An Exact Algorithm for Unicost Set Covering

Emir Demirović[1], Théo Le Calvar[3], Nysret Musliu[1], and Katsumi Inoue[2]

[1] Technische Universität Wien, Austria
Database and Artificial Intelliegence Group
`demirovic` ∨ `musliu@dbai.tuwien.ac.at`
[2] National Institute of Informatics, Japan
The Graduate University for Advanced Studies
`inoue@nii.ac.jp`
[3] University of Angers, France
`theo.lecalvar@etud.univ-angers.fr`

Abstract. In this work we present ongoing work on the development of an exact algorithm which exploits heuristic solvers for the Unicost Set Covering problem. We analyze and discuss various algorithmic properties of an exact set covering algorithm, provide a new heuristic algorithm for solving the dual problem to compute lower bounds, show how computing lower and upper bounds can influence branching decisions, provide several propagation rules to deduce variable assignments or signal backtracking, several other lower bounding techniques, and finally propose an algorithm based on those findings.

Keywords: set covering, unicost, optimization, set packing, tree search

1 Introduction

Set Covering is a well known NP-complete problem. Given a finite set X and a family F of subsets of X, it asks whether there exists a subset $s \subseteq X$ with cardinality k such that every element from F contains at least one element from s. Its NP-hard optimization variant asks for the minimum number of elements to cover X. It is common to associate weights with every element of X and ask a similar question, but taking the sum of weights of selected elements rather than just the cardinality of X. In this work we consider the Unicost Set Covering problem where all weights are equal, as the unicost case appears to be more challenging to solve than its weighted version [10]. We note that there are other equivalent definitions of set covering.

There are many applications of set covering, such as scheduling, construction of optimal logical circuits, crew scheduling in railways [3], urban waste management [1], etc. In [6] the author discusses its use within hypertree decomposition. Set covering is also used in order to provide users the most representative solutions for a multi-objective optimization problem [8]. Team Formation [7] has a direct link to set covering. Additional applications can be found in [9].

Benchmark instances for set covering are available at the OR library [2] and are used in many works in the literature (e.g. [6] [5]). Both weighted and unicost set covering instances can be found, and additional unicost instances can be obtained by setting weights to one in the weighted versions. While general purpose mixed integer programming tools like *cplex* can solve weighed instances efficiently (in some cases within seconds, see [4]), the unicost ones are significantly harder (see [10]). In contrast, the heuristic solver from [5] [6] can solve these unicost instances very well (in some cases to optimality within seconds), but cannot prove optimality. Even though unicost set covering can be formulated very naturally as maxSAT, maxSAT solvers have difficulties to solve unicost set covering, as shown in Section 4. We believe this is because the main problem is minimizing the number of sets used rather than constraint satisfaction, and CDCL and other maxSAT techniques do not offer much help in that regard for set covering (both upper and lower bounds obtained are not satisfactory). All this motivated us to investigate whether it would be possible to devise an exact algorithm for unicost set covering which would exploit the advantages offered by heuristic solvers, in an attempt to offer best of both worlds: ability to prove optimality, while being able to calculate the solution fast. The development of this algorithm can also lead to better understanding of how to incorporate heuristics in exact solvers, possibly allowing us to translate our techniques to other problems. As weighted set covering can be solved efficiently with off-the-shelf tools, we choose to focus on the unicost case. It is important to note that there are no exact algorithms in the literature for the Unicost Set Covering problem other than the use of general mixed integer programming tools, to the best of our knowledge.

Exact algorithms exhaustively go through the search space. We consider such an algorithm in the form of a tree search algorithm based on branch and bound. In these algorithms, an unassigned variable is chosen (*branching variable*) and is assigned a value. Then, estimates on the lower and upper bound of the solution are computed and based on these estimates it is decided whether the algorithm should backtrack or proceed with the current variable assignment. In addition, propagation rules can be applied in order to deduce assignments of unassigned variables. Variable assignments in our case correspond to deciding which sets are in or out of the optimum solution (the subset $s \subseteq X$). This procedure is repeated until all possible variable assignments have been considered and at this point either the optimum has been found or it has been proven that no feasible solution exists. There are several important parts of the algorithm: which variable to *branch* on, what value to assign it, how to compute the lower and upper bounds, and how to infer other variable assignments. We examine and address each of these points in our work.

