PACE Solver Description: DSHunter

Paweł Putra ✉
University of Warsaw, Poland

Abstract

We describe DSHunter, our submission to the exact Dominating Set track of PACE 2025. DSHunter computes the minimium dominating set of a graph, by preprocessing the graph into a smaller kernel of an extended Annotated Dominating Set problem instance. Kernelized instance is then solved by either a Vertex Cover solver, dynamic programming on a tree decomposition or a Branch&Reduce algorithm depending on the instances structural properties.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Dominating Set, PACE 2025, Treewidth, Kernelization, Vertex Cover

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Supplementary Material The source code of DSHunter is available in `https://github.com/Kulezi/pace2025`

Acknowledgements I want to thank prof. Marcin Pilipczuk for guidance and insightful ideas that went into developing this solver.

1 Overview

DSHunter is an exact solver for finding a minimum dominating set of a graph. DSHunter was developed for the 9th Parameterized Algorithms and Computational Experiments challenge (PACE 2025). The core idea behind the solver is first preprocessing the given instance into a generalized Annotated Dominating Set (ADS) problem instance. In some cases the kernel yields an instance that is equivalent to Vertex Cover, and in those cases the solver calls an external Vertex Cover solver WeGotYouCovered [?] from PACE 2019. In other cases where the kernel admits small treewidth the solver finds the minimum dominating set directly by dynamic programming on a tree decomposition. The tree decomposition is found using a heuristic tree decomposition generator FlowCutter [?] from PACE 2017. In all other cases the solver resorts to a Branch&Reduce algorithm.

2 Notation

Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. The graph $G[X]$ is the induced subgraph of G with vertex set X. The set $N(v)$ denotes the open neighborhood of a vertex v and $N[v] = N(v) \cup \{u\}$ the closed neighborhood of a vertex v. The set $N[X] = \bigcup_{v \in X} N[v]$ denotes the closed neighborhood of a vertex set X. The set $N(X) = N[X] \setminus X$ denotes the open neighborhood of a vertex set X.

An Annotated Dominating Set problem instance I is tuple (G, d, m, f), where:

- $d : V(G) \to \{\texttt{true}, \texttt{false}\}$, we say that a vertex v is dominated iff $d(v) = \texttt{true}$, such vertices do not need to be dominated by the solution set.
- $m : V(G) \to \{\texttt{true}, \texttt{false}\}$, we say that a vertex v is disregarded iff $m(v) = \texttt{true}$, such vertices cannot belong to the solution set.
- $f : E(G) \to \{\texttt{true}, \texttt{false}\}$, we say that an edge e is forced iff $f(v) = \texttt{true}$, at least one endpoint of such an edge must belong to the solution set.

That is, a set X is a feasible solution if

(a) every vertex v with $d(v) = \texttt{false}$ is in $N[X]$,
(b) every vertex $x \in X$ satisfies $m(v) = \texttt{false}$,
(c) every edge $uv \in E(G)$ with $f(uv) = \texttt{true}$ satisfies $\{u, v\} \cap X \neq \emptyset$.
By saying we force an edge e we mean setting $f(e) = \texttt{true}$.

By saying we take a vertex v we mean setting $d(u) = \texttt{true}$ for all $u \in N[v]$, then removing v from G, and adding v to the solution set S.

By $C(v)$ we denote the set $\{u \in N[v] \mid d(v) = \texttt{false}\}$ or $C(v) = \emptyset$ in the case when v is disregarded, i.e. the set of undominated vertices vertex v can dominate (cover). We call those vertices dominatees of v.

By $D(v)$ we denote the set $\{u \in N[v] \mid m(v) = \texttt{false}\}$ or $D(v) = \emptyset$ in the case when v is dominated, i.e. the set of vertices that can dominate v. We call those vertices dominators of v.

