Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 28, no. 1, pp. 439–473 (2024)
DOI: 10.7155/jgaa.v28i1.2997

Finding Near-Optimal Weight Independent Sets at Scale

Ernestine Großmann[1] *Sebastian Lamm*[2] *Christian Schulz*[1] *Darren Strash*[3]

[1]Heidelberg University, Germany
[2]Karlsruhe Institute of Technology, Germany
[3]Department of Computer Science, Hamilton College, United States

Submitted: March 2024	Accepted: October 2024	Published: November 2024
Article type: Regular Paper		Communicated by: S. Cornelsen

Abstract. Computing maximum weight independent sets in graphs is an important NP-hard optimization problem. The problem is particularly difficult to solve in large graphs for which data reduction techniques do not work well. To be more precise, state-of-the-art branch-and-reduce algorithms can solve many large-scale graphs if reductions are applicable. Otherwise, their performance quickly degrades due to branching requiring exponential time. In this paper, we develop an advanced memetic algorithm to tackle the problem, which incorporates recent data reduction techniques to compute near-optimal weight independent sets in huge sparse networks. More precisely, we use a memetic approach to recursively choose vertices that are likely to be in a large-weight independent set. We include these vertices into the solution, and further reduce the graph. We show that identifying and removing vertices likely to be in large-weight independent sets opens up the reduction space and speeds up the computation of large-weight independent sets remarkably. Our experimental evaluation indicates that we are able to outperform state-of-the-art algorithms. For example, our two algorithm configurations compute the best results among all competing algorithms for all instances tested. Thus, it can be seen as a useful tool when large-weight independent sets need to be computed in practice.

We acknowledge support by DFG grant SCHU 2567/3-1.

E-mail addresses: e.grossmann@informatik.uni-heidelberg.de (Ernestine Großmann) lamm@ira.uka.de (Sebastian Lamm) christian.schulz@informatik.uni-heidelberg.de (Christian Schulz) dstrash@hamilton.edu (Darren Strash)

1 Introduction

For a given graph $G = (V, E)$ an *independent set* (IS) is defined as a subset $I \subseteq V$ of all vertices such that each pair of vertices in I are non adjacent. A *maximum independent set* (MIS) describes an IS with highest possible cardinality. By transforming the graph G into the complement graph $\overline{G}$ the MIS problem results in the *maximum clique* problem. However, for sparse graphs G, using a maximum clique solver is impractical as the complement $\overline{G}$ is very dense and therefore unlikely to fit in memory for all but the smallest instances. Another related problem is the *minimum vertex cover* problem. Note that for an MIS $\mathcal{I}$ of G, $V \setminus \mathcal{I}$ is a minimum vertex cover. For a weighted graph $G = (V, E, \omega)$ with non-negative vertex weights given by a function $\omega : V \to \mathbb{R}_{\geq 0}$, the *maximum weight independent set* (MWIS) problem is to find an independent set $\mathcal{I}$ with maximum weight $\omega(\mathcal{I}) = \sum_{v \in \mathcal{I}} \omega(v)$. The applications of the MWIS problem, as well as the related problems addressed above, can be used for solving different application problems such as long-haul vehicle routing [15], the winner determination problem [58] or prediction of structural and functional sites in proteins [38]. As a detailed example, consider an application of MWIS for map labeling, where displaying non-overlapping labels throughout dynamic map operations such as zooming and rotating [21] or while tracking a physical movement of a user or set of moving entities [9] is of high interest in many applications. In the underlying map labeling problem, the labels are represented by vertices in a graph, weighted by importance. Each pair of vertices is connected by an edge if the two corresponding labels would overlap. In this graph, a MWIS describes a high-quality set of labels, with regard to their importance level, that can be visualized without any overlap.

Since these problems are NP-hard [19], heuristic algorithms are used in practice to efficiently compute solutions of high quality on *large* graphs [6, 22, 60]. Depending on the definition of the neighborhood, local search algorithms are able to explore local solution spaces very effectively. However, local search algorithms are also prone to get stuck in local optima. As with many other heuristics, results can be improved if several repeated runs are made with some measures taken to diversify the search. Still, even a large number of repeated executions can only scratch the surface of the huge space of possible independent sets for large-scale datasets.

Traditional branch-and-bound methods [20, 26, 32, 44, 45, 53] may often solve small graphs with hundreds to thousands of vertices in practice, and medium-sized instances can be solved exactly in practice using reduction rules to reduce the graph. In particular, it has been observed that if data reductions work very well, then the instance is likely to be solved. If data reductions do not work very well, i.e. the size of the reduced graph is large, then the instance can often not be solved. Even though algorithms such as the struction algorithm, as shown in [20] already manage to solve many large instances, some remain unsolved.

In order to explore the global solution space extensively, more sophisticated metaheuristics, such as GRASP [15] or iterated local search [6, 40], have been used. In this work, we extend the set of metaheuristics used for the MWIS problem by introducing a novel memetic algorithm. Memetic algorithms (MAs) combine genetic algorithms with local search [29] to effectively explore (via global search) and exploit (via local search) the solution space. The general idea behind genetic algorithms is to use mechanisms inspired by biological evolution such as selection, mutation, recombination, and survival of the fittest.

Our Results. Our contribution is two-fold: First, we develop a state-of-the-art memetic algorithm that is based on recombination operations employing graph partitioning techniques. Our algorithm computes large-weight independent sets by incorporating a wide range of recently developed advanced reduction rules. In particular, our algorithm uses a wide range of frequently used data reduction techniques from [20, 32] and also employs a number of recently proposed data reduction rules by Gu et al. [23].

The algorithm may be viewed as performing two functions simultaneously: (1) reduction rules for the weighted independent set problem are used to boost the performance of the memetic algorithm *and* (2) the memetic algorithm opens up the opportunity for further reductions by selecting vertices that are likely to be in large-weight independent sets. In short, our method applies reduction rules to form a reduced graph, then computes vertices to insert into the final solution and removes these vertices and their neighbors from the graph. Then, further reductions can be applied. The process is then repeated recursively until the graph is empty. We show that this technique finds near-optimal weight independent sets much faster than existing local search algorithms, is competitive with state-of-the-art exact algorithms for smaller graphs, and allows us to compute large-weight independent sets on huge sparse graphs. Overall, our algorithm configurations compute the best results among all competing algorithms for every instance, and thus can be seen as the dominating tool when large weight independent sets need to be computed in practice.

Our second contribution in this work is the experimental evaluation of the orderings in which currently available data reductions are applied. We examine the impact of different orderings on solution size and on running time. One outcome of this evaluation are robust orderings of reductions for exact reduction rules, as well as a specific ordering which can improve the solution quality at the expense of computation time.

2 Preliminaries

In this work, a graph $G = (V, E)$ is an undirected graph with $n = |V|$ and $m = |E|$, where $V = \{0, ..., n-1\}$. The neighborhood $N(v)$ of a vertex $v \in V$ is defined as $N(v) = \{u \in V : (u, v) \in E\}$. Additionally, $N[v] = N(v) \cup \{v\}$. The same sets are defined for the neighborhood $N(U)$ of a set of vertices $U \subset V$, i.e. $N(U) = \cup_{v \in U} N(v) \setminus U$ and $N[U] = N(U) \cup U$. The degree of a vertex $\deg(v)$ is defined as the number of its neighbors $\deg(v) = |N(v)|$. The complement graph is defined as $\overline{G} = (V, \overline{E})$, where $\overline{E} = \{(u, v) : (u, v) \notin E\}$ is the set of edges not present in G. A set $I \subseteq V$ is called *independent set* (IS) if for all vertices $v, u \in I$ there is no edge $(v, u) \in E$. For a given IS $\mathcal{I}$ a vertex $v \notin \mathcal{I}$ is called free, if $\mathcal{I} \cup \{v\}$ is still an independent set. An IS is called *maximal* if there are no free vertices. The *maximum independent set problem* (MIS) is that of finding an IS with maximum cardinality. The *maximum weight independent set problem* (MWIS) is that of finding an IS with maximum weight. The weight of an independent set $\mathcal{I}$ is defined as $\omega(\mathcal{I}) = \sum_{v \in \mathcal{I}} \omega(v)$ and $\alpha_\omega(G)$ denotes the weight of an MWIS of G. The complement of an independent set is a vertex cover, i.e. a subset $C \subseteq V$ such that every edge $e \in E$ is covered by at least one vertex $v \in C$. An edge is *covered* if it is incident to one vertex in the set C. The minimum vertex cover problem, defined as looking for a vertex cover with minimum cardinality, is thereby complementary to the MIS problem. Another closely related concept are cliques. A *clique* is a set $Q \subseteq V$ such that all vertices are pairwise adjacent. A clique in the complement graph $\overline{G}$ corresponds to an independent set in the original graph G. A vertex is called *simplicial*, when its neighborhood forms a clique.

The subdivision of the set of vertices V into disjoint blocks $V_1, ..., V_k$ such that $V_1 \cup ... \cup V_k = V$ is called a *k-way partition* (see [13, 50]). To ensure the blocks are roughly of the same size, the balancing constraint $|V_i| \leq L_{max} \coloneqq (1 + \varepsilon) \left\lceil \frac{|V|}{k} \right\rceil$ with the imbalance parameter $\varepsilon > 0$ is introduced. While satisfying this balance constraint, the *edge separator* problem asks for minimizing the total cut, $\sum_{i<j} \omega(E_{ij})$, where E_{ij} is defined by $E_{ij} \coloneqq \{\{u, v\} \in E : u \in V_i, v \in V_j\}$. The edge separator

is the set of all edges in the cut. For the k-vertex separator problem, on the other hand, we look for a division of V into $k+1$ blocks. In addition to the blocks $V_1, ..., V_k$ a separator S exists. This separator has to be chosen such that no edges between the blocks $V_1, ..., V_k$ exist, but there is no balancing constraint on the separator S. However, as for the edge separator problem the balancing constraint on the blocks $|V_i| \leq L_{max} := (1+\varepsilon)\left\lceil \frac{|V|}{k} \right\rceil$ has to hold. To solve the problem, the size of the separator $|S|$ has to be minimized. By removing the separator S from the graph it results in at least k connected components, since the different blocks V_i are not necessarily connected.

3 Related Work

We give a short overview of existing work on both exact and heuristic procedures. For more details on data reduction techniques, we refer the reader to the recent survey [3].

3.1 Exact Methods

Exact algorithms usually compute optimal solutions by systematically exploring the solution space. A frequently used paradigm in exact algorithms for combinatorial optimization problems is called *branch-and-bound* [41, 55]. In the case of the MWIS problem, these types of algorithms compute optimal solutions by case distinctions in which vertices are either included into the current solution or excluded from it, branching into two or more subproblems and resulting in a search tree. Over the years, branch-and-bound methods have been improved by new branching schemes or better pruning methods using upper and lower bounds to exclude specific subtrees [7, 8, 35]. In particular, Warren and Hicks [55] proposed three branch-and-bound algorithms that combine the use of weighted clique covers and a branching scheme first introduced by Balas and Yu [8]. Their first approach extends the algorithm by Babel [7] by using a more intricate data structures to improve its performance. The second one is an adaptation of the algorithm of Balas and Yu, which uses a weighted clique heuristic that yields structurally similar results to the heuristic of Balas and Yu. The last algorithm is a hybrid version that combines both algorithms and is able to compute optimal solutions on graphs with hundreds of vertices.

In recent years, reduction rules have frequently been added to branch-and-bound methods yielding so-called *branch-and-reduce* algorithms [4]. These algorithms are able to improve the worst-case runtime of branch-and-bound algorithms by applying reduction rules to the current graph before each branching step. For the unweighted case, a large number of branch-and-reduce algorithms have been developed in the past. The currently best exact solver [26], which won the PACE challenge 2019 [26, 42, 52], uses a portfolio of branch-and-reduce/bound solvers for the complementary problems. Recently, novel branching strategies have been presented in [25] to further improve both branch-and-bound as well as branch-and-reduce approaches.

However, for a long time, virtually no weighted reduction rules were known, which is why hardly any branch-and-reduce algorithms exist for the MWIS problem. The first branch-and-reduce algorithm for the weighted case was presented by Lamm et al. [32]. The authors first introduce two meta-reductions called neighborhood removal and neighborhood folding, from which they derive a new set of weighted reduction rules. On this foundation a branch-and-reduce algorithm is developed using pruning with weighted clique covers similar to the approach by Warren and Hicks [55] for upper bounds and an adapted version of the ARW local search [6] for lower bounds.

This algorithm was then extended by Gellner et al. [20] to utilize different variants of the struction, originally introduced by Ebenegger et al. [16] and later improved by Alexe et al. [5]. In contrast to previous reduction rules, these do not necessarily decrease the graph size, but rather transform the graph which later can lead to even further reduction. Those variants were integrated into the framework of Lamm et al. [32] in the preprocessing as well as in the reduce step. The experimental evaluation shows that this algorithm can solve a large set of real-world instances and outperforms the branch-and-reduce algorithm by Lamm et al. [32], as well as different state-of-the-art heuristic approaches such as the algorithm HILS presented by Nogueira [40] as well as two other local search algorithms DynWVC1 and DynWVC2 by Cai et al. [11]. Recently, Xiao et al. [60] present further data reductions for the weighted case as well as a simple exact algorithm based on these data reduction rules. Furthermore, in [63] a new reduction-and-branching algorithm was introduced using two new reduction rules. Recently, Xiao et al. [59] also presented a branch-and-bound algorithm idea using reduction rules working especially well on sparse graphs. In their theoretical work they undertake a detailed analysis for the running time bound on special graphs. With the measure-and-conquer technique they can show that the running time of their algorithm is $\mathcal{O}^*(1{,}1443^{(0.624x-0.872)n})$ where x is the average degree of the graph. This is improving previous time bounds for this problem using polynomial space complexity for graphs of average degree up to three.

Figiel et al. [18] introduced a new idea added to the state-of-the-art way of applying reductions. They propose to not only performing reductions, but also the possibility of undoing them during the reduction process. As they showed in their paper for the unweighted independent set problem, this can lead to new possibilities to apply further reductions and finally to smaller reduced graphs.

Finally, there are exact procedures which are either based on other extension of the branch-and-bound paradigm, e.g. [43, 56, 57], or on the reformulation into other $\mathcal{NP}$-complete problems, for which a variety of solvers already exist. For instance, Xu et al. [62] developed an algorithm called SBMS, which calculates an optimal solution for a given MWVC instance by solving a series of SAT instances. Also for the MWVC problem a new exact algorithm using the branch-and-bound idea combined with data reduction rules was recently presented [54]. We additionally note that there are several recent works on the complementary maximum weighted clique problem that are able to handle large real-world networks [17, 24, 27]. However, using these solvers for the MWIS problem requires computing complement graphs. Since large real-world networks are often very sparse, processing their complements quickly becomes infeasible due to their memory requirement.

3.2 Heuristic Methods

A widely used heuristic approach is local search, which usually computes an initial solution and then tries to improve it by simple insertion, removal or swap operations. Although in theory local search generally offers no guarantees for the solution's quality, in practice they find high-quality solutions significantly faster than exact procedures.

For unweighted graphs, the iterated local search (ARW) by Andrade et al. [6], is a very successful heuristic. It is based on so-called $(1, 2)$-swaps which remove one vertex from the solution and add two new vertices to it, thus improving the current solution by one. Their algorithm uses special data structures which find such a $(1, 2)$-swap in linear time in the number of edges or prove that none exists. Their algorithm is able to find (near-)optimal solutions for small to medium-size instances in milliseconds, but struggles on massive instances with millions of vertices and edges.

The hybrid iterated local search (HILS) by Nogueira et al. [40] adapts the ARW algorithm for weighted graphs. In addition to weighted $(1, 2)$-swaps, it also uses $(\omega, 1)$-swaps that add one vertex v into the current solution and exclude its ω neighbors. These two types of neighborhoods are explored separately using variable neighborhood descent (VND). Two other local searches, DynWVC1 and DynWVC2, for the equivalent minimum weight vertex cover problem are presented by Cai et al. [11]. Their algorithms extend the existing FastWVC heuristic [37] by dynamic selection strategies for vertices to be removed from the current solution. In practice, DynWVC1 outperforms previous MWVC heuristics on map labeling instances and large scale networks, and DynWVC2 provides further improvements on large scale networks but performs worse on map labeling instances.

Li et al. [36] presented a local search algorithm NuMWVC for the minimum weight vertex cover (MWVC) problem, which is complementary to the MWIS problem. Their algorithm applies reduction rules during the construction phase of the initial solution. Furthermore, they adapt the configuration checking approach [12] to the MWVC problem which is used to reduce cycling, i.e. returning to a solution that has been visited recently. Finally, they develop a technique called self-adaptive-vertex-removing, which dynamically adjusts the number of removed vertices per iteration. Experiments show that their algorithm outperforms state-of-the-art approaches on both graphs of up to millions of vertices and real-world instances.

Recently, a hybrid method was introduced by Langedal et al. [33] to also solve the MWVC problem. For this approach they combined elements from exact methods with local search, data reductions and graph neural networks. In their experiments they achieve definite improvements compared to DynWVC2 and the HILS algorithm in both solution quality and running time.