The main contributions of this work are the following:

– We identify a good property that branching strategies should adhere to in general for set covering. This way we are able to limit the pool of potential sets for branching as well as detect whether during the search the problem can be split into two or more independent parts (*decomposition detection*),

without referring to a specific branching heuristic. We also propose a concrete branching strategy.
- We develop a new heuristic algorithm for solving the dual problem of unicost set covering: unicost set packing. Solving the dual problem allows us to compute a lower bound. Based on our experiments, our algorithm is able to compute optimal solutions for unicost set packing faster than cplex.
- We show how variable branching influences lower and upper bounds given certain branching decisions. We can exploit this information to ensure that the bounds will be nonincreasing, nondecreasing, or strictly increasing, depending on which conditions are met. This is useful because if we manage to maintain a nonincreasing upper bound while having a strictly increasing lower bound, it could resulting in much faster pruning of the search space.
- Several propagation rules and lower bounding techniques are proposed.
- We experimentally compare maxSAT, IP, and dedicated heuristic solvers for unicost set covering. The results show that heuristics perform very well and that unicost set covering instances are very difficult for maxSAT solvers.

The rest of the paper is organized as follows. In Section 2 we formally define the Set Covering problem and introduce notions used in the paper. We then proceed our main contributions in Section 3 where we describe our algorithms. In Section 4 we show the effectiveness of our heuristic for the dual problem and compare the performance of maxSAT, IP, and dedicated heuristic solvers for set covering. Conclusions are given in Section 5.

2 Problem Description and Notions

The Set Covering problem is formally defined as follows:

$$minimize \sum_{i \in X} w_i * s_i \tag{1}$$

$$\sum_{i \in X} a_{(i,j)} * s_i \geq 1 \qquad \forall j \in Y \tag{2}$$

$$s_i \in \{0,1\} \qquad \forall i \in X \tag{3}$$

The binary constants $a_{(i,j)}$ define whether the *i-th* set can cover the *j-th* element. A solution S is defined as a variable assignment for every set s_i. The *i-th* set is *in* the solution if $s_i = 1$, *out* of the solution if $s_i = 0$, and undecided if no value is assigned. A (partial) solution S is said to be $infeasible$ if not all constraints defined in Equation 2 are satisfied, otherwise it is feasible. Every partial feasible solution can be completed by assigning $s_i = 0$ to all undecided sets. We define $\alpha(i) = \{j : a_{(i,e)} = 1\}$ as the set of elements that the *i-th* set can cover. Two sets i and j are *connected* if $\alpha(i) \cap \alpha(j) \neq \emptyset$. An element e is said to be *uncovered* in the infeasible solution S if no set in S can cover it. Equation 4 is called the cost function. For the Unicost Set Covering problem which we consider in this work, all weights are equal to one ($w_i = 1$) and the cost function of a solution S is simply the number of sets in the solution.

3 Algorithms

In this section we describe the main contributions of the paper. We present our heuristic algorithm for solving the dual problem in order to provide a lower bound, a good general property of branching strategies for unicost set covering, the impact of branching on future lower and upper bounds, as well as some propagation rules and additional ways of calculating lower bounds.

3.1 Dual Problem - Lower Bound

We first define the dual problem, present unicost set packing as a dual for unicost set covering, and then provide an algorithm to compute it.

Let f and g be two scalar functions. We say that $max(g)$ is the dual problem of $min(f)$ iff $max(g) \leq min(f)$. Therefore, any feasible solution for g is a lower bound for $min(f)$. The dual problem for unicost set covering is unicost set packing and is defined as follows:

$$maximize \sum_{j \in Y} e_j \tag{4}$$

$$\sum_{j \in Y} a_{(i,j)} * e_j \leq 1 \qquad \forall i \in X \tag{5}$$

$$e_j \in \{0,1\} \qquad \forall j \in Y \tag{6}$$

In other words, given a unicost set covering problem, its corresponding unicost set packing problem asks for the maximum number of elements that have no common sets.

We developed a local search algorithm to solve unicost set packing inspired by set covering algorithms from [6] and [5]. Analogous to set covering, a solution for set packing E is defined as a variable assignment for every element e_j. The *j-th* element is *in* the solution if $e_j = 1$, *out* of the solution if $e_j = 0$, and undecided if no value is assigned. Before describing the algorithm, we first define the cost and infeasibility score of a solution.

