Exactly Solving the Maximum Weight Independent Set Problem on Large Real-World Graphs*

Sebastian Lamm†, Christian Schulz‡, Darren Strash§, Robert Williger¶, Huashuo Zhang∥

Abstract

One powerful technique to solve NP-hard optimization problems in practice is branch-and-reduce search—which is branch-and-bound that intermixes branching with reductions to decrease the input size. While this technique is known to be very effective in practice for unweighted problems, very little is known for weighted problems, in part due to a lack of known effective reductions. In this work, we develop a full suite of new reductions for the maximum weight independent set problem and provide extensive experiments to show their effectiveness in practice on real-world graphs of up to millions of vertices and edges.

Our experiments indicate that our approach is able to outperform existing state-of-the-art algorithms, solving many instances that were previously infeasible. In particular, we show that branch-and-reduce is able to solve a large number of instances up to two orders of magnitude faster than existing (inexact) local search algorithms—and is able to solve the majority of instances within 15 minutes. For those instances remaining infeasible, we show that combining kernelization with local search produces higher-quality solutions than local search alone.

1 Introduction

The maximum weight independent set problem is an NP-hard problem that has attracted much attention in the combinatorial optimization community, due to its difficulty and its importance in many fields. Given a graph $G = (V, E, w)$ with weight function $w : V \rightarrow \mathbb{R}^+$, the goal of the maximum weight independent set problem is to compute a set of vertices $I \subseteq V$ with maximum total weight, such that no vertices in $I$ are adjacent to one another. Such a set is called a maximum weight independent set (MWIS). The maximum weight independent set problem has applications spanning many disciplines, including signal transmission, information retrieval, and computer vision [6]. As a concrete example, weighted independent sets are vital in labeling strategies for maps [7,19], where the objective is to maximize the number of visible non-overlapping labels on a map. Here, the maximum weight independent set problem is solved in the label conflict graph, where any two overlapping labels are connected by an edge and vertices have a weight proportional to the city’s population.

Similar to their unweighted counterparts, a maximum weight independent set $I \subseteq V$ in $G$ is a maximum weight clique in $G$ (the complement of $G$), and $V \setminus I$ is a minimum vertex cover of $G$ [13,34]. Since all of these problems are NP-hard [18], heuristic algorithms are often used in practice to efficiently compute solutions of high quality on large graphs [13,25,27,30].

Small graphs with hundreds to thousands of vertices may often be solved in practice with traditional branch-and-bound methods [5,6,10,32]. However, even for medium-sized synthetic instances, the maximum weight independent set problem becomes infeasible. Further complicating the matter is the lack of availability of large real-world test instances — instead, the standard practice is to either systematically or randomly assign weights to vertices in an unweighted graph. Therefore, the performance of exact algorithms on real-world data sets is virtually unknown.

In stark contrast, the unweighted variants can be quickly solved on large real-world instances—even with millions of vertices—in practice, by using kernelization [14,20,34] or the branch-and-reduce paradigm [3]. For those instances that can’t be solved exactly, high-quality (and often exact) solutions can be found by com-
bining kernelization with either local search \cite{31, 30} or evolutionary algorithms \cite{22}.

These algorithms first remove (or fold) whole subgraphs from the input graph while still maintaining the ability to compute an optimal solution from the resulting smaller instance. This so-called \textit{kernel} is then solved by an exact or heuristic algorithm. While these techniques are well understood, and are effective in practice for the unweighted variants of these problems, very little is known about the weighted problems.

While the unweighted maximum independent set problem has many known reductions, we are only aware of one explicitly known reduction for the maximum weight independent set problem: the weighted critical independent set reduction by Butenko and Trukhanov \cite{10}, which has only been tested on small synthetic instances with unit weight (unweighted case). However, it remains to be examined how their weighted reduction performs in practice. There is only one other reduction-like procedure of which we are aware, although it is neither called so directly nor is it explicitly implemented as a reduction. Nogueria et al. \cite{27} introduced the notion of a \textit{\( \omega,1 \)-swap} in their local search algorithm, which swaps a vertex into a solution if its neighbors in the current solution have smaller total weight. This swap is not guaranteed to select a vertex in a true MWIS; however, we show how to transform it into a reduction that does.

\textbf{Our Results.} In this work, we develop a full suite of new reductions for the maximum weight independent set problem and provide extensive experiments to show their effectiveness in practice on real-world graphs of up to millions of vertices and edges. While existing exact algorithms are only able to solve graphs with hundreds of vertices, our experiments show that our approach is able to exactly solve real-world label conflict graphs with thousands of vertices, and other larger networks with synthetically generated vertex weights—all of which are infeasible for state-of-the-art solvers. Further, our branch-and-reduce algorithm is able to solve a large number of instances up to two orders of magnitude faster than existing \textit{inexact} local search algorithms—solving the majority of instances within 15 minutes. For those instances remaining infeasible, we show that combining kernelization with local search produces higher-quality solutions than local search alone.