With EvoMIS, Lamm et al. [30] presented an evolutionary approach to tackle the maximum independent set problem. The key feature of their algorithm is to use graph partitioning to come up with natural combine operations, where whole blocks of solutions to the MIS problem can be exchanged easily. To these combine operations also local search algorithms were added to improve the solutions further. Combining the branch-and-reduce approach with the evolutionary algorithm EvoMIS, a reduction evolution algorithm ReduMIS was presented by Lamm et al. [31]. In their experiments, ReduMIS outperformed the local search ARW as well as the pure evolutionary approach EvoMIS. Another reduction based heuristic called HtWIS was presented recently by Gu et al. [23]. In their framework they repeatedly apply reductions exhaustively and then choose one vertex by a tie-breaking policy to add to the solution. Now this vertex as well as its neighbors can be removed from the graph and the reductions can be applied again. Their experiments prove a significant improvement in running time.

Recently, a new metaheuristic was introduced by Dong et al. [15] in particular for vehicle routing instances. With their algorithm METAMIS they developed a new local search algorithm using a new variant of path-relinking to escape local optima. In their experiments they outperform HILS algorithm on a wide range of instances both in time and solution quality.

4 Algorithm

We now present our memetic algorithm for the MWIS problem, which we call *memetic maximum weight independent set* $\textsc{m}^2\textsc{wis}$. This algorithm is inspired by ReduMIS [31] and works in rounds, where each round can be split up into three parts. In the beginning of each round the exact reduction step takes place. Here the graph is reduced as far as possible using a wide range of data reduction rules. On the resulting reduced graph, we apply the memetic part of the algorithm as the second step. We represent a solution, also referred to as an individual, by using bitvectors. Where the independent set $\mathcal{I}$ is represented as an array $s \in \{0,1\}^n$. For each array entry it holds $s[v] = 1$ iff $v \in \mathcal{I}$. The memetic component itself works in rounds as well. Starting with an initial population $\mathcal{P}$, consisting of a set of individuals, this population is evolved over several rounds until a stopping criterion is fulfilled. In the third part, we select a subset of vertices to

Algorithm 1 High Level Structure of M^{2}WIS

```
input graph G = (V, E)
procedure M²WIS(G)
  W = ∅    // best solution
  while G not empty and time limit not reached
    (G, W) ← ExactReduce(G, W)
    if G is empty then return W
    if V(G) ≤ n_K
      W* ← try_to_solve_exact(G, t_exact)
      if W* optimal then return W*
    create initial population P (adding W* if computed)
    P ← Evolve(G, P)
    (G, W) ← HeuristicReduce(G, P, W)
return W
```

be included in the independent set by considering the resulting population. Here, we implement different strategies to select vertices for inclusion. Including these vertices in the independent set enables us to remove them and their neighbors from the instance. This opens up the reduction space, i.e. further reductions might be applicable after the removal process. The steps of exact reduction, memetic search, and heuristic reduction are repeated until the remaining graph is empty or another stopping criterion is fulfilled.

Following the order of the Algorithm 1, we first describe the ExactReduce routine in Section 4.1. Section 4.2 is devoted to the memetic part, followed by the description of different vertex selection strategies used to heuristically reduce the instance and open up the reduction space in Section 4.3.

4.1 Exact Reductions

Especially for large instances, applying exact data reductions is a very important technique to reduce the problem size. In general, reductions identify vertices (1) as part of a solution to the MWIS problem, (2) as non-solution vertices or (3) as deferred, meaning the decision for this vertex is depending on additional information about neighboring vertices that will be obtained later. The resulting reduced graph, after no reduction rule can be applied anymore, we denote by $\mathcal{K}$. Once an MWIS of $\mathcal{K}$ is found, reductions can be undone to reconstruct an MWIS on the original graph. For the reduction process we apply a large set of reductions which we list in the following. In the following list of reductions $\mathcal{I}$ refers to an MWIS of G, $\mathcal{I}'$ to an MWIS of the modified graph G'.

Reduction 1 (Degree-One [23, 32]) *Let v be a degree-one vertex with the neighbor u in G.*

- *Case 1: if $\omega(v) \geq \omega(u)$, v must be contained in some MWIS of G; thus v can be removed from G, i.e. $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$, where G' is the graph obtained by removing both v and u.*
- *Case 2: if $\omega(v) < \omega(u)$, $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$, where G' is the graph obtained by removing v and updating the weight of u to be $\omega(u) = \omega(u) - \omega(v)$. It holds that $u \in \mathcal{I}$ iff $u \in \mathcal{I}'$.*

We note that Case 1 is a special case of Reduction 2.

Reduction 2 (Neighborhood Removal [32]) *For any $v \in V$, if $\omega(v) \geq \omega(N(v))$ then v is in some MWIS of G. Let $G' = G[V \setminus N[v]]$ and $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$.*

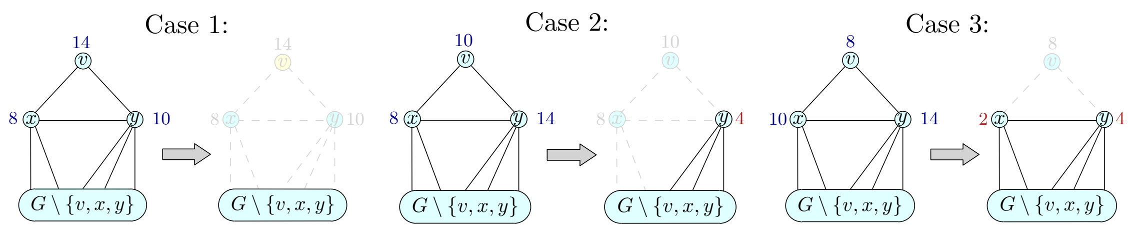

Figure 1: Illustration of the three cases of the Triangle Reduction.

Reduction 3 (Triangle [23]) *This reduction is illustrated in Figure 1. Let v be a degree-two vertex with two neighbors x and y in G, where edge $\{x, y\} \in E$. Without loss of generality, assume $\omega(x) \leq \omega(y)$.*

- *Case 1: if $\omega(v) \geq \omega(y)$, v must be contained in some MWIS of G. This leads to $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$, where G' is obtained by removing v, x, y.*
- *Case 2: if $\omega(x) \leq \omega(v) < \omega(y)$, $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$, where G' is the graph obtained by removing nodes v and x, and updating $\omega(y) = \omega(y) - \omega(v)$. It holds that $y \in \mathcal{I}$ iff $y \in \mathcal{I}'$.*
- *Case 3: if $\omega(v) < \omega(x)$, $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$, where G' is the graph obtained by removing v, and updating $\omega(x) = \omega(x) - \omega(v)$ as well as $\omega(y) = \omega(y) - \omega(v)$. It holds for $z \in \{x, y\}$ that $z \in \mathcal{I}$ iff $z \in \mathcal{I}'$.*

Reduction 4 (Extended V-Shape [23, 32]) *This reduction is illustrated in Figure 2. Let v be a degree-two vertex with the neighbors x and y in G, where edge $\{x, y\} \notin E$. Without loss of generality, assume $\omega(x) \leq \omega(y)$.*

- *Case 1: [32] if $\omega(v) \geq \omega(y)$*
  - *if $\omega(v) \geq \omega(x) + \omega(y)$, v must be contained in some MWIS of G and this leads to $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$, where G' is obtained by removing v, x, y.*
  - *else we fold v, x, y into a vertex v' with weight $\omega(v') = \omega(x) + \omega(y) - \omega(v)$ forming a new graph G'. Then $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$. If $v' \in \mathcal{I}'$ then $\{x, y\} \subset \mathcal{I}$, otherwise $v \in \mathcal{I}$. It holds that $\{x, y\} \subset \mathcal{I}$ iff $v' \in \mathcal{I}'$.*
- *Case 2: [23] if $\omega(x) \leq \omega(v) < \omega(y)$, $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$, where G' is the graph obtained by removing v, updating $N(x) = N(x) \cup N(y)$ and $\omega(y) = \omega(y) - \omega(v)$. It holds that $\forall w \in \{x, y\}$, $w \in \mathcal{I}$ iff $w \in \mathcal{I}'$;*
- *Case 3: [23] if $\omega(x) > \omega(v)$, $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$, where G' is the graph obtained by updating $\omega(x) = \omega(x) - \omega(v)$, $\omega(y) = \omega(y) - \omega(v)$, and $N(v) = N(x) \cup N(y)$. It holds that $\{x, y\} \subseteq \mathcal{I}$ iff $\{x, y\} \subseteq \mathcal{I}'$.*

Reduction 5 (Simplicial Vertex Removal [32]) *Let $v \in V$ be simplicial and $\max_{u \in N(v)} \omega(u) \leq \omega(v)$. Then, v is in some MWIS of G. Let $G' = G[V \setminus N[v]]$ and $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$.*

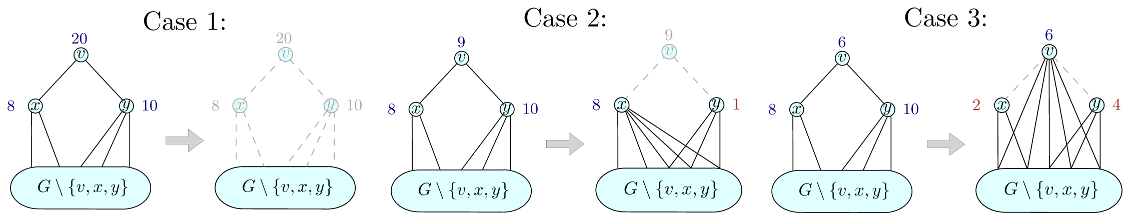

Figure 2: Illustration of the three cases of the Extended V-Shape Reduction.

The Basic Single Edge Reduction is a generalization of the Weighted Domination Reduction [32]. Here we can exclude an endpoint of an edge, if the other endpoint would always be the better option for the solution. The Extended Single Edge Reduction can be applied, when one of the two endpoints of an edge has to be in the solution.

Reduction 6 (Basic Single-Edge [23]) *Given an edge $e(u,v) \in E_G$, if $\omega(v) + \omega(N(u) \setminus N[v]) \leq \omega(u)$, it holds that $\alpha_\omega(G) = \alpha_\omega(G')$, where G' is obtained by removing v from G.*

Reduction 7 (Extended Single-Edge [23]) *Given an edge $e(u,v) \in E_G$, if $\omega(v) \geq \omega(N(v)) - \omega(u)$, it holds that $\alpha_\omega(G) = \alpha_\omega(G')$, where G' is obtained by removing all vertices in $N(u) \cap N(v)$.*

The Twin Reduction deals with vertices that have the same, independent neighborhood. Depending on the weight of of these vertices and their neighborhood, we can proceed differently.

Reduction 8 (Twin [32]) *Let vertices u and v have equal neighborhoods $N(u) = N(v)$, forming an independent set. We have two cases:*

1. *If $\omega(\{u,v\}) \geq \omega(N(v))$, then u and v are in some MWIS of G. Let $G' = G[V \setminus N[\{u,v\}]]$.*

2. *If $\omega(\{u,v\}) < \omega(N(v))$, but $\omega(\{u,v\}) > \omega(N(v)) - \min_{x \in N(v)} \omega(x)$, then we can fold u, v, p, q, r into a new vertex v' with weight $\omega(v') = \omega(N(v)) - \omega(\{u,v\})$ and call this graph G'. Then we construct an MWIS $\mathcal{I}$ of G as follows: if $v' \in \mathcal{I}'$ then $\mathcal{I} = (\mathcal{I}' \setminus \{v'\}) \cup N(v)$, if $v' \notin \mathcal{I}'$ then $\mathcal{I} = \mathcal{I}' \cup \{u,v\}$.*

Furthermore, $\alpha_\omega(G) = \alpha_\omega(G') + \omega(\{u,v\})$.

The Simplicial Weight Transfer is a generalization of Reduction 5. If we can not identify a simplicial vertex that is in some MWIS of G, we can still exclude or reduce the weight of some vertices in the clique.

Reduction 9 (Simplicial Weight Transfer [32]) *Let $v \in V$ be simplicial, and suppose that the set of simplicial vertices $S(v) \subseteq N(v)$ is such that $\forall u \in S(v)$, $\omega(v) \geq \omega(u)$. We*

1. *remove all $u \in N(v)$ such that $\omega(u) \leq \omega(v)$, and let the remaining neighbors be denoted by $N'(v)$,*

2. *remove v and $\forall x \in N'(v)$ set its new weight to $\omega'(x) = \omega(x) - \omega(v)$, and*

let the resulting graph be denoted by G'. Then $\alpha_\omega(G) = \omega(v) + \alpha_\omega(G')$ and an MWIS $\mathcal{I}$ of G can be constructed from an MWIS $\mathcal{I}'$ of G' as follows: if $\mathcal{I}' \cap N'(v) = \emptyset$ then $\mathcal{I} = \mathcal{I}' \cup \{v\}$, otherwise $\mathcal{I} = \mathcal{I}'$.

Reduction 10 (CWIS [10]) *Let $I_c \subseteq V$ be a critical weighted IS of G, i.e. $\omega(\mathcal{I}_c) - \omega(N(\mathcal{I}_c)) = \max\{\omega(\mathcal{I}) - \omega(N(\mathcal{I})) : \mathcal{I}$ is an IS of $G\}$. Then $\mathcal{I}_c$ is in some MWIS of G. We set $G' = G[V \setminus N[\mathcal{I}_c]]$ and $\alpha_\omega(G) = \alpha_\omega(G') + \omega(\mathcal{I}_c)$.*

If the neighborhood of a vertex v is independent and Reduction 2 is not applicable, then, under certain weight conditions, we can fold the vertex with its neighborhood as explained in the Neighborhood Folding.

Reduction 11 (Neighborhood Folding [32]) *Let $v \in V$, and suppose that $N(v)$ is independent. If $\omega(N(v)) > \omega(v)$, but $\omega(N(v)) - \min_{u \in N(v)}\{\omega(u)\} < \omega(v)$, then fold v and $N(v)$ into a new vertex v' with weight $\omega(v') = \omega(N(v)) - \omega(v)$. If $v' \in \mathcal{I}'$ then $\mathcal{I} = (\mathcal{I}' \setminus \{v'\}) \cup N(v)$, otherwise if $v \in \mathcal{I}'$ then $\mathcal{I} = \mathcal{I}' \cup \{v\}$. Furthermore, $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$.*

The Heavy Set Reduction is the most general and expensive reduction, which is why we only apply it after all other reductions have been exhaustively applied. For this reduction rule, we have to solve multiple independent set problems in the neighborhood of two heavy vertices. Under certain conditions explained in Reduction 12, we can then include these two heavy vertices. Because it is computationally expensive, we limit the size of the neighborhood of these heavy vertices.

Reduction 12 (Heavy Set [61]) *Let u and v be non-adjacent vertices having at least one common neighbor x and let the number of their neighbors $|N(\{v, u\})|$ be at most 8. If for any independent set $\mathcal{I}'$ in the induced subgraph $G[N(\{v, u\})]$, $\omega(N(\mathcal{I}')) \cap \{u, v\} \geq \omega(\mathcal{I}')$ there is a MWIS in G that includes u and v. Then, $G' = G - N[\{v, u\}]$ and $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v) + \omega(u)$.*

In the ExactReduce routine, we test for each of these reductions whether they are applicable. The rules are applied in a predefined order. To identify applicable reductions, we employ exhaustive search, with added pruning for the different rules. If one reduction is successfully applied, then the process of testing possible reductions starts from the beginning (according to this order). All reductions, that have been tested already are now only checked in areas of the graph that changed. If no more reductions can be applied, we obtained the reduced graph and continue with the next part of Algorithm 1. The order in which reductions are applied has an effect on the weight offset and the size of the resulting reduced graph, as well as on the time needed for the computation. We give a detailed analysis in Section 5.1.

If the resulting reduced instance is small enough, i.e. its number of vertices is less than the threshold n_K, we try to solve the instance exactly using Struction by Gellner et al. [20] within a time limit t_{exact}. If it is not solved optimally within this time, the computed solution is added to the population and we continue.

4.2 Memetic Algorithm

After ExactReduce, we apply the Evolve routine which is described in Algorithm 2 on the reduced graph $\mathcal{K}$. It starts by generating an initial *population* of size $|\mathcal{P}|$ which we then evolve over several generational cycles (rounds). For the evolution of the population two *individuals* from the population are selected and combined to create an *offspring*. We also apply a mutation operation to this new solution by forcing new vertices into the solution and removing neighboring solution vertices. To keep the population size constant and still add a new offspring to the solution, we look for fit replacements. In this process, we search for individuals in the population, which have smaller weights than the new offspring. Among those, we look for the most similar solution by computing the intersection size of the new and existing individuals. We also added the possibility of forcing

Algorithm 2 High Level Structure of $\textsc{Evolve}(G, \mathcal{P})$

```
input graph G = (V, E), current population P
procedure Evolve(G, P)
  while stopping criterion not fulfilled
    randomly chose a combine operation COMBINE
    k = number of individuals needed for COMBINE
    IS ← ∅ // set of individuals
    IS ← TOURNAMENTSELECT(P)
    OS ← ∅ // set of offspring
    OS ← COMBINE(IS)
    if mutate with probability 10%
      OS ← MUTATE({OS})
    if suitable replacement (different criteria)
      P ← REPLACE(P, O)
return P
```

individuals into the population if it has not changed over a certain number of iterations, as well as rejecting the offspring if the solution with the smallest weight is still better than the new offspring. Note that the size $|\mathcal{P}|$ of the population does not change during this process. Additionally, at any time each individual of our population is an independent set. In the last step of the memetic algorithm we improve the solution by the HILS algorithm [40]. The stopping criterion for the memetic procedure is either a specified number of unsuccessful combine operations or a time limit. In the following we discuss each of these steps in detail. We start with introducing the computation of the initial solution in Section 4.2.1 and then explain the combine operations for the evolutionary process in Section 4.2.2 as well as the mutation operation in Section 4.2.3.

