Originally published in *Encyclopedia of Algorithms*, pages 832–834. Springer, 2008.

Set Cover with Almost Consecutive Ones Property

2004; Mecke, Wagner

Entry author: Michael Dom*

INDEX TERMS: Covering Set problem, data reduction rules, enumerative algorithm.
SYNONYMS: Hitting Set

PROBLEM DEFINITION

The Set Cover problem has as input a set R of m items, a set C of n subsets of R and a weight function $w : C \to \mathbb{R}$. The task is to choose a subset $C' \subseteq C$ of minimum weight whose union contains all items of R.

The sets R and C can be represented by an $m \times n$ binary matrix A that consists of a row for every item in R and a column for every subset of R in C, where an entry $a_{i,j}$ is 1 iff the ith item in R is part of the jth subset in C. Therefore, the Set Cover problem can be formulated as follows.

> **Input:** An $m \times n$ binary matrix A and a weight function w on the columns of A.
> **Task:** Select some columns of A with minimum weight such that the submatrix A' of A that is induced by these columns has at least one 1 in every row.

While Set Cover is NP-hard in general [4], it can be solved in polynomial time on instances whose columns can be permuted in such a way that in every row the ones appear consecutively, that is, on instances that have the *consecutive ones property* (*C1P*).[1]

Motivated by problems arising from railway optimization, Mecke and Wagner [7] consider the case of Set Cover instances that have "almost the C1P". Having almost the C1P means that the corresponding matrices are similar to matrices that have been generated by starting with a matrix that has the C1P and replacing randomly a certain percentage of the 1's by 0's [7]. For Ruf and Schöbel [8], in contrast, having almost the C1P means that the average number of blocks of consecutive 1's per row is much smaller than the number of columns of the matrix. This entry will also mention some of their results.

*Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany, dom@minet.uni-jena.de.

[1] The C1P can be defined symmetrically for columns; this article focusses on rows. Set Cover on instances with the C1P can be solved in polynomial time, e.g., with a linear programming approach, because the corresponding coefficient matrices are totally unimodular (see [9]).

Originally published in *Encyclopedia of Algorithms*, pages 832–834. Springer, 2008.

Notation. Given an instance (A, w) of Set Cover, let R denote the row set of A and C its column set. A column c_j *covers* a row r_i, denoted by $r_i \in c_j$, if $a_{i,j} = 1$.

A binary matrix has the *strong C1P* if (without any column permutation) the 1's appear consecutively in every row. A *block of consecutive 1's* is a maximal sequence of consecutive 1's in a row. It is possible to determine in linear time if a matrix has the C1P, and if so, to compute a column permutation that yields the strong C1P [2, 3, 6]. However, note that it is NP-hard to permute the columns of a binary matrix such that the number of blocks of consecutive 1's in the resulting matrix is minimized [1, 4, 5].

A *data reduction rule* transforms in polynomial time a given instance I of an optimization problem into an instance I' of the same problem such that $|I'| < |I|$ and the optimal solution for I' has the same value (e.g., weight) as the optimal solution for I. Given a set of data reduction rules, *to reduce* a problem instance means to repeatedly apply the rules until no rule is applicable; the resulting instance is called *reduced*.

KEY RESULTS

Data Reduction Rules. For Set Cover there exist well-known data reduction rules:

Row domination rule: If there are two rows $r_{i_1}, r_{i_2} \in R$ with $\forall c \in C : r_{i_1} \in c$ implies $r_{i_2} \in c$, then r_{i_2} is *dominated* by r_{i_1}. Remove row r_{i_2} from A.

Column domination rule: If there are two columns $c_{j_1}, c_{j_2} \in C$ with $w(c_{j_1}) \geq w(c_{j_2})$ and $\forall r \in R : r \in c_{j_1}$ implies $r \in c_{j_2}$, then c_{j_1} is *dominated* by c_{j_2}. Remove c_{j_1} from A.

In addition to these two rules, a column $c_{j_1} \in C$ can also be dominated by a subset $C' \subseteq C$ of the columns instead of a single column: If there is a subset $C' \subseteq C$ with $w(c_{j_1}) \geq \sum_{c \in C'} w(c)$ and $\forall r \in R : r \in c_{j_1}$ implies $(\exists c \in C' : r \in c)$, then remove c_{j_1} from A. Unfortunately, it is NP-hard to find a dominating subset C' for a given set c_{j_1}. Mecke and Wagner [7], therefore, present a restricted variant of this generalized column domination rule.

