

Shadoks Approach to Minimum Hitting Set and Dominating Set

Guilherme D. da Fonseca ✉ 

LIS, Aix-Marseille Université

Fabien Feschet ✉ 

LIMOS, Université Clermont Auvergne

Yan Gerard ✉ 

LIMOS, Université Clermont Auvergne

1 Introduction

We describe the solvers used by the Shadoks team in the PACE 2025 challenge. The challenge considers solvers for the minimum dominating set and hitting set problems. Since it is easy to reduce minimum dominating set to minimum hitting set, we only consider the latter. We present both a heuristic and an exact algorithm for the problem.

Given a hypergraph $H = (V, \mathcal{E})$ with vertices V and hyperedges $\mathcal{E}$, a *hitting set* is a subset $S \subseteq V$ such that for all $E \in \mathcal{E}$ we have $S \cap E \neq \emptyset$. The *minimum hitting set* is the hitting set with the smallest number of elements. In Section 2, we describe the different building blocks of our solvers. In Section 3, we show how the heuristic and exact solvers put these building blocks together.

2 Tools and Techniques

Reductions Rules: There are three key reduction rules [5] that we apply. (i) If a hyperedge E is a superset of another hyperedge, then remove E . (ii) If a vertex v hits a subset of another vertex, then remove v . (iii) If a hyperedge has a single vertex v , then add v to the solution and remove all hyperedges that contain v . For the large heuristic instances, we had to make the implementation of these reductions very efficient.

Greedy Heuristic: Greedy heuristics insert the vertex of maximum “value” in the solution at each step. The classic greedy heuristic sets the value of a vertex as the number of unhit hyperedges that it would hit. Since large hyperedges are likely to be hit anyway, we give a smaller weight to hitting large edges. More precisely, given an estimate c of the cost of the solution (in our case, a previously found solution cost), the weight of a hyperedge E is $w(E) = \left(1 - \frac{c}{|V|}\right)^{|E|-1}$. The value of a vertex is the sum of the weights of the unhit edges it hits.

Local Search: Local search replaces some vertices of the solution by a smaller (or equal sized) set of vertices, obtaining a new solution that is not larger than the previous one. We perform local search by removing several vertices of the solution (that should not be too far from each other in the hypergraph), computing the hypergraph defined by the unhit edges, reducing this hypergraph, and then solving it with a greedy heuristic or an exact algorithm.

Clique and Independent Set: We use the MCS [7] and MCQ [6] algorithms implemented in <https://github.com/darrenstrash/open-mcs>. Maximum cliques and independent sets appear in two places of our solver. The first one is inside EvalMaxSAT, where we replaced the original EvalMaxSAT maximum clique algorithm by MCQ for sparse graphs (density under 0.1) and MCS for dense graphs. There is a performance gain for the rare cases when EvalMaxSAT uses dense graphs, and a small performance loss otherwise, but the modern C++ implementation made it easier to code a time limit without risking memory leaks. The second place where a maximum independent set solver is used is when the hitting set problem has all hyperedges of degree 2, which means we need to solve a minimum vertex cover problem.

Exact MaxSAT: It is easy to formulate minimum hitting set as a MaxSAT instance with one hard clause for each hyperedge and one variable (and soft clause) for each vertex. We used the 2024 version of EvalMaxSAT [1], which is based on the SAT solver Cadical [3]. We modified EvalMaxSAT in order to add a time limit to the computation. To make the task easier, we replaced the clique solver that EvalMaxSAT uses with the one mentioned above.

Heuristic MaxSAT: We also used the heuristic anytime MaxSAT solver tt-open-wbo-inc [4], especially the Nuwls solver in it [2]. This solver has not been modified and is called as a separate process with a timeout.

Integer Programming: Depending on the parameters used, EvalMaxSAT uses SCIP for solving integer programming for around one minute before attempting to solve the problem directly. We noticed, however, that GLPK is significantly faster for small instances that can be solved in under half a second.

3 Our Solvers

Now that we have presented the main tools, we are ready to describe our solvers.

Heuristic Solver:

- 1) We apply the three reduction rules.
- 2) We try to solve each connected component exactly with Integer Programming (GLPK) or MAXSAT (EvalMaxSAT) solvers, from small to large, aborting if taking too long. In fact, the greedy heuristic is used too, as it is optimal if the solution size is at most 2. If all components are solved, then we are done.
- 3) We reduce the problem to MAXSAT and run an anytime MAXSAT heuristic solver for around one to two minutes.
- 4) We then improve the solution using local search until we run out of time.

Step 4 deserves a more detailed explanation. In this step, we iteratively remove vertices from the solution until a *limit* is reached. We start by removing a random solution vertex and at each step we remove a vertex that shares a hyperedge with a previously removed vertex (we tried different strategies to choose the vertices, with no significant difference in the result). We stop when both *limit* hyperedges are unhit and at least $limit/2$ vertices have been removed from the solution. We then compute the subproblem defined by the unhit hyperedges, reduce it, and solve it exactly.

Choosing the right limit to use and whether to use GLPK or EvalMaxSAT are critical parts. There are many messy details in the code, but the general idea is the following. For each *limit*, we keep testing both solvers to see which one is faster and make the fastest one our main solver. The main solver is also tested with *limit* set to $limit + \delta$ and $limit - \delta$ for small δ and the limit is increased or decreased based on which limit gives a better improvement divided by the time taken. We also increase the limit if no improvement is obtained, and decrease it if the solver times out without a solution.

Exact Solver:

- 1) We apply the three reduction rules.
- 2) We try to solve each connected component exactly with Integer Programming (GLPK) or MAXSAT (EvalMaxSAT) solvers, from small to large, aborting if taking too long. If all components are solved, then we are done.
- 3a) If all hyperedges have degree 2 (and the number of vertices is at most 800), the problem is vertex cover and we solve it using the exact independent set solver MCS.
- 3b) Otherwise, we find a good heuristic solution (as steps 3 and 4 of the heuristic solver) and give it as an initial solution to EvalMaxSAT. Step 3b is repeated twice with different parameters.

References

- 1 Florent Avellaneda. A short description of the solver EvalMaxSAT. *MaxSAT Evaluation*, 8:364, 2020.
- 2 Yi Chu, Shaowei Cai, and Chuan Luo. Nuwls-c-2023: Solver description. *MaxSAT Evaluation*, 2023:23–24, 2023.
- 3 ABKFM Fleury and Maximilian Heisinger. Cadical, kissat, paracooba, plingeling and treengeling entering the sat competition 2020. *Sat Competition*, 2020:50, 2020.
- 4 Alexander Nadel. Anytime algorithms for MaxSAT and beyond. In *2020 Formal Methods in Computer Aided Design (FMCAD)*, pages 1–1. IEEE, 2020.
- 5 Lei Shi and Xuan Cai. An exact fast algorithm for minimum hitting set. In *2010 Third International Joint Conference on Computational Science and Optimization*, volume 1, pages 64–67. IEEE, 2010.
- 6 Etsuji Tomita and Tomokazu Seki. An efficient branch-and-bound algorithm for finding a maximum clique. In Cristian S. Calude, Michael J. Dinneen, and Vincent Vajnovszki, editors, *Discrete Mathematics and Theoretical Computer Science*, pages 278–289, 2003.
- 7 Etsuji Tomita, Yoichi Sutani, Takanori Higashi, Shinya Takahashi, and Mitsuo Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In *International workshop on algorithms and computation*, pages 191–203, 2010.