4.2.1 Initial Solutions

At the start of our memetic algorithm, we create an initial population of size $|\mathcal{P}|$. To diversify as much as possible, this population contains solutions computed in six different ways, which we choose uniformly at random to create an individual. Before applying the strategies we permute the order of the nodes such that different solutions are obtained for the same strategy by different tie breaking.

RandomMWIS. The first approach works by starting with an empty solution and adding free vertices uniformly at random until the solution is maximal.

GreedyWeightMWIS. For the *GreedyWeightMWIS* strategy, we start with an empty solution. This is extended to a maximal independent set by adding free vertices ordered by their weight. Starting with the largest weight, we include this vertex and exclude all its neighbors until all vertices are labeled either included or excluded.

GreedyDegreeMWIS. Via this greedy approach, we create initial solutions by successively choosing the next free vertex with the smallest residual degree. Each time a vertex is included, we label the neighboring vertices to be excluded.

GreedyWeightVC. In contrast to the previous approaches, here the vertex cover problem, the complementary problem to the independent set problem, is utilized. Therefore, an empty solution is extended by vertices of the smallest weight until a vertex cover is computed. As soon as the algorithm terminated, we compute the complement and have an initial solution to the MWIS problem.

GreedyDegreeVC. As in *GreedyWeightVC* the complementary vertex cover problem is solved. However, for this approach, we choose those vertices to include in the solution, which cover the maximum number of currently uncovered edges.

Struction. We also add the possibility to compute an initial solution via the Struction algorithm by Gellner et al. [20]. We set a time limit of 60 seconds and use the configuration *CyclicFast*. If we also use Struction to compute initial solutions, we call the algorithm configuration M^{2}WIS + S. Note that if the algorithm does not solve the instance within the time limit, it returns a non-optimal solution.

4.2.2 Combine Operations

The common idea of our combine operations which are inspired by the work of Lamm et al. [31], is to combine whole blocks of independent set vertices. To construct those blocks, we use the graph partitioning framework KaHIP [47] which computes partitions of the graph $V = V_1 \cup ... \cup V_n$. For $j = 1, ..., n$ the solution blocks $\mathcal{I}_j$ are defined by $\mathcal{I}_j = \mathcal{I} \cap V_j$. We created different offspring by using the following combine operations on those solution blocks.

The parents for the first two combine operations are chosen by two runs of the tournament selection [39], where the fittest individual i.e. the solution with highest weight gets selected out of two random individuals from the population. Then we perform one of the combine operations outlined below and finally, after the combine operation, we use the HILS algorithm [40] to improve the computed offspring.

Vertex Separator Combination. The first operator works with a vertex separator $V = V_1 \cup V_2 \cup S$. We use a vertex separator to be able to exchange whole blocks of solutions without violating the independent set property. This can be done because no vertices belonging to different blocks are adjacent to one another. Neighboring vertices would either be part of the same block or one of them has to belong to the separator S. By this property the combination of those blocks will always result in a valid solution to the independent set problem. The two individuals selected by the tournament $\mathcal{I}_1$ and $\mathcal{I}_2$ are split up according to these partitions and are then combined to generate two offspring $O_1 = (V_1 \cap \mathcal{I}_1) \cup (V_2 \cap \mathcal{I}_2)$ and $O_2 = (V_1 \cap \mathcal{I}_2) \cup (V_2 \cap \mathcal{I}_1)$. After that we add as many free vertices greedily by weight until the solution is maximal, we get a local optimum via one iteration of the weighted local search. See Figure 3 for an illustration.

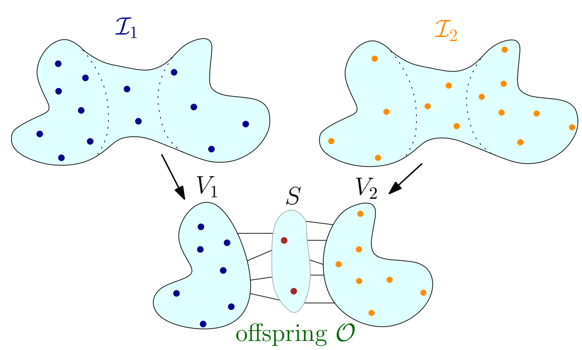

Figure 3: The vertex separator combine operation to create an offspring $\mathcal{O}$ out of two individuals $\mathcal{I}_1$ and $\mathcal{I}_2$.

Multi-way Vertex Separator Combination. We extended the previous described operator to the multi-way vertex separator, where multiple solutions can be used and combined. Therefore, we compute a k-vertex separator $V = V_1 \cup ... \cup V_k \cup S$ and select k individuals. Then for every pair of partition V_i and individual $\mathcal{I}_j$ for $i, j \in \{1, ..., k\}$ a score is computed. This score is defined by $\sum_{v \in V_i \cap I_j} \omega(v)$. We start with the pair resulting in the highest score pair and then select pairs decreasingly. Once an individual or partition block is selected, we do not use it again. In contrast to the previous operator, this combination only results in one offspring. We then maximize this offspring and compute a local maximum.

Edge Separator Combination. For this operator we exploit the duality to the weighted vertex cover problem. Starting with a partition $V = V_1 \cup V_2$ the operator computes temporary offspring for the weighted vertex cover problem. Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be the individuals selected by the tournament rule. Let $C_i = V \setminus \mathcal{I}_i$ be the solution to the weighted vertex cover problem for $i \in \{1, 2\}$. The new offspring are $O_1 = (V_1 \cap C_1) \cup (V_2 \cap C_2)$ and $O_2 = (V_1 \cap C_2) \cup (V_2 \cap C_1)$. However, these offspring can contain some non-covered edges, which are a subset from the cut edges between the two partitions. The graph induced by the non-covered cut edges is bipartite. In this graph we compute a minimum-weight vertex cover using maximum flows.

Multi-way Edge Separator Combination. Similar to the vertex separator also the edge separator can be extended to use multiple solutions. Therefore, a k-way-partition $V = V_1 \cup ... \cup V_k$ is computed. Equivalent to the multi-way vertex separator, we also select k individuals and compute a score for each pair V_j and $\mathcal{I}_i$. For the scoring function, the complement of an independent set inside the given block is used to sum up the weights of the vertices of the vertex cover in this block. For the offspring computation, each block is combined with the individual with the lowest score. As in the basic edge separator combine operator there can be edges in the cut that are not covered. Since the induced graph here is not bipartite we handle this problem using a simple greedy strategy. Afterwards the solution is transformed to get the offspring for the independent set individuals.

4.2.3 Mutation Operation

After each combine operation, a mutation operator can perturb the created offspring. This is done by forcing new vertices into the solution and removing the adjacent vertices to satisfy the independent set property. Those vertices are selected at random among all non-solution nodes in the graph. Afterwards we improve the solution using the HILS algorithm.

4.3 Heuristic Reductions and Recursion

After the memetic algorithm stops, we use a heuristic data reduction to open up the reduction space (and afterwards the next round of exact data reductions begins). We implemented different strategies to select vertices that we put into the solution. In each strategy, vertices are ordered by a rating function. The algorithm inserts a fraction (from only one vertex to 100%) of the vertices selected by the different strategies. We now explain the different selection strategies.

Vertex Selection by Weight. The first rating function is based on the weight $\omega(v)$ of a vertex v (higher is better). The intuition here is that by adding a vertex, we want to increase the weight of our solution as much as possible. More precisely, the fittest individual from the population evolved by the memetic algorithm is selected. The fitness of an individual is defined as the solution weight. From this individual, we select the x vertices from the independent set that have the highest weight and add them to our solution. Since we only consider vertices from one individual, x can be freely chosen without violating the independent set property of our solution. For example, we can choose to only add the highest weight vertex or select a fraction of those solution vertices.

Vertex Selection by Degree. Similar to the previous vertex selection strategy, we choose the fittest individual from which we add vertices to our solution. Here, the vertices are rated by their degree $\deg(v)$ (smaller is better). The intuition here is that adding vertices with a small degree to our solution will not remove too many other vertices from the graph that could be considered later.

Vertex Selection by Weight/Degree. For this selection strategy, we rate the vertices v of the fittest individual by the fraction $\frac{\omega(v)}{\deg(v)}$ (higher is better). This way we combine both of the two previous ratings.

Hybrid Vertex Selection. In the hybrid case, the solution vertices $v \in V$ are rated by the weight difference between a vertex and its neighbors $\omega(v) - \sum_{u \in N(v)} \omega(u)$ (higher is better). This value describes the minimum gain in solution weight we can achieve by adding the vertex v to the solution. Note that Gu et al. [23] proposed this rule for their algorithm. The key difference here is that Gu et al. [23] use this function on all vertices, while our algorithm only considers solution vertices of the fittest solution of the memetic algorithm.

Vertex Selection by Solution Participation. In contrast to the previous strategies, this strategy considers the *whole* population. Moreover, here we consider *each vertex* in the graph. We check the population and assign each vertex a value according to the number of times it is part of a solution. The maximum number a vertex can achieve is therefore bounded by the population size $|\mathcal{P}|$. We can include all vertices that are in each individual, i.e. vertices with a score equal to $|\mathcal{P}|$, as well as exclude all vertices that are in no solution at all, i.e. vertices with a zero score. Note that the total number of vertices selected with this strategy differs from the previous strategies, where we consider all the vertices from the best solution in the population. The vertices selected by solution participation are only a subset of these. This process is similar to the Merge Search introduced by Kenny et al. [28].

5 Experimental Evaluation

Methodology. We implemented our algorithm using C++11. The code is compiled using g++ version 12.2 and full optimizations turned on (-O3). We compare our algorithm against the struction algorithm by Gellner et al. [20] and the (more recent) algorithm HtWIS by Gu et al. [23]. We also compare the results with the branch-and-reduce algorithm by Lamm et al. [32], as well as the HILS algorithm by Nogueira et al. [40]. In most cases HILS outperforms DynWVC1 and DynWVC2 [32]. Hence, we omit comparisons to DynWVC1 and DynWVC2. We run each configuration with four different seeds and a time limit of ten hours and report the mean results. For all algorithms we always report the time when the best solution was found within this time limit. The only algorithms which might not use the whole ten hour time is the exact algorithm, when found and proven an optimal solution and the algorithm HtWIS. Note that this algorithm is not using any randomness, which would enable us to run it multiple times with different seeds for using the whole 10 hours, nor does the algorithm have any other parameters that would increase the solution quality by spending more time. If a solver exceeded a memory threshold of 100 GB during a time limit of ten hours for an instance we note this with a dash. In general, our algorithm does not test the time limit in the ExactReduce routine of m^{2}wis or during the calculation of the separator and partition pool. Hence, if the 10h mark is reached during these steps, the time limit can be exceeded. We used one core of a machine equipped with a AMD EPYC 7702P (64 cores) processor and 1 TB RAM running Ubuntu 20.04.1. We used the fast configuration of the KaHIP graph partitioning package [48, 49] for the computation of the graph partitions and vertex separators. We also present extensive experiments regarding the impact of reduction ordering. We conclude that the order in which we introduce Reductions 1 to 11 is already robust and hence use it for the remaining experiments.

Parameter Configuration. Similar to Lamm et al. [31], we set the population size $|\mathcal{P}|$ to 250, the size of the partition and separator pool to 10 and the mutation rate to 10%. Local search is limited to 15 000 iterations. Finally, for the multi-way combine operations, we bound the number of blocks used by 64. For the state of the art experiments, we set the parameters as discussed in Section 5.2 and 5.3.

Data Sets. The set of instances for the experiments is built with graphs from different sources. We use all the instances used by Gellner et al. [20] and Gu et al. [23]. Our set consists of large social networks from the Stanford Large Network Dataset Repository (snap) [34]. Additionally, we added real-world graphs from OpenStreetMaps (osm) [1, 9, 11]. Furthermore, as in Gu et al. [23] we took the same 6 graphs from the SuiteSparse Matrix Collection (ssmc) [2, 14] where weights correspond to population data. Each weight was increased by one, to avoid a large number of nodes assigned with zero weight. Additionally, we used instances from dual graphs of well-known triangle meshes (mesh) [46], as well as 3d meshes derived from simulations using the finite element method (fe) [51]. For unweighted graphs, we assigned each vertex a random weight that is uniformly distributed in the interval [1, 200]. We also tested our algorithms on the kernels of osm instances as used by Dong et al. [15]. We do not compare our algorithm on the VR instances contained therein, as data reductions do not work on those instances [15] and the reduced graphs are too large to be sufficiently explored by our memetic algorithm. We list all graphs in Table 15.

5.1 Experiments on Reduction Ordering

Different orderings of applying data reductions yield different sizes of the reduced graph. Additionally, the ordering effects the running time of the ExactReduce routine from Algorithm 1. This effect has been described for example by Figiel et al. [18]. We now perform experiments to evaluate the impact different orderings may have. Here, we run the ExactReduce routine, i.e. we apply the reductions in a given ordering exhaustively, and report the results.

Baseline: Our starting point is an intuitive ordering, which we constructed by simplicity of the reductions, from simplest to most complex. Hereby we orientate us towards the ordering chosen by Akiba and Iwata [4]. This initial ordering is precisely the order in which we introduce the reductions from Reduction 1 to Reduction 12. In the following experiments, we use this ordering as a *baseline* to compare against.

5.1.1 Orderings Based on Impact of Single Reductions

The space of possible orderings of data reductions is very large. We start our evaluation by examining the impact of disabling single reductions in our baseline, i.e. we run our baseline reductions and then build a set of reductions where exactly one data reduction of the baseline is disabled. The ordering of the remaining data reductions remains the same. The time to apply all reductions exhaustively using our baseline ordering is denoted as t_{all} and the size of the reduced instance by those reductions is denoted as ω_{all}. Then, we create different data reduction orderings from the baseline in which a single data reduction is disabled. For each of the available reductions r, we get a new time $t_{all\setminus r}$ and solution weight $\omega_{all\setminus r}$ which corresponds to running all reductions except r of the baseline (and in its order). Based on these values, we derive three orderings, a time based ordering, a size based ordering and a combination of both.

Table 1: Comparing geometric mean of running times and reduced graph sizes for different orderings, relative to the initial ordering, where $|\mathcal{K}|$ is the number of vertices in the reduced graph. Additionally, we count how often an ordering found the smallest reduced graph in comparison to the other orderings (# best).

Ordering	$t/t_{initial}$	$\|K\|/\|K\|_{initial}$	# best
initial	1,00	1,000	180/207
time	1,06	1,021	161/207
size	1,46	1,074	164/207
time&size	1,60	0,998	191/207

Time-based Ordering. For the *time-based ordering* we rearranged the reductions such that the mean $\bar{t}_{all\setminus r}$ is decreasing. The intuition here is if removing a reduction from the baseline yields an ordering that has an excessive running time, then this reduction is important for running time and should be applied before a reduction that has a smaller impact. Reductions with only small effects, where the mean solution quality $\overline{\omega_{all\setminus r}}$ is equal to the mean solution quality for $\overline{\omega_{all}}$, are disabled to further reduce the running time. This results in the ordering (4, 6, 8, 4 (case 3), 2, 7, 1, 5, 3, 10, 12).

Size-based Ordering. For the *size-based ordering* the reductions are reordered in decreasing order according to the mean value $\overline{\omega}_{all\setminus r}$ over all graphs. The intuition here is that if $\overline{\omega}_{all\setminus r}$ is large, then not using r has a large impact on the size of reduced graph and hence should be applied before a reduction that has a smaller impact. The resulting ordering is (4, 6, 10, 8, 4 (case 3), 12, 5, 3, 1, 2,7, 9, 11).

Time and Size-based Ordering. Here we use a combination with $x_{all\setminus r} = \bar{t}_{all\setminus r} + 10\overline{\omega}_{all\setminus r}$ decreasingly to order reductions. We use a factor of 10 here, since solution quality is typically more important for applications than running time. This results in the following ordering of reductions (4, 6, 10, 8, 5, 4 (case 3), 12, 3, 1, 2,7, 9, 11).

Discussion. The results for the previous orderings are presented in Table 1. We note that different orderings do not yield significant differences compared to the initial ordering. We can observe, that the *time* as well as the *size* ordering is not able to improve neither the kernel size, nor the computation time. The ordering *time&size* can compute smaller kernel sizes, at an expense of additional 60% of running time. With this we are able to find 191 out of 207 smallest kernels. However, the improvement in the geometric mean kernel size compared to the initial ordering is less than 0.2%. The experiments show, that the initial ordering already yields a good trade of for running time and kernel size. When examining the positions for the reductions, we note that there are some reductions that remain at approximately the same position meaning they are either very important for solution size and quality (or the opposite). For example Reductions 4 and 6 are applied at the beginning, whereas for example Reduction 9 is applied towards the end or removed. On the other hand there also are reductions that are on completely different positions, e.g. Reduction 10. In general, our experiments show that the reduction order has an effect on the size of the reduced instance and especially on the running time. We conclude that the initial ordering (baseline) is already robust w.r.t. the orderings considered in this section.

