PACE Solver Description: Reductions and Heuristic Search for the Dominating Set Problem and the Hitting Set Problem

Florian Fontan
Independent researcher, Grenoble, France

Guillaume Verger
Independent researcher, Lyon, France

Abstract

In this paper, we describe the solver we submitted for both heuristic tracks of the PACE challenge 2025 on the dominating set problem and the hitting set problem. We solve both problems as unicost set covering problems. Our solver first performs reductions on the instance. Then greedy algorithms generate an initial solution that serves as starting point of the large neighborhood search and the local search which are executed afterwards. The solver ranked first in the dominating set heuristic track, and second in the hitting set heuristic track.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization

Keywords and phrases dominating set, hitting set, unicost set covering, reductions, large neighborhood search, local search

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.36

1 Outline

Both dominating set and hitting set instances are solved as unicost set covering instances.

The solving process follows four successive phases:

1. Instance reduction
2. Initial solution generated by greedy algorithms
3. Large neighborhood search
4. Local search

2 Instance reduction

The goal of this phase is to reduce the size of the instance to be solved.

The reductions applied are:

- Mandatory sets removal
- Dominated sets removal
- Dominated elements removal
- Set folding
- Unconfined sets reduction
- Small components reduction

The first three reduction rules are classical reductions for set covering problems. They correspond to Reduction 1 to 5 from [4].

The set folding reduction rule corresponds to Reduction 7 from [4] and is a generalization of the vertex folding reduction rule for the vertex cover problem. The idea is to replace a set j that covers two elements, together with its two neighboring sets j_1 and j_2, by a

single set j'. If j' is included in the solution of the reduced problem, then j_1 and j_2 must be included in the solution of the original problem; otherwise, j must be included in the solution of the original problem.

The unconfined reduction rule is a generalization of the unconfined reduction rule for the vertex cover problem [5]. The idea is to prob the inclusion of a set into an optimal solution and show that another optimal solution without this set exists. If this is the case, then the set can be removed from the instance.

For the small component reduction, we compute simple lower and upper bounds for each connected component of the instance. Any component for which the two bounds coincide can then be removed from the instance.

We also implemented generalizations of the twin reduction rule [5] and the crown reduction rule [3], but they did not have any effect on the challenge instances. Therefore, we did not enable them in our solver.

3 Initial solution

The large neighborhood search and the local search algorithms require an initial solution. Our initial solution generation procedure relies on 4 greedy algorithms.

- The classical greedy algorithm for the set covering problem: while not all elements are covered, the set covering the highest number of uncovered elements is added to the solution.
- The greedy algorithm from [2]: this greedy algorithm is similar to the classical greedy, but uses a different set scoring function.
- The dual greedy algorithm: in this greedy algorithm, while not all elements are covered, an uncovered element is selected, and a set covering it is added to the solution.
- The reverse greedy algorithm: this greedy algorithm starts with a solution containing all sets. Redundant sets are removed until none remain.

The first two greedy algorithms work well when the number of sets in the solution is small compared to the total number of sets. On the other hand, the reverse greedy algorithm works well when most sets are in the solution.

To determine which greedy algorithm to run, we assess the number of sets in the solution by running the dual greedy algorithm. The dual greedy algorithm is always very fast, although in general it yields worse solutions than the other greedy algorithms.

4 Large neighborhood search

The goal of this step is to quickly improve the solution obtained in the previous phase.

To this end, we apply a large neighborhood search algorithm. Each element is assigned a weight, initially weight of 1. The score of a set is then defined as the sum of the weights of the uncovered elements it covers.

The large neighborhood search works as follows. At each iteration:

- The set with the worst score is removed from the solution
- The score of the uncovered elements is incremented
- While not all elements are covered, the set with the best score is added to the solution and while the solution contains redundant sets, a redundant set is removed.

5 Local search

The goal of this last phase is to improve the current solution to get the best possible solution.

It implements a local search with a row weighting scheme similar to the one proposed by [2]. If the best feasible solution found so far contains k sets, the local search works on a solution containing $k-1$ sets. As a result, some elements remain uncovered. A weight is associated to each element. The penalty of a solution is defined as the sum of the weights of its uncovered elements. The goal of the local search is to find a solution with penalty 0. At each iteration, it performs a swap of minimal penalty between a set outside the solution that covers an uncovered element, and one of its set neighbor, that is, a set such that there exists an element that both sets cover. Thus, the solution still contains $k-1$ sets. Then the weight of each newly uncovered element is incremented.

When the number of set neighbors is too high so that computing them makes the solver hit the memory limit, it runs the local search with row weighting from [1] instead. Instead of looking for a swap, at each iteration, this local search first removes a set from the solution, and then adds another one that covers an uncovered element.

References

1 Chao Gao, Xin Yao, Thomas Weise, and Jinlong Li. An efficient local search heuristic with row weighting for the unicost set covering problem. *European Journal of Operational Research*, 246(3):750–761, November 2015. doi:10.1016/j.ejor.2015.05.038.

2 Weibo Lin, Fuda Ma, Zhouxing Su, Qingyun Zhang, Chumin Li, and Zhipeng Lü. Weighting-based parallel local search for optimal camera placement and unicost set covering. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, GECCO '20, pages 3–4, Cancún, Mexico, July 2020. Association for Computing Machinery. doi:10.1145/3377929.3398184.

3 William Suters, III. Crown Reductions and Decompositions: Theoretical Results and Practical Methods. *Masters Theses*, December 2004.

4 Johan M. M. van Rooij and Hans L. Bodlaender. Exact algorithms for dominating set. *Discrete Applied Mathematics*, 159(17):2147–2164, October 2011. doi:10.1016/j.dam.2011.07.001.

5 Mingyu Xiao and Hiroshi Nagamochi. Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs. *Theoretical Computer Science*, 469:92–104, January 2013. doi:10.1016/j.tcs.2012.09.022.