PACE Solver Description : DS7HS *

Sylwester Swat
Institute of Computing Science, Poznań University of Technology, Poland
sylwester.swat@put.poznan.pl

Abstract

This article briefly describes the most important algorithms and techniques used in DS7HS, a dominating set and hitting set solver submitted to the PACE 2025 contest. Used approaches for exact and heuristic tracks are described, for both the dominating set and the hitting set problem.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases dominating set, hitting set, exact algorithms, heuristic algorithms, large graphs, combinatorial optimization, graph algorithms

Category PACE 2025 Solver Description

1 Problem description

1.1 Dominating set

In the dominating set problem the goal is to find the smallest subset S of vertices of a given graph $G = (V, E)$ such that for each node $v \in V$ the intersection $N[v] \cap S$ is nonempty.

1.2 Hitting set

In the hitting set problem, the goal is to find, for a given family F of sets over universe F that smallest subset $S \subseteq U$, such that each set in F contains at least one element from U.

2 Solver overview

In this paper we provide a short description of the most important algorithms implemented in solver DS7HS. Due to a large variety of used methods, this description does not contain full information about used algorithms, their details and their behaviour in many distinct situations.

The DS7HS solver was designed to deal with the generalized dominating set problem (GDS), defined as follows. Given an undirected simple graph $G = (V, E)$ and sets of vertices $H, X \subseteq V$, find the smallest subset $S \subset V \setminus X$ such that for each node $v \in V \setminus H$ the intersection $N[v] \cap S$ is nonempty.

A solver implemented within DS7HS used to solve the hitting set problem works by solving the generalized dominating set problem for a bipartite graph $G' = (V', E')$ with bipartition sets A, B and sets $H = A$, $X = B$, where nodes in the set A represent elements in the universe U, nodes in set B represent sets in family F and there is an edge $\{a, b\} \in E'$ if and only if an element represented by a is contained in the set represented by b. This holds for both exact and heuristic solver for the hitting set problem. Due to this, we will mainly focues on the description of the dominating set solver.

* This is a brief description of a solver submitted to the PACE 2025 contest.

3 Preprocessing

We use an extensive set of data reduction rules for the dominating set problem. This set contains both known rules for the domination set problem ([1, 2]), extensions of those rules, as well as a variety of new reduction rules that were designed, optimized and implemented to be applicable even for very large graph instances. For a given instance of GDS, the goal is to find one of the following:

1. a node that belongs to some optimal solution
2. a node that can be marked as hit (added to set H)
3. a node that can be marked as excluded (added to set X)
4. a modification in the graph structure that could make the graph smaller or the structure easier is some sense, usually measured as the size of result obtained by the local search solver run on the reduced graph

Designed rules are called exhaustively, until no rule can be applied further, or for roughly 25% of total admissible running time, if the preprocessing did not terminate earlier. Many of our rules require lifting solution after finding result for the reduced graph instance. This is usually due to the fact that it is often possible to conclude that there exists a solution that does not contain a node (the node can be added to X) or that a node can be marked as hit and added to H, but only if certain additional constraints hold. Many of these constraints are usually fairly difficult to be taken into account when finding the solution using distinct heuristics and approaches, but it is much easier to fix a solution obtained for the reduced instance to make it a valid one for the original instance.

4 Exact algorithm

The exact algorithm is conceptually very simple: reduce the graph as much as possible using data reduction rules, then solve the reduced instance using a "general-purpose" solver, applied to the hitting-set formulation of the dominating set problem. In DS7HS we tested the following algorithms to solve the HS problem:

1. Integer Linear Programming: HiGHS solver [8] or SCIP solver [5]
2. MaxSAT: solving using MaxSAT approach and either CaDiCaL [4] or Glucose 3 [3] as a SAT solver
3. Constraint Programming: CP-SAT solver [9] from OR-tools software [10]

We also considered a simple branch-and-reduce approach, but our proof-of-concept implementation proved to be much slower than the use-a-"general-purpose"-solver approach, hence it's further development and optimization was abandoned. By default, the HS problem is solved using CP-SAT solver with three workers, as this proved to give the best results (compared to other approaches mentioned above).

5 Heuristic approach

The heuristic approach works in a *CEGAR (counter-example guided abstraction refinement [7])* fashion. Starting with an empty set C - a subset of a set $\mathbf{C}$ containing all neighborhood-constraints (sets of the form $N[v] \setminus X$) generated for the GDS instance, and an initially empty set S (a hitting set of sets in C), we iteratively add some subset of unsatisfied constraints (sets in $\mathbf{C}$ not hit by S), then update S to make it a valid hitting set of sets in C.