Definition 1 (Cost and Infeasibility Score for Unicost Set Packing). *Let E be a solution for set packing. The cost of E is the number of elements in E (noted as $|E|$) and the infeasibility score is defined as score(E) =* $\sum_{i \in X}(max((\sum_{j \in Y} a_{(i,j)} * e_j) - 1, 0))$.

Every feasible solution has an infeasibility cost equal to zero. The infeasibility cost grows with every constraint violation. Initially all elements are in the current solution E_{cur}. In each iteration, an element is either removed or added. If $|E_{cur}| \geq |E_{best}| - 1$, then an element is removed from E_{cur}. Otherwise, an element is added. This is done to focus the search around solutions which cost one less then the best solution found so far. Each element is assigned a cost equal to the amount the infeasibility score will increase by its addition or removal.

During element selection, the one with the minimum cost is selected. In case of ties, elements which have not been used recently are preferred. To prevent getting stuck in a local optimum, a tabu mechanism is used, meaning that element that have been added or removed in the past m iterations are ignored during the selection phase. Despite its simple nature, the algorithm is able to compute optimal values very fast and performs well compared to *cplex*, a general purpose mixed integer programming software, as discussed in Section 4.

3.2 Variable Branching

Variable branching is an important aspect of tree search algorithms. Here we discuss a desirable general property and propose a concrete branching strategy.

We say a problem is decomposable if Y from Equation 2 can be partitioned into Y_1 and Y_2 such that $\forall i \in X(\exists j_1 \in Y_1\ a_{(i,j_1)} = 1 \Leftrightarrow \forall j_2 \in Y_2\ a_{(i,j_2)} = 0)$. In other words, the problem can be split into two independent problems which do not share any sets or elements and the objective value of the problem is the sum of the objective values of the two parts. During the search, sets are assigned values $s = 0$ and it could be the case that the resulting subproblem becomes decomposable. If this happens, a suboptimal solution for one part implies a suboptimal solution for the whole problem. Therefore, it is important to detect whether the problem is decomposable. To elaborate further, we introduce the concept of a *decision level*. The decision level at a particular algorithm iteration is the number of sets which have been assigned values through a decision or backtracking, but not as a result of propagation.

Proposition 1 (Set Restriction and Decomposition Detection). *When an element gets covered or uncovered at decision level i, assign it a label i. In the branching phase, consider only undecided sets which can cover elements with the smallest label, regardless of whether the elements need to be covered or not. If no set can be selected, the current subproblem is decomposable.*

The above proposition is useful regardless of how branching decisions are made, as it only limits which sets should be considered as potential candidates for branching and detects whether a decomposition is possible. This is different from forward checking because sets not considered in one iteration might be considered in future ones when all labeled elements get covered.

We now briefly discuss a concrete branching strategy. We defined the most constrained elements to be the elements which can be covered by the least amount of sets. We propose selecting a set from one of the most constrained elements which also covers the least amount of other elements. By selecting a set in the most constrained element, we are likely to select a set which is in the optimal solution, and by covering the least we limit future set selection through Proposition 1.

3.3 Variable Branching Influence on Lower and Upper Bounds

In Section 3.1 we discussed how to compute a lower bound solution by using the dual problem. For the upper bound, we use the state-of-the-art algorithm proposd in [5]. In this section we analyze the impact of branching decisions on the bounds of the problem. Let S and E be the sets or elements which are in the solutions for upper and lower bounds, respectively.

Proposition 2 (Nonincreasing Upper Bounds).

– *By assigning $s = 1$ for $s \in S$ or $s = 0$ for $s \notin S$, the new upper bound is at most $|S|$.*

Proposition 3 ((Non)Decreasing Lower Bound).

– *By assigning $s = 0$ to any set, the new lower bound is at least $|E|$.*
– *By assigning $s = 1$ such that k elements in the solution E get covered, the new lower bound is at least $|E| - k + 1$.*

Proposition 4 (Strictly Increasing Lower Bound).

– *By assigning $s = 1$ such that no element $e \in E$ gets covered, the lower bound is at least $|E| + 1$.*

These are important because they provide guarantees on the behavior of the upper and lower bounds based on branching. If we are able to branch on sets which guarantee nonincreasing upper bound and strictly increasing lower bound, we can expect earlier prunings of the search, helping the algorithm to detect areas of the search space which are of no use.

