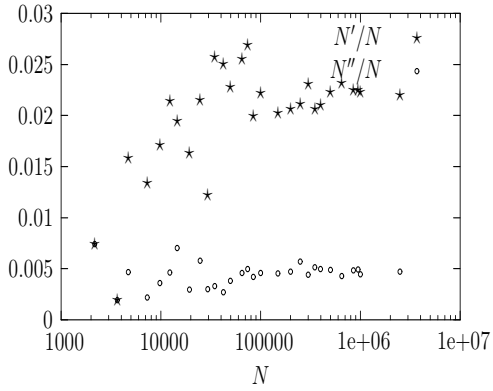


(a) Number of ones before and after simple (N') and advanced reduction (N'').



(b) Corresponding reduction ratios.

Fig. 1. Effectiveness of reduction for the German railway graph.



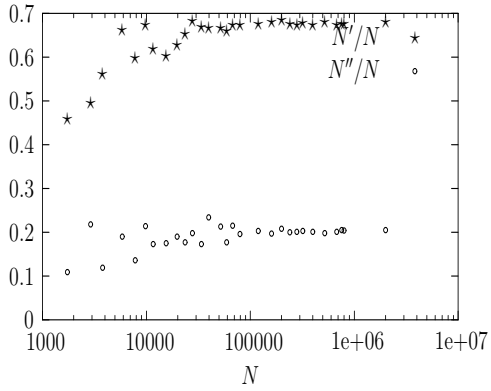


Fig. 2. Reduction ratios for unperturbed resp. 20% perturbed generated instances.

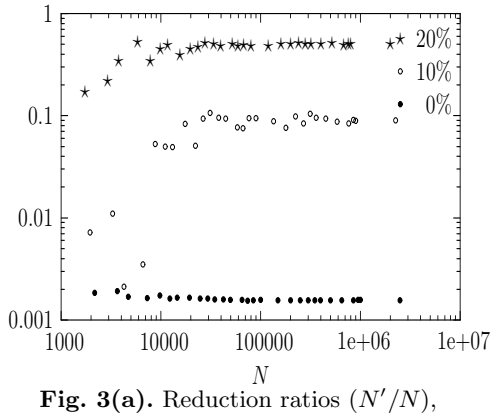


Fig. 3(a). Reduction ratios (N'/N), unweighted generated instances.

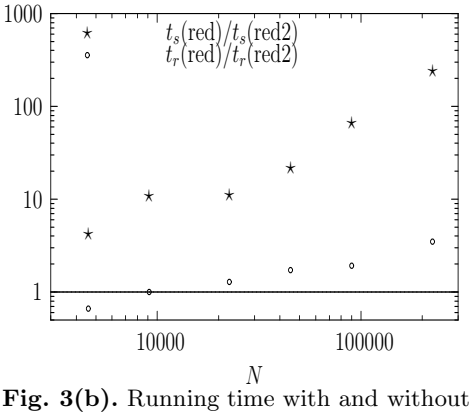


Fig. 3(b). Running time with and without advanced reduction.

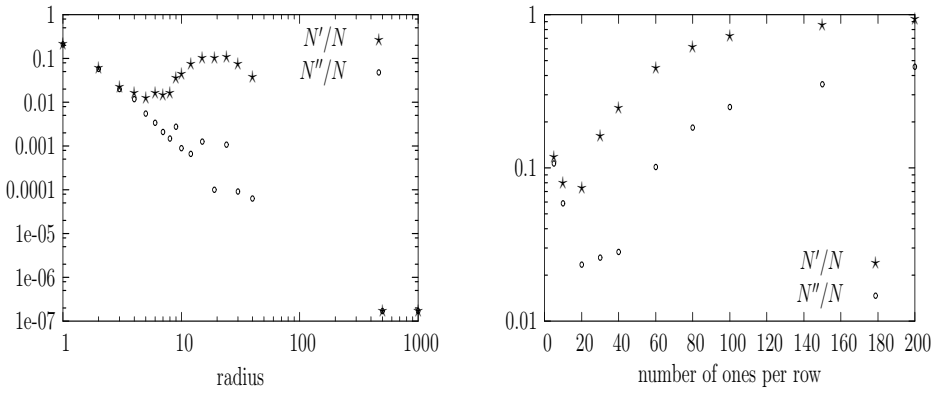


Fig. 4. Reduction ratios for SW-Germany subgraph resp. generated instances.

of Figure 4 shows this effect for a sub-instance of the german railway graph). For generated instances this effect could only be simulated to a certain extent by just increasing the number of ones per row (see right side of Figure 4).