5.1.2 Orderings Based on Impact of Groups of Reductions

We divide the reductions into three groups of roughly similar complexity. The first group contains Reductions 1 and 2, the second group consists of reductions for vertices of degree two, which are Reductions 3 and 4. The third group contains all remaining reductions listed in Section 4.1.

Permutation of Ordering of Reductions in First and Second Group. We now examine all permutations in the first and second group. The permutation in the ordering only takes place inside the groups. The groups themselves always stay in a fixed order, i.e. Reductions 1 and 2 will always be the first two reductions applied. Reductions of the third group are applied as in the initial ordering. Overall, our experiments show that the order of the reductions in the first two groups has a negligible effect on running time and quality of a solution. Thus, for the remaining experiments we use the initial ordering.

Permutation of Ordering of Reductions in Third Group. We now examine permutations in the third group of reductions. We apply additional restrictions to reduce the number of permutations. We apply Reduction 11 always last, and Reduction 6 is always followed by Reduction 7. The best performing permutation is (1, 2, 3, 4, 4 (case 3), 5, 8, 10, 9, 6, 7, 11). The geometric mean weight improvement is $w/w_{initial} = 1{,}0023$ and the geometric mean time compared to the initial ordering is $t/t_{initial} = 2{,}9$.

Conclusion. Overall, there are some orderings that perform better than the initial ordering by complexity, however, these improvements are only on a few instances (and result in significantly higher running time). In most cases all orderings yielded the similar results. Among those, the initial ordering remains one of the fastest. We conclude that the initial ordering presents a very stable reduction ordering. Hence, we use it for the remaining experiments. For some graphs, it might be worth trying multiple runs of algorithms using one of the other orderings we presented in this section as well.

5.2 Heuristic Data Reduction Rules

For the heuristic data reduction rules we perform multiple experiments on a subset of our dataset containing 15 graphs, marked in Table 15. For this we took three large graphs from each class to evaluate the influence on performance of our parameters. For each instance we use four different seeds and a time limit of one hour.

We start with comparing the different vertex selection strategies presented in Section 4.3. Each strategy is evaluated for different fractions. Table 2 summarizes our results. For each configuration we show the geometric mean quality as well as the geometric mean running time over the subset of 15 graphs. Note that the number of selectable vertices is different between *solution participation* and the other strategies. For *solution participation*, where vertices are only considered if they are in each or none solution within the *whole* population the set of vertices selected is only a subset of the vertices selectable by the other strategies. Adding 100% does result in the same solution for these strategies, since the best solution is always included completely. Furthermore, this explains the larger speedup for these other strategies compared to *solution participation* when the fraction is increased.

For all strategies we can reduce the mean time. The smallest speedup is observed for *solution participation*, as explained in the previous paragraph. Using the other strategies we can achieve a speedup of 10 by increasing the fraction parameter. The quality using *solution participation* is

Table 2: M^{2}WIS geometric mean solution weight ω and time t (in seconds) required to compute ω for different vertex selection strategies and fractions f. Note that, $f = 1$ is adding one vertex, while the other rows refer to percentages of the solution. The best result among all configurations are marked **bold**.

	t	ω	t	ω	t	ω	t	ω	t	ω
f	degree		hybrid		solution participation		weight		weight/degree	
1	781,48	4 052 070	753,95	4 052 039	654,53	4 054 053	785,94	4 052 026	610,46	4 052 120
5%	405,97	4 052 805	549,90	4 053 186	798,08	4 052 516	362,88	4 051 708	345,88	4 052 895
25%	208,75	4 052 056	252,43	4 052 409	685,24	4 053 765	135,34	4 051 263	199,10	4 051 819
75%	69,95	4 050 596	100,64	4 050 646	558,72	4 054 196	68,99	4 050 476	69,75	4 050 397
100%	59,60	4 050 453	60,17	4 050 453	515,11	**4 054 303**	59,82	4 050 453	60,41	4 050 453

increasing with increasing fraction until $f = 100\%$, while the other strategies perform best with $f = 5\%$. When further increasing the amount of vertices added in these strategies, the solution quality gets worse. The difference between the best and the worst mean result is around 0,1%. The overall best mean solution quality is achieved by using the *solution participation* with adding 100% of the vertices possible. We set this configuration for the next set of our experiments.

5.3 Solving Small Reduced Graphs Exactly

With this experiment we examine whether it is beneficial at some point to solve the reduced instance exactly. Therefore, we introduced two new parameters. First, we add a threshold n_K determining when to start solving the reduced instance exactly. When the number of vertices in the reduced graph K is smaller than n_K, we apply STRUCTION. The second parameter is a time limit t_{exact} which restricts this exact solver. If the instance is not solved within this time limit, we continue with the algorithm M^{2}WIS.

As in the previous section, we use the same subset of 15 graphs, marked in Table 15 and for each instance we use four different seeds and an overall time limit of one hour. We present the summary of the results using different n_K from 0 to 15 000 and t_{exact} from 10 seconds to no time limit for M^{2}WIS in Table 3 and for M^{2}WIS + S in Table 4. We report the time when the best solution was found. Often the exact solver finds a good solution very fast, but is then not able to improve the solution. This is why for increasing n_K, we can sometimes see an decrease in the reported time.

In both tables the solution quality does not differ much between the different configurations. These differences are all less than 0,01%. However, we can see in Table 3 that for M^{2}WIS $n_K =$

Table 3: M^{2}WIS geometric mean solution weight ω and time t (in seconds) required to compute ω for different thresholds n_K and time limits t_{exact} to solve reduced instances exactly. The best result among all configurations are marked **bold**.

	t	ω	t	ω	t	ω	t	ω
n_K	$t_{exact} = 10$ [s]		$t_{exact} = 100$ [s]		$t_{exact} = 1\,000$ [s]		no limit	
0	515,11	4 054 303	515,11	4 054 303	515,11	4 054 303	515,11	4 054 303
100	519,69	4 054 164	513,77	4 054 313	535,64	4 054 080	533,28	4 054 208
500	509,72	4 054 191	525,53	4 054 337	517,47	4 054 132	532,67	4 054 066
1 000	480,82	4 054 037	501,77	4 054 305	522,39	4 054 213	505,01	4 054 234
5 000	188,45	4 054 285	196,32	4 054 114	205,92	4 053 974	204,50	4 054 138
10 000	79,43	4 054 144	85,13	**4 054 367**	99,65	4 054 057	114,17	4 054 068
15 000	34,22	4 054 249	37,33	4 054 180	45,45	4 054 229	48,74	4 054 052

Table 4: M^{2}WIS + S geometric mean solution weight ω and time t (in seconds) required to compute ω for different thresholds n_K and time limits t_{exact} to solve reduced instances exactly. The best result among all configurations are marked **bold**.

	t	ω	t	ω	t	ω	t	ω
n_K	$t_{exact} = 10$ [s]		$t_{exact} = 100$ [s]		$t_{exact} = 1\,000$ [s]		no limit	
0	15,58	4 054 480	15,58	4 054 480	15,58	4 054 480	15,58	4 054 480
100	15,37	4 054 599	15,70	4 054 336	15,55	**4 054 666**	15,90	4 054 579
500	15,31	4 054 530	15,90	4 054 537	15,52	4 054 546	15,60	4 054 439
1 000	15,52	4 054 467	15,52	4 054 520	15,48	4 054 386	15,34	4 054 445
5 000	13,21	4 054 461	12,92	4 054 571	14,18	4 054 628	15,69	4 054 481
10 000	13,23	4 054 538	14,39	4 054 409	18,25	4 054 463	20,90	4 054 377
15 000	13,04	4 054 569	14,04	4 054 492	17,85	4 054 543	20,58	4 054 520

15 000 the mean time is reduced up to a factor of 15 for all time limits t_{exact} tested. The threshold $n_K = 10\,000$ yields the best result with a time limit of $t_{exact} = 100$. For M^{2}WIS + S in Table 4 we see that the best parameter configuration is $n_K = 100$ and t_{exact} =1 000 seconds. With this configuration we also achieved the overall best result when comparing to M^{2}WIS.

For the comparison against the state of the art, we use the following configurations: We set the parameters for M^{2}WIS to $n_K = 15\,000$ and $t_{exact} = 100$ seconds. We chose this configuration, since compared to the parameters yielding the best solution quality this configuration is only 0,00002% worse regarding solution quality, however, it is 2,5 times faster. For M^{2}WIS + S we chose the configuration yielding the best solution quality, which is $n_K = 100$ and $t_{exact} = 1\,000$ seconds.

5.4 Limiting the Time for Evolve

Additionally to limiting the number of rounds of the evolution cycles, we also restrict the time used within the EVOLVE procedure. This is especially important for very complex instances where EVOLVE can take up all the time. We test different limits for *evotime* starting with 450 seconds up to 4 hours on the same 15 graphs, marked in Table 15. For each instance we use four different seeds and, in contrast to the previous experiments, an overall time limit of 10 hours. We present the geometric mean over the results for both M^{2}WIS and M^{2}WIS + S in Table 5. Generally, we see that this parameter has an higher impact on the solution quality compared to the previous parameters tested. For M^{2}WIS + S we set *evotime* $= 900$, and for M^{2}WIS we set *evotime* $= 1\,800$. These are the configurations for which our algorithms yielded the best geometric mean solution quality within the test set.

Table 5: Geometric mean solution weight ω and time t (in seconds) required to compute ω for different time limits *evotime* in seconds) for the EVOLVE procedure, see Algorithm 2. The best result among all configurations are marked **bold**.

	t	ω	t	ω
evotime	M^{2}WIS + S		M^{2}WIS	
450	15,36	4 055 174	37,55	4 055 028
900	15,85	**4 055 342**	46,90	4 055 206
1 800	17,59	4 055 150	59,16	**4 055 253**
3 600	18,11	4 054 872	66,08	4 055 179
7 200	18,22	4 054 202	75,86	4 054 664
14 400	18,70	4 053 447	81,22	4 053 511

Table 6: Average solution weight ω and time t (in seconds) required to compute ω for a representative sample of our instances comparing the best performing competitors. The best solutions among all algorithms are marked **bold**. Rows are colored gray if branch reduce or struction are optimal. See Tables 9 to 13 for more details.

	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω
finEl	branch reduce		HtWIS		HILS		struction		M^{2}WIS + S		M^{2}WIS	
body	-	-	0,04	1 645 650	1 259,67	1 678 510	-	-	112,85	**1 680 182**	901,26	1 680 179
ocean	4,88	**7 248 581**	0,07	6 803 672	11 142,43	7 075 329	-	-	5,20	**7 248 581**	5,01	**7 248 581**
pwt	-	-	0,03	1 153 600	761,52	1 175 437	-	-	3 846,33	**1 178 734**	5 131,59	1 178 583
mesh	branch reduce		HtWIS		HILS		struction		M^{2}WIS + S		M^{2}WIS	
blob	0,14	**855 547**	0,01	854 484	260,10	854 803	0,02	**855 547**	0,04	**855 547**	0,03	**855 547**
dragon	2,90	**7 956 530**	0,04	7 950 526	9 026,60	7 947 535	0,17	**7 956 530**	0,31	**7 956 530**	0,33	**7 956 530**
ecat	9,18	**36 650 298**	0,50	36 606 394	36 000,05	36 562 652	1,92	**36 650 298**	2,83	**36 650 298**	3,01	**36 650 298**
osm	branch reduce		HtWIS		HILS		struction		M^{2}WIS + S		M^{2}WIS	
flor.-3	1 724,45	**237 333**	0,13	234 218	216,24	**237 333**	1,33	**237 333**	11,20	**237 333**	9,57	**237 333**
mas.-3	-	-	1,77	144 381	355,93	**145 866**	-	-	77,14	**145 866**	104,87	**145 866**
utah-3	239,50	**98 847**	0,04	97 754	72,21	**98 847**	0,08	**98 847**	2,34	**98 847**	1,94	**98 847**
snap	branch reduce		HtWIS		HILS		struction		M^{2}WIS + S		M^{2}WIS	
as-skit.	-	-	1,04	124 141 373	36 000,25	123 994 141	-	-	3 422,79	**124 157 729**	621,62	**124 157 729**
ca-Gr.	<0,01	**287 919**	<0,01	287 850	144,49	**287 919**	<0,01	**287 919**	<0,01	**287 919**	<0,01	**287 919**
web-BS.	36 000,12	43 891 206	9,94	43 889 843	36 000,10	43 888 267	6,52	**43 907 482**	8,75	**43 907 482**	9,32	**43 907 482**
ssmc	branch reduce		HtWIS		HILS		struction		M^{2}WIS + S		M^{2}WIS	
ga2010	36 000,10	4 644 324	0,47	4 639 891	29 522,41	4 642 807	0,62	**4 644 417**	1,64	**4 644 417**	1,85	**4 644 417**
nh2010	36 000,00	581 637	0,03	587 059	2 163,80	588 797	0,11	**588 996**	0,47	**588 996**	0,47	**588 996**
ri2010	36 000,00	447 427	0,02	457 108	782,49	458 489	0,09	**459 275**	0,62	**459 275**	0,49	**459 275**
overall	branch reduce		HtWIS		HILS		struction		M^{2}WIS + S		M^{2}WIS	
# best		175/207		97/207		154/207		188/207		204/207		198/207
gmean ω		-		153 434		154 355		-		154 430		154 430
gmean t		-		<0,01		59,18		-		0,03		0,03

5.5 Comparison against the State of the Art

We now compare our algorithm M^{2}WIS against a range of algorithms: HtWIS by Gu et al. [23], both struction configurations by Gellner et al. [20] where we always report the better of the two results in the column named struction, the branch-and-reduce solver by Lamm et al. [32], and HILS by Nogueira et al. [40]. Additionally, we test against the vertex cover algorithms NuMWVC by Li et al. [36] and GNN VC by Langedal et al. [33]. Note that we used the provided GNN VC model, which was trained on instances from the SuiteSparse Matrix Collection [2]. We also include the variant of our algorithm, called M^{2}WIS + S, using Struction from Gellner et al. [20] with a time limit of 60 seconds to compute individuals for the initial population. We present a representative sample of our full experiments in Table 6. In the last part we give a summary over all instances. This consists of the number of instances solved best, the geometric mean time and solution quality respectively. Detailed per-instance results are presented in the Tables 9 to 13 in the Appendix.

Overall, we see that M^{2}WIS + S has the largest number of best solutions for our full set of 207 instances. In particular, it is able to compute the best solution for all but three instances, i.e. *kentucky-3*, *hawaii-3* (osm) and *soc-p.rel.* (snap). In these three cases M^{2}WIS + S was outperformed by our other variant M^{2}WIS. Additionally, M^{2}WIS + S is able to compute the best solutions for all graphs in the graph classes finiteElement, mesh and ssmc. Finally, for all but 18 of these 207 instances, M^{2}WIS + S finds the best solution in less than 100 seconds. Our algorithm M^{2}WIS is still able to compute the best solution for 198 instances, including *hawaii-3*, *kentucky-3* and *soc-p.rel.*. The geometric mean running time over all instances of both of our variants is the same.

When looking at the running times, we see that HtWIS achieves the smallest geometric mean running time. However, the quality is less or equal to the results of M^{2}WIS and M^{2}WIS + S on *all* of the tested instances, with multiple instances having a significant difference in weight–larger than 10 000. If running time is crucial, the competitor HtWIS is noteworthy. However, our algorithm also computes high quality initial solutions which are then enhanced over time. M^{2}WIS + S for example needs a geometric mean time of 0,0046 seconds to compute initial solutions which are on average 0,3% better than the solutions computed by HtWIS which needs a geometric mean time of 0,039 seconds. The best improvement already after the initial computations is found for *greenland-AM3*, where we need 3 times as long as HtWIS while getting an improvement of more than 12% over the result of HtWIS. The running time achieved by the struction-based algorithms is also to be noted. These are for example the second fastest on ssmc instances, see Table 10. The struction algorithms solve 47 instances faster than all competitors. However, these only find 188 best solutions overall. The branch reduce solver is able to compute 175 best solutions within the limitations of the experiment. It works especially good on *snap* and *mesh* instances, while it is not able to solve any of the *ssmc* instances optimally. When looking at the performance of the four competing heuristic algorithms, we see that HILS can compute overall the most best solutions, followed by GNN VC. When comparing HILS and GNN VC directly, the comparison is highly dependent on the graph class. On the *mesh* and *snap* instances for example, GNN VC beat HILS on every instance regarding solution quality. But since we have 148 *osm* instances, where GNN VC was not trained on, overall HILS has the larger number of best solved instances. Regarding running time, HILS needs on average almost 6 times as long as GNN VC. The overall fastest competitor HtWIS computed on 97 instances the best solution, while NuMWVC never computed a best solution within our experimental setup.

In terms of memory requirement, when the struction variants are able to solve the instance very fast, memory usage is usually below 1 GB and also a bit smaller than the the memory required by our algorithm. For more difficult instances, as for example *body* (finiteElement) where M^{2}WIS + S and M^{2}WIS have a memory usage of less than 1 GB, the struction algorithm requires more than 100 GB. This high memory consumption compared to our solver can be explained since, particularly for challenging instances, the struction algorithm reaches a memory-intensive branching phase.