By $N_{exit}(X)$ we denote the set

$$\{w \in N(X) \mid N(w) \setminus N[X] \neq \emptyset \vee \exists_{vw \in E(G[N(X)])} f(vw) = \texttt{true}\}$$

By $N_{guard}(X)$ we denote the set

$$\{w \in N(X) \setminus N_{exit}(X) \mid N(w) \cap N_{exit}(X) \neq \emptyset\}$$

By $N_{prison}(X)$ we denote the set

$$N[X] \setminus (N_{exit}(X) \cup N_{guard}(X))$$

By $\overline{N_{prison}}(X)$ we denote the set

$$N_{prison}(X) \cap \{u \in V(G) \mid d(u) = \texttt{false}\}$$

Note that a Dominating Set problem instance G can be expressed as an Annotated Dominating Set instance $I = (G, d, m, f, \emptyset)$, where d, m and f return `false` for all arguments.

3 Presolving

The solver starts by applying several reduction rules in a loop in the following order until all fail to apply. If a reduction rule succeeds, the loop starts from the beginning. Rules 3.5 and 3.6 are based on work by Xiong and Xiao [**?**]. Together with the instance the presolver maintains a set S of vertices removed by reduction rules known to belong to the optimal solution.

3.1 Removal of vertices seeing both ends of a forced edge

If there exists a vertex v such that there exists a forced edge vw, set $d(v) = \texttt{true}$ as it is guaranteed to be dominated by either v or w, since one of them must belong to the optimal solution set. This rule is an exception since it is not a part of the loop, but is applied the moment an edge is being forced, e.g. $f(e)$ changes from `false` to `true`.

3.2 ForceEdgeRule

If there exists an undominated vertex u of degree 2 with neighbors v and w, such that v or w is not disregarded:

- if $vw \in E(G)$ and $f(uv) = f(uw) = \texttt{false}$ then remove u from G and force edge vw.

- if $vw \in E(G)$ and exactly one of the edges uv, uw is forced and its non-u endpoint is not disregarded, take this endpoint into S.
- if $vw \notin E(G)$ and both v and w are dominated, remove u from G from add an edge vw to G and force it.

3.3 Disregard rule

If there exists $uv \in E(G)$ such that $m(v) = \texttt{false}$, $m(u) = \texttt{false}$, $C(u) \subseteq C(v)$ and there is no forced edge $uw \in E(G)$ s.t $v \neq w$ then disregard vertex u, since in an optimal solution we can always replace it with vertex v.

3.4 Remove Disregarded Rule

If there exists a disregarded, dominated vertex u, for all forced edges (u, v) take v and then remove u from G.

3.5 Single Dominator Rule

If there exists an undominated vertex v such that $D(v) = \{u\}$, take u.

3.6 Same Dominators Rule

If there exists a pair of undominated vertices u, v such that $D(v) \subseteq D(u)$, mark u as dominated.

3.7 Alber et al. rules

We have extended the rules described by Alber et al. [**?**], to apply to the ADS problem by careful analysis how they should handle disregarded vertices and forced edges. We did that for *Simple Rules 1, 2, 3 and 4*, as well as for *Main rules 1 and 2* from the referenced work.

3.7.1 Simple Rule 1

If there exists an unforced edge that both its endpoints are either dominated or disregarded, remove it from the graph.

If there exists a forced edge $uv \in E(G)$ s.t. u is not disregarded and v is disregarded, take vertex u into the solution set.

3.7.2 Simple Rule 2

If there exists an isolated dominated vertex v, remove it from the graph.

This rule was also meant apply to dominated vertices of degree 1, however due to an oversight we only apply it to isolated edges, for which it is optimal to take just one of the endpoints.

3.7.3 Simple Rule 3

If there exists a dominated vertex v of degree 2 with undominated neighbors u_1 and u_2, s.t. $u_1u_2 \in E(G)$ or u_1 and u_2 share a common, non-disregarded neighbor then:

- If the edge vu_1 is forced and the edge vu_2 is not, take u_1.
- If both of the edges vu_1, vu_2 are unforced, remove v from G.

Note that it is never the case that $u_1u_2 \in E(G)$ and both u_1 and u_2 are disregarded as *Simple Rule 1* is applied before *Simple Rule 3*.

3.7.4 Simple Rule 4

If there exists a dominated vertex v of degree 3 with undominated neighbors u_1, u_2 and u_3 s.t. $u_1u_2, u_1u_3 \in E(G)$ then

- If the vertex u_1 is not disregarded and the edges vu_2, vu_3 are unforced, remove v from G, also if the edge vu_1 was forced take u_1.