For every row $r \in R$, let $c_{\min}(r)$ be a column in C that covers r and has minimum weight under this property. For two columns $c_{j_1}, c_{j_2} \in C$, define $X(c_{j_1}, c_{j_2}) := \{c_{\min}(r) \mid r \in c_{j_1} \wedge r \notin c_{j_2}\}$. The new data reduction rule then reads as follows.

Advanced column domination rule: If there are two columns $c_{j_1}, c_{j_2} \in C$ and a row that is covered by both c_{j_1} and c_{j_2}, and if $w(c_{j_1}) \geq w(c_{j_2}) + \sum_{c \in X(c_{j_1}, c_{j_2})} w(c)$, then c_{j_1} is *dominated* by $\{c_{j_2}\} \cup X(c_{j_1}, c_{j_2})$. Remove c_{j_1} from A.

Theorem 0.1 ([7]). *A matrix A can be reduced in $O(Nn)$ time with respect to the column domination rule, in $O(Nm)$ time with respect to the row domination rule, and in $O(Nmn)$ time with respect to all three data reduction rules described above, when N is the number of 1's in A.*

In the databases used by Ruf and Schöbel [8], matrices are represented by the column indices of the first and last 1's of its blocks of consecutive 1's. For such matrix representations, a fast data reduction rule is presented [8], which eliminates "unnecessary" columns and which, in the implementations, replaces

Originally published in *Encyclopedia of Algorithms*, pages 832–834. Springer, 2008.

the column domination rule. The new rule is faster than the column domination rule (a matrix can be reduced in $O(mn)$ time with respect to the new rule), but not as powerful: Reducing a matrix A with the new rule can result in a matrix that has more columns than the matrix resulting from reducing A with the column domination rule.

Algorithms. Mecke and Wagner [7] present an algorithm that solves SET COVER by enumerating all feasible solutions.

Given a row r_i of A, a *partial solution for the rows* $r_1, \ldots, r_i$ is a subset $C' \subseteq C$ of the columns of A such that for each row r_j with $j \in \{1, \ldots, i\}$ there is a column in C' that covers row r_j.

The main idea of the algorithm is to find an optimal solution by iterating over the rows of A and updating in every step a data structure S that keeps *all* partial solutions for the rows considered so far. More exactly, in every iteration step the algorithm considers the first row of A and updates the data structure S accordingly. Thereafter, the first row of A is deleted. The following code shows the algorithm.

```
1 Repeat m times: {
2     for every partial solution C' in S that does not cover the first row of A: {
3         for every column c of A that covers the first row of A: {
4             Add {c} ∪ C' to S; }
5         Delete C' from S; }
6     Delete the first row of A; }
```

This straightforward enumerative algorithm could create a set S of exponential size. Therefore, the data reduction rules presented above are used to delete after each iteration step partial solutions that are not needed any more. To this end, a matrix B is associated with the set S, where every row corresponds to a row of A and every column corresponds to a partial solution in S—an entry $b_{i,j}$ of B is 1 iff the jth partial solution of B contains a column of A that covers the row r_i. The algorithm uses the matrix $C := \left(\begin{array}{c|c} A & B \\ \hline 0 \ldots 0 & 1 \ldots 1 \end{array}\right)$, which is updated together with S in every iteration step.[2] Line 6 of the code shown above is replaced by the following two lines:

```
6     Delete the first row of the matrix C;
7     Reduce the matrix C and update S accordingly; }
```

At the end of the algorithm, S contains exactly one solution, and this solution is optimal. Moreover, if the SET COVER instance is nicely structured, the algorithm has polynomial running time:

Theorem 0.2 ([7]). *If A has the strong C1P, is reduced, and its rows are sorted in lexicographic order, then the algorithm has a running time of $O(M^3 n)$ where M is the maximum number of 1's per row and per column.*

Theorem 0.3 ([7]). *If the distance between the first and the last 1 in every column is at most k, then at any time throughout the algorithm the number of columns in the matrix B is $O(2^k n)$, and the running time is $O(2^{2k} kmn^2)$.*

[2] The last row of C allows to distinguish the columns belonging to A from those belonging to B.

Ruf and Schöbel [8] present a branch and bound algorithm for Set Cover instances that have a small average number of blocks of consecutive 1's per row.