5.6 Comparison against the State of the Art on Reduced Instances

We now compare the same algorithms on the reduced instances computed with the set of reduction rules presented in Section 4.1. In this experiment, we show the impact of our algorithm apart from the reduction rules, which can be used as a preprocessing step in general. Table 14 shows the detailed per instance results. In Table 7, we show the same sample of the results. Especially for the branch and reduce algorithm by Lamm et al. [32] we can see a lot of improvement over the results in Table 6. With additionally using the reductions, we are able to solve more of the ssmc instances optimally as well reducing the memory consumption. For example *massachusetts-3* can now be solved without reaching the memory threshold. The other algorithms also benefit from using the reductions, which can be seen in higher solution qualities and less running time. Still, our two algorithm variants perform best also on the set of reduced instances. There is only one reduced snap instance (*soc-pokec-relationships*), see Table 14, where our algorithms yield a smaller solution. On this instance, they are outperformed by HILS, but computed the second and third best solution.

Table 7: Average solution weight ω and time t (in seconds) required to compute ω for a representative sample of our instances comparing the best performing competitors. The best solutions among all algorithms are marked **bold**. Rows are colored gray if branch reduce or struction are optimal. See Table 14 for more details.

	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω
redFinEl	branch reduce		HtWIS		HILS		struction		M^{2}WIS + s		M^{2}WIS	
body	-	-	0,01	222 320	74,81	224 550	-	-	60,30	**224 744**	100,23	**224 744**
pwt	-	-	0,07	1 019 262	655,57	1 034 418	-	-	1 391,40	**1 038 139**	4 767,83	1 038 046
redMesh	branch reduce		HtWIS		HILS		struction		M^{2}WIS + s		M^{2}WIS	
ecat	0,02	**31 254**	<0,01	**30 789**	8,17	**31 254**	<0,01	**31 254**	0,03	**31 254**	0,03	**31 254**
redOsm	branch reduce		HtWIS		HILS		struction		M^{2}WIS + s		M^{2}WIS	
florida-3	1 714,07	**25 992**	0,11	23 471	65,56	**25 992**	1,34	**25 992**	8,20	**25 992**	7,80	**25 992**
mas.-3	36 000,45	14 610	1,79	13 285	247,66	**14 757**	-	-	74,22	**14 757**	102,60	**14 757**
utah-3	200,33	**16 090**	0,03	15 186	41,05	**16 090**	0,07	**16 090**	1,77	**16 090**	1,64	**16 090**
redSnap	branch reduce		HtWIS		HILS		struction		M^{2}WIS + s		M^{2}WIS	
as-skitter	-	-	0,04	134 500	45,35	135 976	-	-	1 904,18	**135 998**	287,87	**135 998**
web-BerkStan	28,05	**135 172**	<0,01	133 309	36,21	135 156	0,04	**135 172**	0,17	**135 172**	0,15	**135 172**
redSsmc	branch reduce		HtWIS		HILS		struction		M^{2}WIS + s		M^{2}WIS	
ga2010	159,38	**76 316**	0,01	76 297	59,10	76 297	0,07	**76 316**	0,34	**76 316**	0,33	**76 316**
nh2010	2,28	**26 770**	<0,01	26 477	17,20	26 768	0,01	**26 770**	0,07	**26 770**	0,05	**26 770**
ri2010	6 477,22	**66 963**	<0,01	65 979	32,71	66 960	0,02	**66 963**	0,15	**66 963**	0,12	**66 963**
overall	branch reduce		HtWIS		HILS		struction		M^{2}WIS + s		M^{2}WIS	
# best		67/93		14/93		75/93		75/93		88/93		88/93
gmean ω		-		15 964		16 683		-		16 688		16 688
gmean t		-		0,01		25,01		-		0,60		0,66

5.6.1 Comparison to METAMIS

Recently, Dong et al. [15] presented a novel heuristic for MWIS called METAMIS. Since their code is not publicly available, we compare the solution quality of our algorithm against the results presented in their work. Detailed per-instance results can be found in Table 8. We use the same pre-reduced osm instances as Dong et al. [15]. In their experiments the time limit for METAMIS is 1 500 seconds. However, as our algorithm unfolds its full potential over a long period of time, and our algorithm focused on higher-quality solutions and not fast running times, we stayed with a 10-hour time limit. Moreover, note that the results have been computed on different machines. Summarizing the results, we are able to compute the same or better solutions for all graphs. In total we were able to improve three solutions compared to the METAMIS results with both of our configurations. Especially for large instances, our algorithms outperform the results stated in [15]. However, it is not clear whether METAMIS would compute equally good solutions for the instances where M^{2}WIS performed better with a longer running time.

6 Conclusion and Future Work

In this work, we developed a novel memetic algorithm for the MWIS problem. It repeatedly reduces the graph until a high-quality solution to the MWIS problem is found. After applying exact reductions, we use the best solution computed by the evolutionary algorithm on the reduced graph to identify vertices likely to be in an MWIS. These are removed from the graph which further

Table 8: Comparison to quality of METAMIS from [15] for osm instances. **Bold** numbers indicate the best solution among the algorithms. As done by Dong et al. [15] we reduced the osm instances in advance using KaMIS [32]. For METAMIS the best result out of five runs is reported; we report the best solution out of four runs, each with a 10h time limit.

	t	ω	t	ω	t	ω
redOsm	metamis		M^{2}WIS + S		M^{2}WIS	
alabama-3	5,41	**45 449**	23,35	**45 449**	23,30	**45 449**
district-of-columbia-2	58,38	**100 302**	128,11	**100 302**	105,50	**100 302**
district-of-columbia-3	1 347,00	142 910	4 133,86	**143 056**	1 674,92	**143 056**
florida-3	3,08	**46 132**	7,09	**46 132**	7,04	**46 132**
greenland-3	28,02	**11 960**	2 556,12	**11 960**	363,89	**11 960**
hawaii-3	1 207,00	58 819	5 415,14	58 869	4 595,13	**58 870**
idaho-3	21,23	**9 224**	1 560,69	**9 224**	271,12	**9 224**
kansas-3	3,38	**5 694**	62,60	**5 694**	102,45	**5 694**
kentucky-3	1 387,00	30 789	9 125,55	**31 107**	6 457,95	**31 107**
massachusetts-3	2,22	**17 224**	63,29	**17 224**	103,09	**17 224**
north-carolina-3	0,38	**13 062**	61,20	**13 062**	116,08	**13 062**
oregon-3	11,56	**34 471**	150,93	**34 471**	230,12	**34 471**
rhode-island-2	0,27	**43 722**	0,68	**43 722**	0,64	**43 722**
rhode-island-3	449,70	**81 013**	3 175,87	**81 013**	1 660,12	**81 013**
vermont-3	9,33	**28 349**	1 238,25	**28 349**	135,12	**28 349**
virginia-3	9,08	**97 873**	144,95	**97 873**	146,12	**97 873**
washington-3	62,35	**118 196**	1 357,56	**118 196**	257,62	**118 196**
overall	metamis		M^{2}WIS + S		M^{2}WIS	
# best		14/17		16/17		17/17
gmean ω		36 153		36 179		36 179
gmean t		20,89		357,31		145,07

opens the reduction space and creates the possibility to apply this process repeatedly.

Overall, our two algorithm configurations compute the same or better results among the competitors. For most instances these results are probably close to the optimum and even small improvements in solution quality can yield substantial cost reduction for some applications [15].

For future work, we are interested in an island-based approach to obtain a parallelization of our evolutionary approach, as well as parallelization of the reductions. Both the ExactReduce and the HeuristicReduce routine can result in a disconnected reduced graph. We are interested in solving the problem on each of the resulting connected components separately, which also enables new parallelization possibilities. The code of our work is publicly available under `https://github.com/KarlsruheMIS`.

References

[1] Openstreetmap. *https://www.openstreetmap.org*. URL: https://www.openstreetmap.org.

[2] Suitesparce matrix collection. *https://sparse.tamu.edu*. URL: https://sparse.tamu.edu.

[3] F. N. Abu-Khzam, S. Lamm, M. Mnich, A. Noe, C. Schulz, and D. Strash. Recent advances in practical data reduction. In H. Bast, C. Korzen, U. Meyer, and M. Penschuck, editors, *Algorithms for Big Data: DFG Priority Program 1736*, pages 97–133. Springer Nature Switzerland, Cham, 2022. doi:10.1007/978-3-031-21534-6_6.

[4] T. Akiba and Y. Iwata. Branch-and-reduce exponential/FPT algorithms in practice: A case study of vertex cover. *Theoretical Computer Science*, 609, Part 1:211–225, 2016. doi:10.1016/j.tcs.2015.09.023.

[5] G. Alexe, P. L. Hammer, V. V. Lozin, and D. de Werra. Struction revisited. *Discrete applied mathematics*, 132(1-3):27–46, 2003. doi:10.1016/S0166-218X(03)00388-3.

[6] D. V. Andrade, M. G. Resende, and R. F. Werneck. Fast local search for the maximum independent set problem. *Journal of Heuristics*, 18(4):525–547, 2012. doi:10.1007/s10732-012-9196-4.

[7] L. Babel. A fast algorithm for the maximum weight clique problem. *Computing*, 52(1):31–38, 1994. doi:10.1007/BF02243394.

[8] E. Balas and C. S. Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal on Computing*, 15(4):1054–1068, 1986. doi:10.1137/0215075.

[9] L. Barth, B. Niedermann, M. Nöllenburg, and D. Strash. Temporal map labeling: A new unified framework with experiments. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '16, pages 23:1–23:10. ACM, 2016. doi:10.1145/2996913.2996957.

[10] S. Butenko and S. Trukhanov. Using critical sets to solve the maximum independent set problem. *Operations Research Letters*, 35(4):519–524, 2007. doi:10.1016/j.orl.2006.07.004.

[11] S. Cai, W. Hou, J. Lin, and Y. Li. Improving local search for minimum weight vertex cover by dynamic strategies. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 1412–1418, 2018. doi:10.24963/ijcai.2018/196.

[12] S. Cai, K. Su, and A. Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, 175(9-10):1672–1696, 2011. doi:10.1016/j.artint.2011.03.003.

[13] Ü. V. Çatalyürek, K. D. Devine, M. F. Faraj, L. Gottesbüren, T. Heuer, H. Meyerhenke, P. Sanders, S. Schlag, C. Schulz, D. Seemaier, and D. Wagner. More recent advances in (hyper)graph partitioning. *CoRR*, abs/2205.13202, 2022. arXiv:2205.13202, doi:10.48550/arXiv.2205.13202.

[14] T. A. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1–25, 2011.

[15] Y. Dong, A. V. Goldberg, A. Noe, N. Parotsidis, M. G. C. Resende, and Q. Spaen. A local search algorithm for large maximum weight independent set problems. In S. Chechik, G. Navarro, E. Rotenberg, and G. Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 45:1–45:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.45.

[16] C. Ebenegger, P. Hammer, and D. De Werra. Pseudo-boolean functions and stability of graphs. In *North-Holland mathematics studies*, volume 95, pages 83–97. Elsevier, 1984. doi:10.1016/S0304-0208(08)72955-4.

[17] Z. Fang, C. Li, and K. Xu. An exact algorithm based on maxsat reasoning for the maximum weight clique problem. *J. Artif. Intell. Res.*, 55:799–833, 2016. URL: https://doi.org/10.1613/jair.4953, doi:10.1613/JAIR.4953.

[18] A. Figiel, V. Froese, A. Nichterlein, and R. Niedermeier. There and back again: On applying data reduction rules by undoing others. In S. Chechik, G. Navarro, E. Rotenberg, and G. Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 53:1–53:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.53.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[20] A. Gellner, S. Lamm, C. Schulz, D. Strash, and B. Zaválnij. Boosting data reduction for the maximum weight independent set problem using increasing transformations. In M. Farach-Colton and S. Storandt, editors, *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2021, Virtual Conference, January 10-11, 2021*, pages 128–142. SIAM, 2021. doi:10.1137/1.9781611976472.10.

[21] A. Gemsa, M. Nöllenburg, and I. Rutter. Evaluation of labeling strategies for rotating maps. *ACM J. Exp. Algorithmics*, 21(1):1.4:1–1.4:21, 2016. doi:10.1145/2851493.

[22] A. Grosso, M. Locatelli, and W. Pullan. Simple Ingredients Leading to Very Efficient Heuristics for the Maximum Clique Problem. *J. Heuristics*, 14(6):587–612, 2008.

[23] J. Gu, W. Zheng, Y. Cai, and P. Peng. Towards computing a near-maximum weighted independent set on massive graphs. In F. Zhu, B. C. Ooi, and C. Miao, editors, *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 467–477. ACM, 2021. doi:10.1145/3447548.3467232.

[24] S. Held, W. J. Cook, and E. C. Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Math. Program. Comput.*, 4(4):363–381, 2012. URL: https://doi.org/10.1007/s12532-012-0042-3, doi:10.1007/S12532-012-0042-3.

[25] D. Hespe, S. Lamm, and C. Schorr. Targeted branching for the maximum independent set problem. In D. Coudert and E. Natale, editors, *19th International Symposium on Experimental Algorithms, SEA 2021, June 7-9, 2021, Nice, France*, volume 190 of *LIPIcs*, pages 17:1–17:21.

Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.SEA.2021.17.

[26] D. Hespe, S. Lamm, C. Schulz, and D. Strash. Wegotyoucovered: The winning solver from the PACE 2019 challenge, vertex cover track. In H. M. Bücker, X. S. Li, and S. Rajamanickam, editors, *Proceedings of the SIAM Workshop on Combinatorial Scientific Computing, CSC 2020, Seattle, USA, February 11-13, 2020*, pages 1–11. SIAM, 2020. doi:10.1137/1.9781611976229.1.

[27] H. Jiang, C. Li, and F. Manyà. An exact algorithm for the maximum weight clique problem in large graphs. In S. Singh and S. Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 830–838. AAAI Press, 2017. URL: https://doi.org/10.1609/aaai.v31i1.10648, doi:10.1609/AAAI.V31I1.10648.

[28] A. Kenny, X. Li, and A. T. Ernst. A merge search algorithm and its application to the constrained pit problem in mining. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, page 316–323, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3205455.3205538.

[29] J. Kim, I. Hwang, Y. H. Kim, and B. R. Moon. Genetic Approaches for Graph Partitioning: A Survey. In *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO'11)*, pages 473–480. ACM, 2011.

[30] S. Lamm, P. Sanders, and C. Schulz. Graph partitioning for independent sets. In E. Bampis, editor, *Experimental Algorithms - 14th International Symposium, SEA 2015, Paris, France, June 29 - July 1, 2015, Proceedings*, volume 9125 of *Lecture Notes in Computer Science*, pages 68–81. Springer, 2015. doi:10.1007/978-3-319-20086-6_6.

[31] S. Lamm, P. Sanders, C. Schulz, D. Strash, and R. F. Werneck. Finding near-optimal independent sets at scale. *Journal of Heuristics*, 23(4):207–229, 2017. doi:10.1007/s10732-017-9337-x.

[32] S. Lamm, C. Schulz, D. Strash, R. Williger, and H. Zhang. Exactly solving the maximum weight independent set problem on large real-world graphs. In S. G. Kobourov and H. Meyerhenke, editors, *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019, San Diego, CA, USA, January 7-8, 2019*, pages 144–158. SIAM, 2019. doi:10.1137/1.9781611975499.12.

[33] K. Langedal, J. Langguth, F. Manne, and D. T. Schroeder. Efficient minimum weight vertex cover heuristics using graph neural networks. In C. Schulz and B. Uçar, editors, *20th International Symposium on Experimental Algorithms, SEA 2022, July 25-27, 2022, Heidelberg, Germany*, volume 233 of *LIPIcs*, pages 12:1–12:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: https://doi.org/10.4230/LIPIcs.SEA.2022.12, doi:10.4230/LIPICS.SEA.2022.12.

[34] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. URL http://snap.stanford.edu/data, June 2014.

[35] C.-M. Li, H. Jiang, and F. Manyà. On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Computers & Operations Research*, 84:1–15, 2017. doi:10.1016/j.cor.2017.02.017.

[36] R. Li, S. Hu, S. Cai, J. Gao, Y. Wang, and M. Yin. Numwvc: A novel local search for minimum weighted vertex cover problem. *J. Oper. Res. Soc.*, 71(9):1498–1509, 2020. doi:10.1080/01605682.2019.1621218.

[37] Y. Li, S. Cai, and W. Hou. An efficient local search algorithm for minimum weighted vertex cover on massive graphs. In *Asia-Pacific Conference on Simulated Evolution and Learning (SEAL 2017)*, volume 10593 of *LNCS*, pages 145–157. 2017. doi:10.1007/978-3-319-68759-9_13.

[38] F. Mascia, E. Cilia, M. Brunato, and A. Passerini. Predicting structural and functional sites in proteins by searching for maximum-weight cliques. In M. Fox and D. Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, pages 1274–1279. AAAI Press, 2010. URL: https://doi.org/10.1609/aaai.v24i1.7495, doi:10.1609/AAAI.V24I1.7495.

[39] B. L. Miller and D. E. Goldberg. Genetic Algorithms, Tournament Selection, and the Effects of Noise. *Evolutionary Computation*, 4(2):113–131, 1996.

[40] B. Nogueira, R. G. S. Pinheiro, and A. Subramanian. A hybrid iterated local search heuristic for the maximum weight independent set problem. *Optimization Letters*, 12(3):567–583, 2018. doi:10.1007/s11590-017-1128-7.

[41] P. R. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1-3):197–207, 2002. doi:10.1016/S0166-218X(01)00290-6.

[42] P. Prosser and J. Trimble. *Peaty: An exact solver for the vertex cover problem*, 2019. doi:10.5281/zenodo.3082356.