3.4 Propagation and Additional Lower Bounds

As variable assignments are made during the search, it might be the case that certain sets become redundant. These sets should be removed from further consideration. This is formalized in the following two propositions.

Proposition 5 (Propagation - Set Subsumption). *In every optimal solution we have: $\alpha(s_i) \supseteq \alpha(s_j) \wedge i < j \Rightarrow s_j = 0$.*

Proposition 6 (Propagation - Set Redundancy). *If a set s does not cover any uncovered element ($\alpha(s) = \emptyset$), or its removal from the solution does not uncover any elements, it cannot be part of the optimal solution ($s = 0$).*

Let $coverage(s)$ be the number of uncovered elements which would get covered if s was inserted into the solution, $c_{max} = max\{coverage(s) : i \notin S\}$, m be the number of uncovered elements, S_{cur} be the current (partial) solution, S_{best} be the best solution found so far, and $r = \lceil \frac{m}{c_{max}} \rceil$. We can then make the following statements.

Proposition 7 (Optimistic Lower Bound, Value). *A lower bound can be calculated by: $LB_{optimistic} = |S_{cur}| + r$.*

Proposition 8 (Optimistic Lower Bound, Backtrack). *$|S_{cur}|$ cannot be completed to a solution better than $|S_{best}|$ if $r > (|S_{best}| - |S_{cur}| - 1)$.*

A better estimate than Proposition 7 and 8 can be done, although it is not as simple to calculate. Let C_{max} be the sorted array of set coverages in descending order, $p(i) = \sum_{1 \leq k < i} C_{max}(k)$ be the *i-th* partial sum of C_{max}, and $i_{min} = min\{i : p(i) \geq m\}$.

Proposition 9 (Optimistic Lower Bound, Refined Value). *A refined lower bound can be calculated by:* $LB_{refined} = |S| + i_{min}$.

Proposition 10 (Optimistic Lower Bound, Refined Backtrack). $|S_{cur}|$ *cannot be completed to a solution better than* $|S_{best}|$ *if* $i_{min} > (|S_{best}| - |S_{cur}| - 1)$.

3.5 Main Algorithm Outline

The previous subsections are now combined and summarized. Initially $dl = 0$, $S_{cur} = \emptyset$, $S_{best} = X$, and every element is unlabeled. Our proposed algorithm is as follows.

1. Compute upper and lower bounds based on [5] and Section 3.1.
2. Compute candidate sets S_{cand} as the unassigned sets that can cover at least one element with the lowest label.
   (a) If $S_{cand} \neq \emptyset$, select a set $s \in S_{cand}$ according to a heuristic and insert it into the (partial) solution (assigning $s = 1$), taking into account the influence of s based on Section 3.3.
   (b) Else, check if the problem is decomposable into two or more parts.
      i. If not, select a unassigned set $s \in X$ according to a heuristic.
      ii. Else, solve every decomposable part independently with this algorithm and add their assignments to S_{cur}, and go to Step 5.
3. Assign label dl to all unlabeled elements which s covers.
4. Apply propagation rules (Section 3.4).
5. If S_{cur} satisfies Equation 2 and $|S_{cur}| < |S_{best}|$, then assign $|S_{best}| = |S_{cur}|$.
6. Determine if backtracking is needed by checking if any of the following holds:
   (a) The upper bound is equal to one of the lower bounds from Section 3.1 and Section 3.4.
   (b) Proposition 6.
   (c) $|S_{cur}| \geq |S_{best}| - 1$.
7. When backtracking, let s_k be the last nonpropagated assignment:
   (a) If $s_k = 1$, assign $s_k = 0$ and go to Step 4.
   (b) Else, unassign s_k and propagation assignments cause by it, remove the labels from all uncovered elements which s_k can cover, $dl = dl - 1$, and go to Step 1.
8. $dl = dl + 1$ and go to Step 1.

4 Experiments

We experimentally compared our heuristic algorithm for unicost set packing (Section 3.1) with an IP approach, and compared IP, maxSAT, and an existing heuristic approach [5] for unicost set covering. We considered 45 unicost set covering instances from the packages scp4x, scp5x, scp6x, and scpnrx, taken from the OR library [2]. We used *cplex* for IP, several maxSAT solvers (ahmaxsat1.55, optiriss, WPM3-2015-in, and open-wbo), the heuristic algorithm for unicost set covering from [5], and our implementation for heuristic unicost set packing (Section 3.1). The computational time was restricted to 10 minutes.