In order to generate a set of constraints to add to C, we select some number of nodes closest to the set S in considered graph G, nodes farthest from S, nodes of smallest/largest

degree and several randomly selected nodes, then add corresponding constraints. However, the total number of added constraints is limited to at most $\alpha \cdot |UH|^{\beta}$, where UH is the set of all nodes v for which the intersection $N[v] \setminus S$ is enmpty and α, β are parameters whose values depend on basic graph characteristics.

After selecting the set D of constraints to add, we find a hitting set S_D of all sets in D using a local search approach, then create a solution S' to $C \cup D$ by taking $S' = S \cup S_D$. Afterwards we take $S = S'$ and $C = C \cup D$ and try to optimize S using a local search approach. If the size of S could not be improved for a few subsequent iterations, a state perturbation is applied to try to get out of a found local optimum. A state perturbation is achieved by removing some small random subset of nodes in S and/or sets in C (and fixing S afterwards).

6 Local search approach

Our local search solver for the hitting set problem works by iteratively removing a node from a set S and adding a node back to S, unless it is not necessary. A node to remove from S is preferable over other nodes if it has a smaller value of a scoring function $ca(v)$, where $ca(v)$ is equal to the number of sets in C that are hit only by v. In case of a tie, a bunch of other heuristics are used. A node to add to S is preferable over other nodes if it has a greater value of a scoring function $hu(v)$, where $hu(v)$ is equal to the number of sets in C that are unhit by S and contain v. In case of a tie, a bunch of other heuristics are used, which form slight variations of rules that can be found in several existing heuristic dominating-set solvers (see, e.g., [11]).

Among the approaches used for tie-breaking, we use the following:

1. tabu search to prohibit visiting the same state more than once
2. configuration checking strategy [6]
3. selecting the less frequently chosen node if the two compared nodes do not have “similar” state-change frequencies
4. selecting the node that did not change its state for the longest time, if those times the two compared nodes are not “similar”

To effficiently track changes to values necessary to select next node to add/remove we use a custom-made heap that allows efficient modification of those values without necessity of removing and reinserting the nodes from/to the set. Random node moves are also applied to increase variability. Such random moves usually consist in removing a node and some number of nodes closest to it from the current solution, then adding the same number of nodes back, but using the standard comparator to select nodes to add back. Another method of increasing variability used in DS7HS consists in “freezing” some nodes - prohibting it from beeing selected for some number of consecutive iterations.

A variation to the standard add/remove approach is admitted and sometimes executed. This variation allows adding a node to the set S only if it will result in S being a valid solution. Since we start improvement with a valid solution, such a node always exists, but it might be the same that was just removed from S. We observed that this usually cuts the search space, but significantly improves the convergence speed on some instances.

7 Availability

The source code of DS7HS is freely available and can be found at `https://github.com/swacisko/pace-2025`.

References

1 Jochen Alber, Nadja Betzler, and Rolf Niedermeier. Experiments on data reduction for optimal domination in networks. *Annals of Operations Research*, 146, 09 2003. doi:10.1007/s10479-006-0045-4.

2 Jochen Alber, Michael Fellows, and Rolf Niedermeier. Polynomial time data reduction for dominating set. *Journal of the ACM*, 51, 05 2004. doi:10.1145/990308.990309.

3 Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solvers. pages 399–404, 07 2009.

4 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2024. doi:10.1007/978-3-031-65627-9_7.

5 Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and Lixing Xu. The SCIP Optimization Suite 9.0. ZIB-Report 24-02-29, Zuse Institute Berlin, February 2024. URL: https://nbn-resolving.org/urn:nbn:de:0297-zib-95528.

6 Shaowei Cai and Kaile Su. Configuration checking with aspiration in local search for sat. *Proceedings of the AAAI Conference on Artificial Intelligence*, 26(1):434–440, Sep. 2021. URL: https://ojs.aaai.org/index.php/AAAI/article/view/8133, doi:10.1609/aaai.v26i1.8133.

7 Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, September 2003. doi:10.1145/876638.876643.

8 Q. Huangfu and Julian Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10, 03 2015. doi:10.1007/s12532-017-0130-5.

9 Laurent Perron and Frédéric Didier. Cp-sat. URL: https://developers.google.com/optimization/cp/cp_solver/.

10 Laurent Perron and Vincent Furnon. Or-tools. URL: https://developers.google.com/optimization/.

11 Enqiang Zhu, Yu Zhang, Shengzhi Wang, Darren Strash, and Chanjuan Liu. A dual-mode local search algorithm for solving the minimum dominating set problem. 07 2023. doi:10.48550/arXiv.2307.16815.