3.7.5 Main Rule 1

If there exists a non-disregarded vertex u such that $\overline{N_{prison}}(\{u\}) \neq \emptyset$ then take u, and remove $N_{prison}(\{u\})$ and $N_{guard}(\{u\})$ from G.

3.7.6 Main Rule 2

If there exists a pair of non-disregarded vertices v, w such that $\overline{N_{prison}}(\{v, w\}) \neq \emptyset$ and cannot be dominated by a single vertex from $N_{prison}(\{v, w\}) \cup N_{guard}(\{v, w\})$ then

1. If $\overline{N_{prison}}(\{v, w\})$ can be dominated by both just v and just w, and there are no forced edges incident to v and w, then:
   Let $R = N_{prison}(\{v, w\}) \cup (N_{guard}(\{v, w\}) \cap N(v) \cap N(w))$.
   - If $vw \in E(G)$, then force edge vw.
   - If $vw \notin E(G)$ and $|R| > 3$ add three new undominated disregarded vertices z_1, z_2, z_3 and non-forced edges vz_i, wz_i for $i \in \{1, 2, 3\}$ to G.

   If one of the above conditions were true, remove the vertex set R from G.
2. If $\overline{N_{prison}}(\{v, w\})$ can be dominated by just v but not w, and there are no forced edges incident to w then take w and remove $N_{prison}(\{v, w\})$ and $N_{guard}(\{v, w\}) \cap N(v)$ from G.
3. If $\overline{N_{prison}}(\{v, w\})$ can neither be dominated by just v or just w, take both v and w and remove $N_{prison}(\{v, w\}) \cup N_{guard}(\{v, w\})$ from G.

3.8 Local bruteforce rule

If there exists $X \subseteq V(G)$ and $T_A \subseteq A = N[X] \setminus \{w \in N(X) \mid N(w) \setminus N[X] \neq \emptyset\}$, such that for all possible choices of a proper solution set $T_B \subseteq V[G] \setminus A$ for $I[V(G) \setminus N[X]]$ set T_A is a smallest subset of A such that $T_A \cup T_B$ is a proper solution of I, take all vertices from T_A and disregard all vertices from $A \setminus T_A$.

This rule can be applied in time $\mathcal{O}(|G| \cdot 4^{|A|})$ for a given X.

In our solver we limit our choice of sets X to all subsets of the bags of a heurisitcally found tree decomposition with size at most 10, and the sets of all vertices at a distance at most r for $r \in \{0, 1, 2, 3\}$, for which $|X|, |A| \leq 10$.

4 Solving the kernel

After preprocessing is done the solver attempts to solve each of the graph connected components independently using the following methods in order.

1. If the kernelized instance has only forced edges, then it is a Vertex Cover instance, solve it using an external Vertex Cover solver.

2. If the kernel admits a tree decomposition of a small enough width to fit in the memory limit within 5 minutes of running an external decomposer, solve using dynamic programming on the found decomposition. The solver also attempts to reduce the instance by branching on the vertices belonging to the largest bags of the decomposition.
3. If all previous methods fail, perform an exhaustive Branch&Reduce search for the optimal solution.

4.1 Tree decomposition dynamic programming

Our implementation of tree decomposition dynamic programming for Dominating Set is a direct implementation of the algorithm described in the book Parameterized Algorithms [**?**], extended to work for ADS.

The modifications are:

- We count the vertex in its **Forget** node instead of its **Introduce vertex** node .
- A vertex is counted with a weight of $+\infty$ if it is disregarded.
- In the **Introduce Edge**, we return $+\infty$ for the case where no endpoint of this edge is taken into the solution.

4.2 Branch&Reduce

Our branching algorithm works by branching first on the vertices with maximum number of incident forced edges. If there are no such vertices, the solver branches on the vertex with minimum degree in the graph. After each decision made by branching the same reductions as in the presolver are applied. When the graph becomes disconnected as a result of branching the solver solves each component independently using branching.

We use a the size maximal 3-scattered set of G as a lower bound to discard branches that are guaranteed to produce unoptimal solutions.