4.3 Time Complexity

As noted before real world instances with existing stops can be reduced very fast, because they decompose into many, small components. For the variants without existing stations, although the reduction rates were always very good, the running time for solving the instances grew quite fast for radii greater or equal to around 10km. Note, however, that radii larger than 10km are irrelevant in our scenario. For moderate radii we observed a quadratic or slightly subquadratic behaviour (in terms of the matrix size) of the running times for reduction and an almost linear running time for solving the set covering problem.

Looking at the generated instances, our experiments suggest a quadratic time complexity for reduction and an almost linear time complexity for solving the set covering problem (Figure 5). Only for the 20% perturbed, unweighted C1P matrices we observed a quadratic behaviour.

For weighted instances advanced reduction also improved the running time for our solver. Figure 3(b) shows this for the case of 10% perturbed, weighted, generated instances. A speedup factor of 100 and more could be observed here. Even the running time for advanced reduction was usually reduced, owing to a the reduced matrix size after the first rounds of reduction.

5 Conclusions

The station location problem was solved almost completely by reduction for our initial real world data. The experimental study confirms the conjecture that one reason for this nice behaviour is the closeness of our instances to C1P. This explains also the results from [7] about the power of data reduction for similar covering problems. Actually it is likely that the data considered there in many cases have (at least almost) C1P.

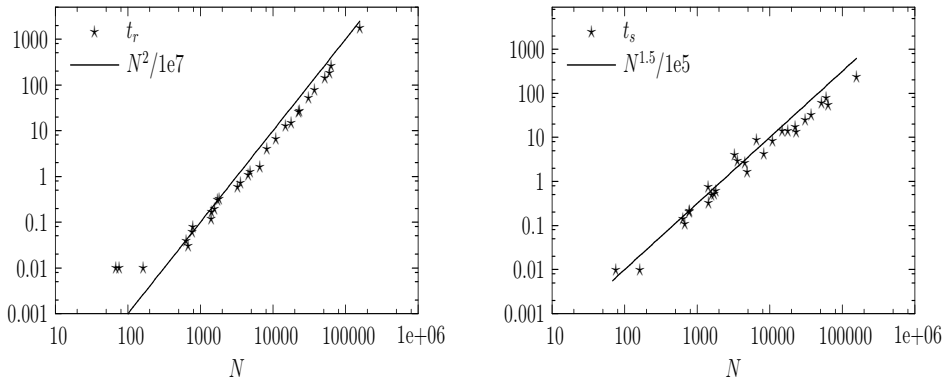


Fig. 5. Reduction time resp. solve time vs. matrix size, weights $\in (0..10)$, 10% perturbation.

References

1. Hamacher, H.W., Liebers, A., Schöbel, A., Wagner, D., Wagner, F.: Locating new stops in a railway network. Electronic Notes in Theoretical Computer Science, Proc. ATMOS 2001 **50** (2001)
2. Schöbel, A., Hamacher, H.W., Liebers, A., Wagner, D.: The continuous stop location problem in public transportation networks. Report in Wirtschaftsmathematik **81** (2002)
3. Caprara, A., Fischetti, M., Toth, P.: Algorithms for the set covering problem. Annals of Operations Research **98** (2000) 353–371
4. Lund, C., Yannakakis, M.: On the hardness of approximating minimization problems. Journal of the ACM **41** (1994) 960–981
5. Kranakis, E., Penna, P., Schlude, K., Taylor, D.S., Widmayer, P.: Improving customer proximity to railway stations. In Petreschi, R., Persiano, G., Silvestri, R., eds.: Proceedings of 5th Italian Conference on Algorithms and Complexity (CIAC 2003). Volume 2653 of Lecture Notes in Computer Science., Springer (2003) 264–276
6. Nemhauser, G., Wolsey, L.: Integer and Combinatorial Optimization. Wiley (1988)
7. Weihe, K.: Covering trains by stations or the power of data reduction. In Battiti, R., Bertossi, A.A., eds.: Proceedings of Algorithms and Experiments (ALEX 98). (1998) 1–8
8. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. Journal of Computer and System Science **13** (1976) 335–379
9. Yellin, D.M., Jutla, C.S.: Finding extremal sets in less than quadratic time. Inform. Process. Lett. **48** (1993) 29–34
10. Pritchard, P.: An old sub-quadratic algorithm for finding extremal sets. Information Processing Letters **62** (1997) 329–334
11. Schöbel, A.: Customer-Oriented Optimization in Public Transportation. PhD thesis, University of Kaiserslautern (2002) Habilitation.
12. Cuthill, E., McKee, J.: Reducing the bandwidth of sparse symmetric matrices. In: Proc. ACM National Conference, Association of Computing Machinery (1969)