[43] S. Rebennack, M. Oswald, D. O. Theis, H. Seitz, G. Reinelt, and P. M. Pardalos. A branch and cut solver for the maximum stable set problem. *Journal of combinatorial optimization*, 21(4):434–457, 2011. doi:10.1007/s10878-009-9264-3.

[44] P. San Segundo, F. Matia, D. Rodriguez-Losada, and M. Hernando. An improved bit parallel exact maximum clique algorithm. *Optimization Letters*, 7(3):467–479, 2013. URL: http://dx.doi.org/10.1007/s11590-011-0431-y, doi:10.1007/s11590-011-0431-y.

[45] P. San Segundo, D. Rodríguez-Losada, and J. Agustín. An exact bit-parallel algorithm for the maximum clique problem. *Computers & Operations Research*, 38(2):571–581, 2011. URL: http://www.sciencedirect.com/science/article/pii/S0305054810001504, doi:http://dx.doi.org/10.1016/j.cor.2010.07.019.

[46] P. V. Sander, D. Nehab, E. Chlamtac, and H. Hoppe. Efficient traversal of mesh edges using adjacency primitives. *ACM Transactions on Graphics (TOG)*, 27(5):1–9, 2008. doi:10.1145/1409060.1409097.

[47] P. Sanders and C. Schulz. KaHIP – Karlsruhe High Qualtity Partitioning Homepage. http://algo2.iti.kit.edu/documents/kahip/index.html.

[48] P. Sanders and C. Schulz. Engineering Multilevel Graph Partitioning Algorithms. In *19th European Symposium on Algorithms*, volume 6942 of *LNCS*, pages 469–480. Springer, 2011.

[49] P. Sanders and C. Schulz. Advanced multilevel node separator algorithms. In *Experimental Algorithms - 15th International Symposium, (SEA), Proceedings*, volume 9685 of *LNCS*, pages 294–309. Springer, 2016. doi:10.1007/978-3-319-38851-9_20.

[50] C. Schulz and D. Strash. Graph partitioning: Formulations and applications to big data. In *Encyclopedia of Big Data Technologies*, pages 858–864. Springer International Publishing, 2019. doi:10.1007/978-3-319-77525-8_312.

[51] A. J. Soper, C. Walshaw, and M. Cross. A combined evolutionary search and multilevel optimisation approach to graph-partitioning. *J. Glob. Optim.*, 29(2):225–241, 2004. URL: https://doi.org/10.1023/B:JOGO.0000042115.44455.f3, doi:10.1023/B:JOGO.0000042115.44455.F3.

[52] S. Szabó and B. Zavalnij. Combining algorithms for vertex cover and clique search. In *Proceedings of the 22nd International Multiconference INFORMATION SOCIETY – IS 2019, Volume I: Middle-European Conference on Applied Theoretical Computer Science*, pages 71–74, 2019.

[53] E. Tomita, Y. Sutani, T. Higashi, S. Takahashi, and M. Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In M. S. Rahman and S. Fujita, editors, *WALCOM: Algorithms and Computation*, volume 5942 of *LNCS*, pages 191–203. Springer Berlin Heidelberg, 2010. URL: http://dx.doi.org/10.1007/978-3-642-11440-3_18, doi:10.1007/978-3-642-11440-3_18.

[54] L. Wang, C.-M. Li, J. Zhou, B. Jin, and M. Yin. An exact algorithm for minimum weight vertex cover problem in large graphs. *Computing Research Repository (CoRR)*, abs/1903.05948, 2019. doi:10.48550/ARXIV.1903.05948.

[55] J. S. Warren and I. V. Hicks. Combinatorial branch-and-bound for the maximum weight independent set problem. 2006. URL: https://www.caam.rice.edu/~ivhicks/jeff.rev.pdf.

[56] D. Warrier. *A branch, price, and cut approach to solving the maximum weighted independent set problem*. PhD thesis, Texas A&M University, 2007. doi:1969.1/5814.

[57] D. Warrier, W. E. Wilhelm, J. S. Warren, and I. V. Hicks. A branch-and-price approach for the maximum weight independent set problem. *Networks: An International Journal*, 46(4):198–209, 2005. doi:10.1002/net.20088.

[58] Q. Wu and J.-K. Hao. Solving the winner determination problem via a weighted maximum clique heuristic. *Expert Systems with Applications*, 42(1):355–365, 2015. doi:10.1016/j.eswa.2014.07.027.

[59] M. Xiao, S. Huang, and X. Chen. Maximum weighted independent set: Effective reductions and fast algorithms on sparse graphs. *Algorithmica*, 86(5):1293–1334, 2024. URL: https://doi.org/10.1007/s00453-023-01197-x, doi:10.1007/S00453-023-01197-X.

[60] M. Xiao, S. Huang, Y. Zhou, and B. Ding. Efficient reductions and a fast algorithm of maximum weighted independent set. In J. Leskovec, M. Grobelnik, M. Najork, J. Tang, and L. Zia, editors, *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 3930–3940. ACM / IW3C2, 2021. doi:10.1145/3442381.3450130.

[61] M. Xiao, S. Huang, Y. Zhou, and B. Ding. Efficient reductions and a fast algorithm of maximum weighted independent set. In J. Leskovec, M. Grobelnik, M. Najork, J. Tang, and L. Zia, editors, *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 3930–3940. ACM / IW3C2, 2021. doi:10.1145/3442381.3450130.

[62] H. Xu, T. S. Kumar, and S. Koenig. A new solver for the minimum weighted vertex cover problem. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 392–405. Springer, 2016. doi:10.1007/978-3-319-33954-2_28.

[63] W. Zheng, J. Gu, P. Peng, and J. X. Yu. Efficient weighted independent set computation over large graphs. In *IEEE Intl. Conf. on Data Engineering (ICDE)*, pages 1970–1973, 2020. doi:10.1109/ICDE48307.2020.00216.

7 Detailed Per-Instance Results for State of the Art Comparison

Table 9: Average solution weight ω and time t in seconds required to compute ω for our set of finite element instances. **Bold** numbers indicate the best solution among all algorithms. Rows have a gray background color, if branch reduce or struction computed an exact solution. We also report the number of best solutions and the geometric mean time needed to find the best solution over all instances.

finEl	branch reduce t	branch reduce ω	GNN VC t	GNN VC ω	HILS t	HILS ω	HtWIS t	HtWIS ω	NuMWVC t	NuMWVC ω	struction t	struction ω	M²WIS + S t	M²WIS + S ω	M²WIS t	M²WIS ω
body	-	-	3 112,38	1 678 235	1 259,67	1 678 510	0,04	1 645 650	289,44	1 654 412	-	-	112,85	**1 680 182**	901,26	1 680 179
ocean	4,88	**7 248 581**	1 018,08	7 210 228	11 142,43	7 075 329	0,07	6 803 672	840,54	7 071 829	-	-	5,20	**7 248 581**	5,01	**7 248 581**
pwt	-	-	2 754,08	1 174 472	761,52	1 175 437	0,03	1 153 600	396,97	1 149 919	-	-	3 846,33	**1 178 734**	5 131,59	1 178 583
rotor	-	-	5 559,60	2 643 669	6 503,27	2 650 018	0,24	2 591 456	1 071,61	2 569 961	-	-	29 249,04	**2 662 247**	24 255,33	2 661 514
sphere	-	-	51,71	615 017	257,42	615 958	0,02	608 401	171,28	613 128	0,57	**617 816**	1,96	**617 816**	2,25	**617 816**

overall	branch reduce	GNN VC	HILS	HtWIS	NuMWVC	struction	M²WIS + S	M²WIS
# best	1/5	0/5	0/5	0/5	0/5	1/5	5/5	2/5
gmean ω	-	1 873 906	1 868 676	1 827 149	1 841 892	-	1 882 030	1 881 878
gmean t	-	1 201,98	1 780,48	0,05	446,40	-	166,87	263,30

Table 10: Average solution weight ω and time t in seconds required to compute ω for our set of ssmc instances. **Bold** numbers indicate the best solution among all algorithms. Rows have a gray background color, if branch reduce or struction computed an exact solution. We also report the number of best solutions and the geometric mean time needed to find the best solution over all instances.

	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω
ssmc	branch reduce		GNN VC		HILS		HtWIS		NuMWVC		struction		M^{2}WIS + S		M^{2}WIS	
ca2010	-	-	4 702,04	16 817 567	36 000,07	16 828 547	0,47	16 792 827	21 336,94	16 053 480	6,18	**16 869 550**	28,32	**16 869 550**	10 985,99	16 869 127
fl2010	36 000,10	8 638 961	883,37	8 721 942	36 000,05	8 732 113	0,44	8 719 272	14 712,40	8 248 077	2,02	**8 743 506**	13,57	**8 743 506**	7 492,86	8 743 474
ga2010	36 000,10	4 644 324	18 636,56	4 637 542	29 522,41	4 642 807	0,16	4 639 891	7 492,19	4 437 269	0,62	**4 644 417**	1,64	**4 644 417**	1,85	**4 644 417**
il2010	36 000,10	5 852 296	4 106,19	5 981 072	36 000,00	5 983 871	0,31	5 963 974	13 341,29	5 776 584	2,33	**5 998 539**	22,98	**5 998 539**	13 136,79	5 998 288
nh2010	36 000,00	581 637	2 328,87	586 642	2 163,80	588 797	0,03	587 059	758,10	568 188	0,11	**588 996**	0,47	**588 996**	0,47	**588 996**
ri2010	36 000,00	447 427	717,01	457 288	782,49	458 489	0,02	457 108	371,27	454 423	0,09	**459 275**	0,62	**459 275**	0,49	**459 275**
overall	branch reduce		GNN VC		HILS		HtWIS		NuMWVC		struction		M^{2}WIS + S		M^{2}WIS	
# best		0/6		0/6		0/6		0/6		0/6		6/6		6/6		3/6
gmean ω		-		3 208 739		3 213 935		3 206 698		3 093 453		3 218 537		3 218 537		3 218 499
gmean t		-		2 845,45		11 515,77		0,13		4 546,46		0,75		4,02		88,11

Table 11: Average solution weight ω and time t in seconds required to compute ω for our set of mesh instances. **Bold** numbers indicate the best solution among all algorithms. Rows have a gray background color, if branch reduce or struction computed an exact solution. We also report the number of best solutions and the geometric mean time needed to find the best solution over all instances.

	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω
mesh	branch reduce		GNN VC		HILS		HtWIS		NuMWVC		struction		M^{2}WIS + S		M^{2}WIS	
blob	0,14	**855 547**	20 854,69	854 991	260,10	854 803	0,01	854 484	190,14	853 616	0,02	**855 547**	0,04	**855 547**	0,03	**855 547**
buddha	51,87	**57 555 880**	1 439,13	57 497 819	36 000,07	57 258 790	0,47	57 508 556	29 022,48	55 973 791	1,57	**57 555 880**	4,83	**57 555 880**	2 417,30	57 555 864
bunny	0,48	**3 686 960**	1 582,74	3 683 941	1 927,84	3 680 587	0,03	3 682 356	817,06	3 659 471	0,09	**3 686 960**	0,17	**3 686 960**	0,15	**3 686 960**
cow	0,04	**269 543**	6 571,10	269 402	52,05	269 336	<0,01	269 304	15,62	269 220	0,01	**269 543**	0,01	**269 543**	0,01	**269 543**
dragon	2,90	**7 956 530**	555,32	7 949 744	9 026,60	7 947 535	0,04	7 950 526	1 922,25	7 846 579	0,17	**7 956 530**	0,31	**7 956 530**	0,33	**7 956 530**
dragonsub	4,76	**32 213 898**	619,21	32 182 406	36 000,07	32 148 544	0,24	32 163 872	16 990,27	31 294 908	0,93	**32 213 898**	2,10	**32 213 898**	1,93	**32 213 898**
ecat	9,18	**36 650 298**	431,79	36 616 204	36 000,05	36 562 652	0,50	36 606 394	16 759,95	35 603 546	1,92	**36 650 298**	2,83	**36 650 298**	3,01	**36 650 298**
face	0,16	**1 219 418**	480,83	1 218 552	403,52	1 218 433	0,01	1 218 515	199,42	1 213 445	0,02	**1 219 418**	0,04	**1 219 418**	0,03	**1 219 418**
fandisk	0,04	**463 288**	146,07	462 910	114,01	462 794	<0,01	462 765	67,53	462 463	0,01	**463 288**	0,02	**463 288**	0,01	**463 288**
feline	0,34	**2 207 219**	450,92	2 205 443	794,54	2 204 454	0,02	2 204 947	474,86	2 189 581	0,05	**2 207 219**	0,10	**2 207 219**	0,09	**2 207 219**
gameguy	0,10	**2 325 878**	27,44	2 324 222	789,78	2 322 814	0,02	2 324 088	431,10	2 306 182	0,04	**2 325 878**	0,06	**2 325 878**	0,05	**2 325 878**
gargoyle	0,22	**1 059 559**	1 748,69	1 058 724	346,19	1 058 536	0,01	1 058 656	283,58	1 055 563	0,02	**1 059 559**	0,04	**1 059 559**	0,03	**1 059 559**
turtle	3,98	**14 263 005**	3 342,67	14 249 692	20 430,40	14 245 854	0,09	14 247 883	5 991,49	13 962 425	0,35	**14 263 005**	0,74	**14 263 005**	0,73	**14 263 005**
venus	0,02	**305 749**	298,23	305 571	59,48	305 556	<0,01	305 182	26,18	305 457	0,01	**305 749**	0,01	**305 749**	0,01	**305 749**
overall	branch reduce		GNN VC		HILS		HtWIS		NuMWVC		struction		M^{2}WIS + S		M^{2}WIS	
# best		14/14		0/14		0/14		0/14		0/14		14/14		14/14		13/14
gmean ω		3 054 726		3 052 303		3 050 072		3 051 422		3 020 035		3 054 726		3 054 726		3 054 726
gmean t		0,50		827,18		1 418,48		0,03		682,18		0,07		0,14		0,19

Table 12: Average solution weight ω and time t in seconds required to compute ω for our set of osm instances. Displayed as single row are only instances with $|V| \geq 1\,000$, the summary includes all instances. **Bold** numbers indicate the best solution among all algorithms. Rows have a gray background color, if branch reduce or struction computed an exact solution. We also report the number of best solutions and the geometric mean time needed to find the best solution over all instances.