The algorithm considers in each step a row r_i of the current matrix (which has been reduced with data reduction rules before) and branches into bl_i cases, where bl_i is the number of blocks of consecutive 1's in r_i. In each case, one block of consecutive 1's in row r_i is selected, and the 1's of all other blocks in this row are replaced by 0's. Thereafter, a lower and an upper bound on the weight of the solution for each resulting instance is computed. If a lower bound differs by a factor of more than $1 + \epsilon$, for a given constant ϵ, from the best upper bound achieved so far, the corresponding instance is subjected to further branchings. Finally, the best upper bound that was found is returned.

In each branching step, the bl_i instances that are newly generated are "closer" to have the (strong) C1P than the instance from which they descend. If an instance has the C1P, the lower and upper bound can easily be computed by exactly solving the problem. Otherwise, standard heuristics are used.

APPLICATIONS

Set Cover instances occur e.g. in railway optimization, where the task is to determine where new railway stations should be built. Each row then corresponds to an existing settlement, and each column corresponds to a location on the existing trackage where a railway station could be build. A column c covers a row r, if the settlement corresponding to r lies within a given radius around the location corresponding to c.

If the railway network consisted of one straight line rail track only, the corresponding Set Cover instance would have the C1P; instances arising from real world data are close to have the C1P [7, 8].

EXPERIMENTAL RESULTS

Mecke and Wagner [7] make experiments on real-world instances as described in the Applications section and on instances that have been generated by starting with a matrix that has the C1P and replacing randomly a certain percentage of the 1's by 0's. The real-world data consists of a railway graph with 8200 nodes and 8700 edges, and 30000 settlements. The generated instances consist of 50–50000 rows with 10–200 1's per row. Up to 20% of the 1's are replaced by 0's.

In the real-world instances, the data reduction rules decrease the number of 1's to between 1% and 25% of the original number of 1's without and to between 0.2% and 2.5% with the advanced column reduction rule. In the case of generated instances that have the C1P, the number of 1's is decreased to about 2% without and to 0.5% with the advanced column reduction rule. In instances with 20% perturbation, the number of 1's is decreased to 67% without and to 20% with the advanced column reduction rule.

The enumerative algorithm has a running time that is almost linear for real-world instances and most generated instances. Only in the case of generated instances with 20% perturbation, the running time is quadratic.

Ruf and Schöbel [8] consider three instance types: real-world instances, instances arising from Steiner triple systems, and randomly generated instances. The latter have a size of 100×100 and contain either 1–5 blocks of consecutive 1's in each row, each one consisting of between one and nine 1's, or they are generated with a probability of 3% or 5% for any entry to be 1.

The data reduction rules used by Ruf and Schöbel turn out to be powerful for the real-world instances (reducing the matrix size from about 1100×3100 to 100×800 in average), whereas for all other instance types the sizes could not be reduced noticeably.

The branch and bound algorithm could solve almost all real-world instances up to optimality within a time of less than a second up to one hour. In all cases where an optimal solution has been found, the first generated subproblem had already provided a lower bound equal to the weight of the optimal solution.

CROSS REFERENCES

Greedy Set-Cover Algorithm (entry 00159)

RECOMMENDED READING

[1] J. E. Atkins and M. Middendorf. On physical mapping and the consecutive ones property for sparse matrices. *Discrete Appl. Math.*, 71(1–3):23–40, 1996.

[2] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. System Sci.*, 13:335–379, 1976.

[3] D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific J. Math.*, 15(3):835–855, 1965.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[5] P. W. Goldberg, M. C. Golumbic, H. Kaplan, and R. Shamir. Four strikes against physical mapping of DNA. *J. Comput. Biol.*, 2(1):139–152, 1995.

[6] W.-L. Hsu and R. M. McConnell. PC trees and circular-ones arrangements. *Theor. Comput. Sci.*, 296(1):99–116, 2003.

[7] S. Mecke and D. Wagner. Solving geometric covering problems by data reduction. In *Proceedings of the 12th Annual European Symposium on Algorithms (ESA '04)*, volume 3221 of *LNCS*, pages 760–771. Springer, 2004.

[8] N. Ruf and A. Schöbel. Set covering with almost consecutive ones property. *Discrete Optim.*, 1(2):215–228, 2004.

[9] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.