Due to space limitations we briefly comment on the results overall rather than providing a table with detailed results. For unicost set packing, both our heuristic and *cplex* manage to compute the optimal solutions. Our heuristic does so within a second, while *cplex* takes around 40 seconds and an average of 90 seconds to prove that it is optimal. In this case there is a clear tradeoff: speed versus ability to prove optimality. However, the difference in speed is significant. For unicost set covering, the heuristic algorithm from [5] is better than *cplex* on 31 out of 45 instances and equal on the rest, averaging a solution of 31.8 compared to 32.7. In the allocated time limit *cplex* did not manage to prove optimality for any instance. It is important to note that *cplex*'s result were always obtained with the heuristic approach faster and neither approach made use of the complete computational time: the heuristic algorithm computes its best solution on average in 45 seconds, while *cplex* does on average in 30 seconds. The maxSAT solvers struggled to provide results and were always worse than the other two approaches. Overall, the effectiveness of heuristic approaches motivated the development of our exact algorithm.

5 Conclusion

In this work we presented ongoing work on the development of an exact algorithm which exploits heuristic solvers for the Unicost Set Covering problem. We showed a good general branching property and a concrete branching strategy, a new heuristic algorithm for computing the solution for the dual problem (unicost set packing), showed how branching influences lower and upper bounds given certain conditions, a number of propagation rules to deduce variable assignments or signal backtracking, and other lower bounding techniques. We believe the ideas presented are promising for solving the Unicost Set Covering problem. Additionally, we compared maxSAT, IP, and heuristic solvers for set covering and revealed a big discrepancy between maxSAT solvers and other approaches. Given that there are many applications for unicost set covering, the development of this algorithm is important, especially since there are no other exact algorithms in the literature other than using general optimization tools.

Acknowledgments. The work was supported by the Austrian Science Fund (FWF): P24814-N23, the Vienna PhD School of Informatics, and the National Institute of Informatics in Tokyo, Japan.

References

1. Bautista, J., and Pereira, J. Modeling the problem of locating collection areas for urban waste management. an application to the metropolitan area of barcelona. *Omega 34*, 6 (2006), 617–629.
2. Beasley, J. E. OR-Library: distributing test problems by electronic mail. *Journal of the operational research society* (1990), 1069–1072.
3. Caprara, A., Fischetti, M., Toth, P., Vigo, D., and Guida, P. L. Algorithms for railway crew management. *Math. Program. 79* (1997), 125–141.
4. Caprara, A., Toth, P., and Fischetti, M. Algorithms for the set covering problem. *Annals OR 98*, 1-4 (2000), 353–371.
5. Gao, C., Yao, X., Weise, T., and Li, J. An efficient local search heuristic with row weighting for the unicost set covering problem. *European Journal of Operational Research 246*, 3 (2015), 750–761.
6. Musliu, N. Local search algorithm for unicost set covering problem. In *Advances in Applied Artificial Intelligence, 19th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2006, Annecy, France, June 27-30, 2006, Proceedings* (2006), M. Ali and R. Dapoigny, Eds., vol. 4031 of *Lecture Notes in Computer Science*, Springer, pp. 302–311.
7. Okimoto, T., Schwind, N., Clement, M., Ribeiro, T., Inoue, K., and Marquis, P. How to Form a Task-Oriented Robust Team. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015* (2015), G. Weiss, P. Yolum, R. H. Bordini, and E. Elkind, Eds., ACM, pp. 395–403.
8. Schwind, N., Okimoto, T., Clement, M., and Inoue, K. Representative Solutions for Multi-Objective Constraint Optimization Problems. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.* (2016), C. Baral, J. P. Delgrande, and F. Wolter, Eds., AAAI Press, pp. 601–604.
9. Vemuganti, R. Applications of Set Covering, Set Packing and Set Partitioning Models: A Survey. In *Handbook of combinatorial optimization*. Springer, 1998, pp. 573–746.
10. Yelbay, B., Birbil, Ş. İ., and Bülbül, K. The set covering problem revisited: an empirical study of the value of dual information. *European Journal of Operational Research* (2012).