osm	branch reduce		GNN VC		HILS		HtWIS		NuMWVC		struction		M²WIS + S		M²WIS	
	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω
alabama-2	0,35	**174 309**	867,30	174 124	41,71	**174 309**	0,01	172 797	0,89	172 871	0,01	**174 309**	0,03	**174 309**	0,03	**174 309**
alabama-3	36 000,00	185 707	32 238,96	181 548	321,23	**185 744**	0,40	182 667	39,57	180 461	1,53	**185 744**	32,67	**185 744**	28,53	**185 744**
district-of-columbia-1	-	-	11 408,24	196 341	41,16	**196 475**	0,01	193 364	19,42	196 044	0,47	**196 475**	1,50	**196 475**	1,26	**196 475**
district-of-columbia-2	-	-	11 021,45	204 843	985,93	209 131	2,25	198 327	178,71	202 111	-	-	184,04	**209 132**	107,25	**209 132**
district-of-columbia-3	36 000,20	207 787	6 233,48	212 734	9 278,60	227 634	351,94	210 461	681,32	212 583	-	-	4 283,28	**227 682**	3 231,32	227 681
florida-2	0,01	**230 595**	294,75	**230 595**	42,92	**230 595**	<0,01	230 008	4,65	229 496	<0,01	**230 595**	0,01	**230 595**	0,01	**230 595**
florida-3	1 724,45	**237 333**	3 533,44	232 105	216,24	**237 333**	0,13	234 218	25,20	234 940	1,33	**237 333**	11,20	**237 333**	9,57	**237 333**
georgia-3	1 772,88	**222 652**	269,99	219 891	101,22	**222 652**	0,09	218 573	15,81	218 610	0,77	**222 652**	7,55	**222 652**	6,13	**222 652**
greenland-3	36 000,00	13 894	261,66	12 561	1 226,78	**14 011**	32,75	12 505	122,36	12 494	-	-	521,57	**14 011**	185,01	**14 011**
hawaii-2	8,20	**125 284**	1 311,40	124 466	203,81	**125 284**	0,14	123 173	42,44	123 757	0,07	**125 284**	0,46	**125 284**	0,36	**125 284**
hawaii-3	36 003,35	132 806	3 568,95	132 562	17 330,34	141 045	1 577,65	134 703	-	-	-	-	6 516,88	141 056	5 124,22	**141 058**
idaho-3	36 000,00	77 122	180,84	75 831	1 549,78	**77 145**	52,16	75 527	42,76	75 901	-	-	2 142,01	**77 145**	410,29	**77 145**
kansas-3	36 001,02	87 963	23 741,49	87 688	759,69	**87 976**	2,43	87 424	137,89	87 497	16,79	**87 976**	65,05	**87 976**	104,24	**87 976**
kentucky-2	63,34	**97 397**	3 595,24	96 550	319,15	**97 397**	0,74	97 362	141,39	96 838	0,20	**97 397**	0,83	**97 397**	0,68	**97 397**
kentucky-3	36 001,57	100 311	2 502,83	98 310	28 313,04	100 508	3 508,49	97 906	-	-	-	-	13 121,68	100 508	8 865,42	**100 510**
louisiana-3	22,52	**60 024**	7 238,25	59 473	62,47	**60 024**	0,02	59 040	7,55	58 854	0,05	**60 024**	0,60	**60 024**	0,47	**60 024**
maryland-3	9,48	**45 496**	628,67	44 988	95,12	**45 496**	0,04	44 539	12,53	45 081	0,10	**45 496**	0,73	**45 496**	0,68	**45 496**
massachusetts-2	0,37	**140 095**	1 186,63	140 051	51,59	**140 095**	0,02	139 799	6,05	139 697	0,04	**140 095**	0,09	**140 095**	0,09	**140 095**
massachusetts-3	-	-	4 622,32	144 140	355,93	**145 866**	1,77	144 381	79,71	143 984	-	-	77,14	**145 866**	104,87	**145 866**
mexico-3	921,72	**97 663**	2 969,21	95 902	90,92	**97 663**	0,05	96 700	1,38	95 999	0,86	**97 663**	14,73	**97 663**	14,07	**97 663**
new-hampshire-3	14,46	**116 060**	40,64	115 734	51,78	**116 060**	0,01	115 161	2,76	112 771	0,04	**116 060**	1,69	**116 060**	1,40	**116 060**
north-carolina-3	36 000,05	49 563	1 658,84	48 309	205,78	**49 720**	0,42	49 253	51,48	48 306	37,25	**49 720**	142,20	**49 720**	117,07	**49 720**
oregon-2	0,03	**165 047**	91,84	164 144	74,19	**165 047**	0,03	164 786	7,18	164 006	0,01	**165 047**	0,02	**165 047**	0,01	**165 047**
oregon-3	36 001,90	**175 078**	1 735,05	170 748	1 120,13	**175 078**	27,87	172 813	435,08	171 321	-	-	292,46	**175 078**	266,55	**175 078**
pennsylvania-3	107,51	**143 870**	111,17	143 406	60,62	**143 870**	0,02	142 472	2,39	140 352	0,06	**143 870**	2,30	**143 870**	2,01	**143 870**
rhode-island-2	-	-	18 480,55	182 234	166,18	**184 596**	0,36	179 366	48,37	183 127	0,38	**184 596**	1,39	**184 596**	1,46	**184 596**
rhode-island-3	36 000,20	196 173	21 044,39	192 449	3 899,85	201 751	280,42	190 341	162,00	189 587	-	-	3 319,95	**201 769**	1 964,28	201 768
utah-3	239,50	**98 847**	14 144,18	97 033	72,21	**98 847**	0,04	97 754	9,95	96 251	0,08	**98 847**	2,34	**98 847**	1,94	**98 847**
vermont-3	36 000,45	63 305	25 708,04	59 773	842,04	63 304	3,81	60 518	167,97	60 454	-	-	1 855,99	**63 312**	142,43	**63 312**
virginia-2	0,60	**295 867**	1 341,69	294 149	87,88	**295 867**	0,02	290 535	3,56	295 171	0,02	**295 867**	0,12	**295 867**	0,12	**295 867**
virginia-3	36 000,80	307 981	35,87	296 735	482,02	**308 305**	1,14	300 335	82,45	298 038	-	-	1 021,68	**308 305**	172,77	**308 305**
washington-2	5,67	**305 619**	2 794,95	301 157	199,38	**305 619**	0,06	300 195	18,68	303 632	0,05	**305 619**	0,23	**305 619**	0,19	**305 619**
washington-3	-	-	9,58	295 610	1 946,06	**314 288**	11,74	305 019	168,96	297 908	-	-	2 731,64	**314 288**	327,63	**314 288**
west-virginia-3	36 000,27	**47 927**	9 801,99	46 791	147,10	**47 927**	0,25	46 344	0,37	45 499	2,35	**47 927**	64,82	**47 927**	104,41	**47 927**

overall	branch reduce	GNN VC	HILS	HtWIS	NuMWVC	struction	M²WIS + S	M²WIS
# best	132/148	101/148	142/148	86/148	0/148	136/148	146/148	146/148
gmean ω	-	39 597	39 823	39 520	-	-	39 823	39 823
gmean t	-	4,92	12,49	<0,01	-	-	0,01	0,01

Table 13: Average solution weight ω and time t in seconds required to compute ω for our set of snap instances. **Bold** numbers indicate the best solution among all algorithms. Rows have a gray background color, if branch reduce or struction computed an exact solution. We also report the number of best solutions and the geometric mean time needed to find the best solution over all instances.

snap	branch reduce		GNN VC		HILS		HtWIS		NuMWVC		struction		$\mathrm{M^2WIS + S}$		$\mathrm{M^2WIS}$	
	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω
as-skitter	-	-	1 166,25	124 155 737	36 000,25	123 994 141	1,04	124 141 373	34 265,33	123 594 776	-	-	3 422,79	**124 157 729**	621,62	**124 157 729**
ca-AstroPh	0,02	**797 510**	1,02	**797 510**	924,29	797 508	0,02	797 363	315,84	796 125	0,02	**797 510**	0,04	**797 510**	0,04	**797 510**
ca-CondMat	0,02	**1 147 950**	0,34	**1 147 950**	1 015,80	1 147 947	0,01	**1 147 950**	430,29	1 146 066	0,01	**1 147 950**	0,02	**1 147 950**	0,02	**1 147 950**
ca-GrQc	<0,01	**287 919**	0,21	**287 919**	144,49	**287 919**	<0,01	287 850	59,07	287 638	<0,01	**287 919**	<0,01	**287 919**	<0,01	**287 919**
ca-HepPh	0,01	**581 039**	18,03	**581 039**	589,17	**581 039**	0,01	580 864	212,89	580 439	0,01	**581 039**	0,02	**581 039**	0,02	**581 039**
ca-HepTh	0,01	**562 004**	0,17	**562 004**	279,70	**562 004**	<0,01	561 736	150,18	561 331	<0,01	**562 004**	0,01	**562 004**	0,01	**562 004**
com-amazon	0,48	**19 271 031**	2,64	**19 271 031**	33 178,00	19 270 284	0,14	19 270 078	10 316,17	19 083 928	0,36	**19 271 031**	0,56	**19 271 031**	0,53	**19 271 031**
com-youtube	0,76	**90 295 294**	4,96	**90 295 294**	36 000,10	90 289 947	0,34	90 295 285	11 288,89	89 986 918	0,69	**90 295 294**	0,90	**90 295 294**	0,97	**90 295 294**
email-Enron	0,02	**2 461 254**	0,40	**2 461 254**	1 432,11	2 461 242	0,01	**2 461 254**	587,52	2 459 380	0,02	**2 461 254**	0,03	**2 461 254**	0,03	**2 461 254**
email-EuAll	0,07	**25 286 322**	0,73	**25 286 322**	17 439,93	**25 286 322**	0,03	25 265 214	2 089,93	25 285 560	0,04	**25 286 322**	0,11	**25 286 322**	0,11	**25 286 322**
loc-gowalla_e.	-	-	985,50	12 276 922	17 018,50	12 275 375	0,08	12 276 781	3 886,18	12 203 398	1,32	**12 276 929**	3,89	**12 276 929**	4,71	**12 276 929**
p2p-G.04	0,01	**679 111**	0,18	**679 111**	250,11	679 110	<0,01	679 085	123,87	678 920	0,01	**679 111**	0,01	**679 111**	0,01	**679 111**
p2p-G.05	0,01	**554 943**	0,18	**554 943**	192,45	**554 943**	<0,01	**554 943**	76,17	554 757	<0,01	**554 943**	0,01	**554 943**	0,01	**554 943**
p2p-G.06	0,01	**548 612**	0,20	**548 612**	183,91	**548 612**	<0,01	**548 612**	70,78	548 449	<0,01	**548 612**	0,01	**548 612**	0,01	**548 612**
p2p-G.08	<0,01	**434 577**	0,13	**434 577**	109,72	**434 577**	<0,01	**434 577**	14,93	434 513	<0,01	**434 577**	<0,01	**434 577**	<0,01	**434 577**
p2p-G.09	<0,01	**568 439**	0,16	**568 439**	152,25	**568 439**	<0,01	**568 439**	20,44	568 351	<0,01	**568 439**	0,01	**568 439**	0,01	**568 439**
p2p-G.24	0,01	**1 984 567**	0,15	**1 984 567**	569,39	**1 984 567**	0,01	**1 984 567**	339,70	1 984 248	0,01	**1 984 567**	0,02	**1 984 567**	0,01	**1 984 567**
p2p-G.25	0,01	**1 701 967**	0,13	**1 701 967**	467,31	**1 701 967**	0,01	**1 701 967**	129,05	1 701 819	0,01	**1 701 967**	0,01	**1 701 967**	0,01	**1 701 967**
p2p-G.30	0,02	**2 787 907**	0,18	**2 787 907**	810,63	**2 787 907**	0,01	2 787 902	285,78	2 787 660	0,01	**2 787 907**	0,02	**2 787 907**	0,02	**2 787 907**
p2p-G.31	0,03	**4 776 986**	0,28	**4 776 986**	1 795,27	4 776 969	0,01	4 776 925	789,92	4 776 386	0,02	**4 776 986**	0,04	**4 776 986**	0,04	**4 776 986**
roadNet-CA	279,50	**111 360 828**	3 911,60	111 337 979	36 000,15	109 991 788	0,61	111 325 524	35 932,85	108 909 808	1,54	**111 360 828**	5,02	**111 360 828**	4,67	**111 360 828**
roadNet-PA	16,44	**61 731 589**	7 394,67	61 719 900	36 000,07	61 549 659	0,33	61 710 606	23 724,64	60 461 602	0,85	**61 731 589**	2,19	**61 731 589**	2,38	**61 731 589**
roadNet-TX	15,76	**78 599 946**	3 846,17	78 586 678	36 000,10	78 164 327	0,42	78 575 460	34 979,99	76 992 375	1,05	**78 599 946**	2,81	**78 599 946**	2,90	**78 599 946**
soc-Epinions1	0,05	**5 690 970**	0,59	**5 690 970**	2 813,40	5 690 859	0,02	**5 690 970**	1 043,52	5 686 352	0,05	**5 690 970**	0,07	**5 690 970**	0,06	**5 690 970**
soc-LiveJ.	36 002,35	284 008 877	16 424,34	284 026 908	36 000,67	281 688 778	12,20	283 922 214	-	-	-	-	779,88	**284 036 239**	1 738,22	**284 036 239**
soc-pokec-rel.	36 059,01	82 778 214	23 661,02	83 924 150	36 000,42	83 696 885	55,41	83 920 370	31 310,08	83 187 970	634,61	79 620 979	9 691,59	83 939 404	9 038,95	**83 944 926**
soc-S.0811	0,06	**5 660 899**	0,63	**5 660 899**	4 106,47	5 660 734	0,02	**5 660 899**	1 091,76	5 655 800	0,06	**5 660 899**	0,08	**5 660 899**	0,08	**5 660 899**
soc-S.0902	0,07	**5 971 849**	0,62	**5 971 849**	4 260,67	5 971 574	0,02	5 971 821	1 082,17	5 965 971	0,07	**5 971 849**	0,08	**5 971 849**	0,13	**5 971 849**
web-BerkStan	36 000,12	43 891 206	472,62	43 904 999	36 000,10	43 888 267	9,94	43 889 843	12 844,27	43 473 969	6,52	**43 907 482**	8,75	**43 907 482**	9,32	**43 907 482**
web-Google	2,33	**56 326 504**	45,40	56 326 476	36 000,15	56 319 614	0,65	56 323 382	9 398,74	56 023 547	1,52	**56 326 504**	2,40	**56 326 504**	2,47	**56 326 504**
web-NotreD.	496,64	**26 016 941**	1 596,06	26 016 642	27 389,91	26 014 810	0,12	26 013 830	7 659,14	25 928 547	1,36	**26 016 941**	0,99	**26 016 941**	1,09	**26 016 941**
web-Stanford	-	-	1 746,45	17 792 664	35 324,07	17 789 989	0,50	17 789 430	8 338,97	17 605 207	1,36	**17 792 930**	1,84	**17 792 930**	1,93	**17 792 930**
wiki-Talk	1,08	**235 837 346**	16,94	**235 837 346**	36 000,12	235 837 287	0,41	**235 837 346**	34 505,67	235 822 182	0,95	**235 837 346**	1,39	**235 837 346**	1,53	**235 837 346**
wiki-Vote	0,01	**500 079**	5,21	**500 079**	201,84	**500 079**	0,01	499 740	24,84	499 993	0,01	**500 079**	0,02	**500 079**	0,02	**500 079**

overall	branch reduce	GNN VC	HILS	HtWIS	NuMWVC	struction	$\mathrm{M^2WIS + S}$	$\mathrm{M^2WIS}$
# best	28/34	23/34	12/34	11/34	0/34	31/34	33/34	34/34
# fastest	8/34	0/34	0/34	33/34	0/34	14/34	3/34	4/34
gmean ω	-	6 678 015	6 671 408	6 677 139	-	-	6 678 187	6 678 200
gmean t	-	8,06	3 338,88	0,06	-	-	0,25	0,24

Table 14: Average solution weight ω and time t in seconds required to compute ω for all, not completely reduced instances with the reductions stated in Section 4.1. Only instances with $|V| \geq 1\,000$ are detailed, the summary includes all. **Bold** numbers indicate the best solution among all algorithms. Rows have a gray background color, if an algorithm computed an exact solution. We also report the number of best solutions and the geometric mean time needed to find the best solution over all instances.

	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω
redFinEl	branch reduce		GNN VC		HILS		HtWIS		NuMWVC		struction		M²WIS + S		M²WIS	
body	-	-	717,66	223 946	74,81	224 550	0,01	222 320	45,36	224 168	-	-	60,30	**224 744**	100,23	**224 744**
pwt	-	-	13 585,64	1 034 567	655,57	1 034 418	0,07	1 019 262	174,21	1 014 280	-	-	1 391,40	**1 038 139**	4 767,83	1 038 046
rotor	36 000,10	2 483 283	8 029,89	2 632 254	3 078,32	2 639 734	0,67	2 580 529	1 034,92	2 553 422	-	-	29 550,69	2 651 439	23 323,54	**2 651 735**
sphere	36 000,00	461 060	16,72	472 884	179,43	474 290	0,01	467 030	161,77	472 508	0,24	**475 344**	1,20	**475 344**	1,28	**475 344**
redMesh	branch reduce		GNN VC		HILS		HtWIS		NuMWVC		struction		M²WIS + S		M²WIS	
buddha	0,01	**23 026**	0,36	**23 026**	6,85	**23 026**	<0,01	22 726	0,86	22 937	<0,01	**23 026**	0,03	**23 026**	0,02	**23 026**
ecat	0,02	**31 254**	0,24	**31 254**	8,17	**31 254**	<0,01	30 789	1,12	31 121	<0,01	**31 254**	0,03	**31 254**	0,03	**31 254**
redOsm	branch reduce		GNN VC		HILS		HtWIS		NuMWVC		struction		M²WIS + S		M²WIS	
alabama-3	36 000,00	35 944	1 780,24	34 715	107,88	**35 968**	0,23	34 667	4,58	33 440	1,12	**35 968**	30,52	**35 968**	29,27	**35 968**
california-3	821,86	**13 689**	15 006,79	13 253	37,87	**13 689**	0,03	13 058	3,31	12 534	0,22	**13 689**	20,57	**13 689**	19,89	**13 689**
canada-3	55,19	**17 591**	6 069,32	17 487	19,42	**17 591**	0,01	16 063	0,50	16 892	0,13	**17 591**	2,43	**17 591**	2,30	**17 591**
colorado-3	0,14	**9 580**	0,17	**9 580**	7,11	**9 580**	<0,01	9 274	0,14	9 221	0,01	**9 580**	0,05	**9 580**	0,04	**9 580**
d.o.c.-1	-	-	31,71	**55 063**	12,13	**55 063**	0,01	52 450	1,41	54 803	0,43	**55 063**	1,63	**55 063**	1,53	**55 063**
d.o.c.-2	-	-	9 513,25	93 988	403,56	96 318	2,02	88 932	41,91	94 839	-	-	182,18	**96 322**	106,23	**96 322**
d.o.c.-3	36 000,10	119 502	11 779,45	124 774	4 626,91	138 744	283,13	124 431	254,73	127 833	-	-	3 810,44	138 795	2 788,96	**138 797**
florida-3	1 714,07	**25 992**	4 482,86	25 700	65,56	**25 992**	0,11	23 471	5,62	25 375	1,34	**25 992**	8,20	**25 992**	7,80	**25 992**
georgia-3	763,74	**33 714**	14 126,14	31 934	54,48	**33 714**	0,08	30 570	5,13	32 412	0,76	**33 714**	6,78	**33 714**	6,63	**33 714**
greenland-2	16,22	**3 537**	310,67	**3 537**	19,18	**3 537**	0,01	3 142	0,30	3 390	0,05	**3 537**	1,13	**3 537**	0,98	**3 537**
greenland-3	36 000,00	11 419	495,96	10 304	836,96	**11 581**	43,14	9 905	33,53	10 508	-	-	478,88	**11 581**	161,78	**11 581**
hawaii-2	1,42	**11 617**	203,72	11 595	16,62	**11 617**	0,01	11 410	0,81	11 452	0,01	**11 617**	0,22	**11 617**	0,22	**11 617**
hawaii-3	36 001,92	50 612	460,52	52 089	13 683,70	58 812	1 747,34	52 595	424,67	52 169	-	-	4 965,81	58 858	3 845,37	**58 861**
idaho-3	36 000,80	9 221	2 850,49	7 891	1 128,20	**9 224**	55,91	8 139	19,64	8 488	-	-	2 103,78	**9 224**	375,21	**9 224**
kansas-3	36 001,72	5 681	8 074,38	5 512	410,05	**5 694**	1,80	5 173	19,38	5 459	12,08	**5 694**	62,66	**5 694**	102,59	**5 694**
kentucky-2	17,08	**7 019**	8 979,49	6 981	30,49	**7 019**	0,02	6 979	0,26	6 816	0,04	**7 019**	0,52	**7 019**	0,43	**7 019**
kentucky-3	36 001,50	26 198	6 103,84	24 580	23 202,28	26 397	4 335,68	24 614	-	-	-	-	9 536,24	26 397	5 679,08	**26 398**
louisiana-3	2,52	**9 326**	5,30	9 094	16,06	**9 326**	<0,01	8 977	0,74	9 151	0,02	**9 326**	0,25	**9 326**	0,26	**9 326**
maryland-3	1,83	**7 105**	466,86	7 070	13,94	**7 105**	<0,01	6 561	0,45	6 153	0,09	**7 105**	0,42	**7 105**	0,40	**7 105**
mas.-2	0,32	**7 938**	14,41	**7 938**	9,80	**7 938**	<0,01	7 650	0,12	7 797	0,03	**7 938**	0,08	**7 938**	0,08	**7 938**
mas.-3	36 000,45	14 610	13 976,78	13 673	247,66	**14 757**	1,79	13 285	29,15	14 085	-	-	74,22	**14 757**	102,60	**14 757**
mexico-3	551,33	**16 137**	12 688,47	14 623	48,35	**16 137**	0,04	15 598	2,79	14 745	0,28	**16 137**	8,90	**16 137**	8,12	**16 137**

	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω
redOsm	branch reduce		GNN VC		HILS		HtWIS		NuMWVC		struction		M^{2}WIS + S		M^{2}WIS	
minnesota-3	8,04	**4 343**	20,69	**4 343**	12,04	**4 343**	<0,01	4 104	0,14	4 281	0,02	**4 343**	0,50	**4 343**	0,50	**4 343**
montana-3	874,67	**5 116**	10 322,49	4 991	66,85	**5 116**	0,10	5 094	7,32	5 015	0,19	**5 116**	3,85	**5 116**	3,72	**5 116**
new-hamp.-3	6,69	**11 473**	13,97	**11 473**	14,40	**11 473**	<0,01	11 104	0,63	10 597	0,04	**11 473**	1,03	**11 473**	1,12	**11 473**
new-york-2	0,22	**4 540**	3,77	**4 540**	8,43	**4 540**	<0,01	**4 540**	<0,01	4 163	0,01	**4 540**	0,03	**4 540**	0,04	**4 540**
new-york-3	9 493,18	**5 897**	9 405,58	5 832	76,89	**5 897**	0,15	4 922	2,78	5 552	0,29	**5 897**	60,80	**5 897**	100,98	**5 897**
north-car.-3	36 000,07	11 073	359,98	10 044	166,57	**11 191**	0,43	10 653	11,23	10 582	67,23	**11 191**	141,71	**11 191**	116,51	**11 191**
ohio-3	7,08	**5 213**	157,84	5 170	13,05	**5 213**	<0,01	4 794	0,23	4 050	0,01	**5 213**	0,61	**5 213**	0,61	**5 213**
oregon-3	36 002,17	23 422	17 211,78	21 920	804,84	**23 427**	28,97	20 386	98,83	22 395	-	-	249,18	**23 427**	268,10	**23 427**
pennsyl.-3	11,45	**14 766**	27,10	**14 766**	15,90	**14 766**	<0,01	13 921	0,14	13 745	0,03	**14 766**	1,31	**14 766**	1,25	**14 766**
puerto-r.-3	194,64	**3 544**	905,34	**3 544**	20,06	**3 544**	0,01	3 527	0,13	3 176	0,05	**3 544**	5,69	**3 544**	5,19	**3 544**
rhode-i.-2	4 746,44	**42 587**	7 789,73	42 526	62,79	**42 587**	0,17	39 486	7,04	41 619	0,28	**42 587**	0,98	**42 587**	1,03	**42 587**
rhode-i.-3	36 000,20	70 873	18 572,74	66 854	3 225,06	76 431	394,81	65 468	23,92	68 350	-	-	3 229,68	**76 458**	1 348,52	**76 458**
utah-3	200,33	**16 090**	980,62	15 018	41,05	**16 090**	0,03	15 186	3,58	15 167	0,07	**16 090**	1,77	**16 090**	1,64	**16 090**
vermont-2	17,92	**4 308**	5 549,13	4 290	25,14	**4 308**	0,01	3 343	0,94	3 862	0,02	**4 308**	1,04	**4 308**	0,99	**4 308**
vermont-3	36 000,20	22 804	16 180,19	21 215	457,32	**22 813**	3,66	20 011	51,72	21 726	-	-	1 849,27	**22 813**	133,53	**22 813**
virginia-2	0,26	**23 680**	2,80	**23 680**	9,70	**23 680**	<0,01	23 275	0,13	23 081	0,01	**23 680**	0,06	**23 680**	0,05	**23 680**
virginia-3	36 000,77	90 854	291,78	80 459	285,12	**91 046**	1,37	83 975	21,52	86 081	-	-	1 014,80	**91 046**	304,16	**91 046**
wash.-2	4,24	**33 415**	37,83	**33 415**	20,71	**33 415**	0,01	32 185	0,15	32 965	0,02	**33 415**	0,17	**33 415**	0,19	**33 415**
wash.-3	-	-	534,33	98 789	1 592,22	113 514	23,64	104 692	71,86	103 006	-	-	2 101,05	**113 516**	289,58	**113 516**
west-virg.-3	32 129,35	**16 666**	17 644,66	15 573	135,05	**16 666**	0,32	15 290	2,42	14 996	2,35	**16 666**	64,48	**16 666**	104,57	**16 666**
redSnap	branch reduce		GNN VC		HILS		HtWIS		NuMWVC		struction		M^{2}WIS + S		M^{2}WIS	
as-skitter	-	-	21 606,10	135 910	45,35	135 976	0,04	134 500	26,37	135 783	-	-	1 904,18	**135 998**	287,87	**135 998**
loc-gowalla_e.	294,11	**16 521**	25,68	**16 521**	8,40	**16 521**	<0,01	**16 521**	0,15	16 479	0,64	**16 521**	0,94	**16 521**	0,78	**16 521**
soc-LiveJ.	-	-	5 669,30	277 920	180,88	**278 532**	0,13	273 672	76,47	277 988	-	-	308,08	**278 532**	145,28	**278 532**
soc-pokec-rel.	36 038,14	34 227 442	18 389,52	35 219 027	36 000,15	**35 242 470**	81,18	35 205 509	7 035,18	34 828 738	644,98	32 989 873	8 238,05	35 225 724	7 259,99	35 230 752
web-BerkStan	28,05	**135 172**	15,35	134 942	36,21	135 156	<0,01	133 309	14,97	134 676	0,04	**135 172**	0,17	**135 172**	0,15	**135 172**
web-Google	0,13	**20 182**	1,69	**20 182**	9,22	**20 182**	<0,01	20 091	0,67	20 171	0,01	**20 182**	0,05	**20 182**	0,04	**20 182**
web-NotreD.	0,33	**29 145**	22,52	**29 145**	11,67	**29 145**	<0,01	28 640	0,56	29 061	0,07	**29 145**	0,11	**29 145**	0,09	**29 145**
web-Stanford	4 832,78	**31 695**	793,53	31 668	11,21	**31 695**	<0,01	31 427	2,36	31 617	0,01	**31 695**	0,04	**31 695**	0,05	**31 695**
redSsmc	branch reduce		GNN VC		HILS		HtWIS		NuMWVC		struction		M^{2}WIS + S		M^{2}WIS	
ca2010	-	-	2 873,13	1 979 294	1 470,66	1 980 551	0,16	1 951 444	916,03	1 951 636	1,47	**1 989 209**	5,79	**1 989 209**	8 652,80	1 989 106
fl2010	36 000,00	649 704	84,54	682 793	932,09	684 214	0,10	673 852	563,46	676 284	0,83	**686 985**	3,82	**686 985**	5 130,54	686 946
ga2010	159,38	**76 316**	492,05	76 214	59,10	76 297	0,01	75 201	26,71	76 001	0,07	**76 316**	0,34	**76 316**	0,33	**76 316**
il2010	-	-	3 714,46	998 051	1 290,10	996 126	0,14	981 297	740,62	985 582	0,70	**1 001 624**	4,65	**1 001 624**	8 983,65	1 001 415
nh2010	2,28	**26 770**	44,27	26 737	17,20	26 768	<0,01	26 477	4,56	26 687	0,01	**26 770**	0,07	**26 770**	0,05	**26 770**
ri2010	6 477,22	**66 963**	1 008,11	66 672	32,71	66 960	<0,01	65 979	19,66	66 743	0,02	**66 963**	0,15	**66 963**	0,12	**66 963**
overall	branch reduce		GNN VC		HILS		HtWIS		NuMWVC		struction		M^{2}WIS + S		M^{2}WIS	
# best		67/93		46/93		75/93		14/93		0/93		75/93		88/93		88/93
gmean ω		-		16 357		16 683		15 964		-		-		16 688		16 688
gmean t		-		35,30		25,01		0,01		-		-		0,60		0,66

Table 15: Graph properties. **Bold** graphs where used to determine the best parameters. The set *redOsm* are the osm instances used for the metamis comparison reduced using KaMIS [32].

finEl	$\|V\|$	$\|E\|$
body	45 087	327 468
ocean	143 437	819 186
pwt	36 519	289 588
rotor	99 617	1 324 862
sphere	16 386	98 304
mesh	$\|V\|$	$\|E\|$
blob	16 068	48 204
buddha	1 087 716	3 263 148
bunny	68 790	206 034
cow	5 036	14 732
dragon	150 000	450 000
dragonsub	600 000	1 800 000
ecat	684 496	2 053 488
face	22 871	68 108
fandisk	8 634	25 636
feline	41 262	123 786
gameguy	42 623	127 700
gargoyle	20 000	60 000
turtle	267 534	802 356
venus	5 672	17 016
osm	$\|V\|$	$\|E\|$
alabama-1	320	1 162
alabama-2	1 164	38 772
alabama-3	3 504	619 328
alaska-1	31	62
alaska-2	54	312
alaska-3	86	950
arkansas-1	26	38
arkansas-2	55	466
arkansas-3	103	2 752
california-1	77	260
california-2	231	6 148
california-3	587	55 072
canada-1	189	480
canada-2	449	5 894
canada-3	943	40 482
colorado-1	128	464
colorado-2	283	4 052
colorado-3	538	16 730
connec.-1	87	192
connec.-2	211	1 950
connec.-3	367	7 538
delaware-1	2	2
delaware-2	3	6
delaware-3	5	18
d.o.c.-1	2 500	49 302
d.o.c.-2	13 597	3 219 590
d.o.c.-3	46 221	55 458 274
florida-1	475	2 554
florida-2	1 254	33 872
florida-3	2 985	308 086
georgia-1	294	868
georgia-2	746	15 506
georgia-3	1 680	148 252
greenland-1	77	682
greenland-2	686	100 436
greenland-3	4 986	7 304 722
hawaii-1	411	2 846
hawaii-2	2 875	530 316
hawaii-3	28 006	98 889 842
idaho-1	136	416
idaho-2	552	70 442
idaho-3	4 064	7 848 160
illinois-1	113	404
illinois-2	261	4 276
indiana-1	2	2
indiana-2	2	2
indiana-3	4	12

osm	$\|V\|$	$\|E\|$
iowa-1	90	328
iowa-2	155	1 908
kansas-1	190	800
kansas-2	602	32 948
kansas-3	2 732	1 613 824
kentucky-1	381	4 804
kentucky-2	2 453	1 286 856
kentucky-3	19 095	119 067 260
louisiana-1	157	362
louisiana-2	436	6 222
louisiana-3	1 162	74 154
maine-1	38	58
maine-2	81	486
maine-3	143	1 700
maryland-1	104	432
maryland-2	316	9 430
maryland-3	1 018	190 830
massach.-1	413	2 178
massach.-2	1 339	70 898
massach.-3	3 703	1 102 982
mexico-1	175	716
mexico-2	516	18 822
mexico-3	1 096	94 262
michigan-1	133	224
michigan-2	241	1 500
michigan-3	376	4 918
minnesota-1	86	272
minnesota-2	253	5 160
minnesota-3	683	68 376
mississippi-1	74	120
mississippi-2	151	732
mississippi-3	242	2 232
missouri-1	10	12
missouri-2	13	24
missouri-3	17	48
montana-1	109	388
montana-2	307	10 308
montana-3	837	138 586
nebraska-1	40	92
nebraska-2	93	1 468
nebraska-3	145	4 336
nevada-1	89	186
nevada-2	242	3 062
nevada-3	569	30 032
new-hamp.-1	195	604
new-hamp.-2	514	6 738
new-hamp.-3	1 107	36 042
new-jersey-1	4	12
new-jersey-2	4	12
new-jersey-3	4	12
new-mex.-1	3	6
new-mex.-2	3	6
new-mex.-3	3	6
new-york-1	42	236
new-york-2	224	12 798
new-york-3	837	177 456
north-car.-1	93	300
north-car.-2	398	20 232
north-car.-3	1 557	473 478
ohio-1	78	192
ohio-2	211	3 630
ohio-3	482	22 752
oregon-1	381	1 992
oregon-2	1 325	115 034
oregon-3	5 588	5 825 402
penns.-1	193	552
penns.-2	521	7 624
penns.-3	1 148	52 928

osm	$\|V\|$	$\|E\|$
puerto-rico-1	60	126
puerto-rico-2	165	2 570
puerto-rico-3	494	53 852
rhode-is.-1	455	3 946
rhode-is.-2	2 866	590 976
rhode-is.-3	15 124	25 244 438
south-car.-1	75	138
south-car.-2	165	1 426
south-car.-3	317	9 016
tennessee-1	49	78
tennessee-2	100	836
tennessee-3	212	6 430
utah-1	230	618
utah-2	589	9 384
utah-3	1 339	85 744
vermont-1	128	836
vermont-2	766	75 214
vermont-3	3 436	2 272 328
virginia-1	570	2 960
virginia-2	2 279	120 080
virginia-3	6 185	1 331 806
washington-1	713	4 632
washington-2	3 025	304 898
washington-3	10 022	4 692 426
w-virg.-1	65	300
w-virg.-2	317	16 656
w-virg.-3	1 185	251 240
wisconsin-1	54	102
wisconsin-2	89	438
wisconsin-3	136	1 176
wyoming-1	7	22
wyoming-2	8	32
wyoming-3	12	84
snap	$\|V\|$	$\|E\|$
as-skitter	1 696 415	22 190 596
ca-AstroPh	18 772	396 100
ca-CondMat	23 133	186 878
ca-GrQc	5 242	28 968
ca-HepPh	12 008	236 978
ca-HepTh	9 877	51 946
com-amazon	334 863	1 851 738
com-youtube	1 134 890	5 975 248
email-Enron	36 692	367 662
email-EuAll	265 214	728 962
loc-gowalla	196 591	1 900 654
p2p-G.04	10 876	79 988
p2p-G.05	8 846	63 678
p2p-G.06	8 717	63 050
p2p-G.08	6 301	41 554
p2p-G.09	8 114	52 026
p2p-G.24	26 518	130 738
p2p-G.25	22 687	109 410
p2p-G.30	36 682	176 656
p2p-G.31	62 586	295 784
roadNet-CA	1 965 206	5 533 214
roadNet-PA	1 088 092	3 083 796
roadNet-TX	1 379 917	3 843 320
soc-Ep.1	75 879	811 480
soc-LiveJ.1	4 847 571	85 702 474
soc-Sl.0811	77 360	938 360
soc-Sl.0902	82 168	1 008 460
soc-p.-rel.	1 632 803	44 603 928
web-BS.	685 230	13 298 940
web-Google	875 713	8 644 102
web-ND.	325 729	2 180 216
web-Stanford	281 903	3 985 272
wiki-Talk	2 394 385	9 319 130
wiki-Vote	7 115	201 524

ssmc	$\|V\|$	$\|E\|$
ca2010	710 145	3 489 366
fl2010	484 481	2 346 294
ga2010	291 086	1 418 056
il2010	451 554	2 164 464
nh2010	48 837	234 550
ri2010	25 181	125 750
redOsm	$\|V\|$	$\|E\|$
alabama-3	1 614	117 426
d.o.c.-2	6 360	592 457
d.o.c.-3	33 367	17 459 296
florida-3	1 069	62 088
greenland-3	3 942	2 348 539
hawaii-3	24 436	40 724 109
idaho-3	3 208	2 864 466
kansas-3	1 605	408 108
kentucky-3	16 871	54 160 431
massach.-3	2 008	373 537
north-car.-3	1 178	189 362
oregon-3	3 670	1 958 180
rhode-is.-2	1 103	81 688
rhode-is.-3	13 031	11 855 557
vermont-3	2 630	811 482
virginia-3	3 867	485 330
washington-3	8 030	2 120 696