Finally, we develop new \textit{meta} reductions, which are general rules that subsume traditional reductions. We show that weighted variants of popular unweighted reductions can be explained by two general (and intuitive) rules—which use MWIS search as a subroutine. This yields a simple framework covering many reductions.

\section{Related Work}

We now present important related work on finding high-quality weighted independent sets. This includes exact branch-and-bound algorithms, reduction based approaches, as well as inexact heuristics, e.g. local search algorithms. We then highlight some recent approaches that combine both exact and inexact algorithms.

\subsection{Exact Algorithms}

Much research has been devoted to improve exact branch-and-bound algorithms for the MWIS and its complementary problems. These improvements include different pruning methods and sophisticated branching schemes \cite{5, 6, 28, 32}.

Warren and Hicks \cite{32} proposed three combinatorial branch-and-bound algorithms that are able to quickly solve DIMACS and weighted random graphs. These algorithms use weighted clique covers to generate upper bounds that reduce the search space via pruning. Furthermore, they all use a branching scheme proposed by Balas and Yu \cite{6}. In particular, their first algorithm is an extension and improvement of a method by Babel \cite{5}. Their second one uses a modified version of the algorithm by Balas and Yu that uses clique covers that borrow structural features from the ones by Babel \cite{5}. Finally, their third approach is a hybrid of both previous algorithms. Overall, their algorithms are able to quickly solve instances with hundreds of vertices.

An important technique to reduce the base of the exponent for exact branch-and-bound algorithms are so-called \textit{reduction rules}. Reduction rules are able to reduce the input graph to an irreducible \textit{kernel} by removing well-defined subgraphs. This is done by selecting certain vertices that are provably part of some maximum(-weight) independent set, thus maintaining optimality. We can then extend a solution on the kernel to a solution on the input graph by undoing the previously applied reductions. There exist several well-known reduction rules for the unweighted vertex cover problem (and in turn for the unweighted MIS problem) \cite{3}. However, there are only a few reductions known for the MWIS problem.

One of these was proposed by Butenko and Trukhanov \cite{10}. In particular, they show that every critical weighted independent set is part of a maximum weight independent set. A critical weighted set is a subset of vertices such that the difference between its weight and the weight of its neighboring vertices is maximal for all such sets. They can be found in polynomial time via minimum cuts. Their neighborhood is recursively removed from the graph until the critical set is empty.

As noted by Larson \cite{23}, it is possible that in the unweighted case the initial critical set found by Butenko and Trukhanov might be empty. To prevent
this case, Larson \cite{23} proposed an algorithm that finds a maximum (unweighted) critical independent set. Later Iwata \cite{21} has shown how to remove a large collection of vertices from a maximum matching all at once; however, it is not known if these reductions are equivalent. For the maximum weight clique problem, Cai and Lin \cite{12} give an exact branch-and-bound algorithm that interleaves between clique construction and reductions. We briefly note that their algorithm and reductions are targeted at sparse graphs, and therefore their reductions would likely work well for the MWIS problem on dense graphs—but not on the sparse graphs we consider here.

2.2 Heuristic Algorithms. Heuristic algorithms such as local search work by maintaining a single solution that is gradually improved through a series of vertex deletions, insertion and swaps. Additionally, plateau search allows these algorithms to explore the search space by performing node swaps that do not change the value of the objective function. We now cover state-of-the-art heuristics for both the unweighted and weighted maximum independent set problem.

For the unweighted case, the iterated local search algorithm by Andrade et al. \cite{4} (ARW) is one of the most successful approaches in practice. Their algorithm is based on finding improvements using so-called (1, 2)-swaps that can be found in linear time. Such a swap removes a single vertex from the current solution and inserts two new vertices instead. Their algorithm is able to find (near-)optimal solutions for small to medium-sized instances in milliseconds, but struggles on massive instances with millions of vertices and edges \cite{16}.

Several local search algorithms have been proposed for the maximum weight independent set problem. Most of these approaches interleave a sequence of iterative improvements and plateau search. Further strategies developed for these algorithms include the usage of sub-algorithms for vertex selection \cite{29,30}, tabu mechanisms using randomized restarts \cite{33}, and adaptive perturbation strategies \cite{8}. Local search approaches are often able to obtain high-quality solutions on medium to large instances that are not solvable using exact algorithms. Next, we cover some of the most recent state-of-the-art local search algorithms in greater detail.

The hybrid iterated local search (HILS) by Nogueira et al. \cite{27} extends ARW to the weighted case. It uses two efficient neighborhood structures: $(\omega, 1)$-swaps and weighted $(1, 2)$-swaps. Both of these structures are explored using a variable neighborhood descent procedure. Their algorithm outperforms state-of-the-art algorithms on well-known benchmarks and is able to find known optimal solution in milliseconds.

Recently, Cai et al. \cite{13} proposed a heuristic algorithm for the weighted vertex cover problem that was able to derive high-quality solution for a variety of large real-world instances. Their algorithm is based on a local search algorithm by Li et al. \cite{25} and uses iterative removal and maximization of a valid vertex cover.

2.3 Hybrid Algorithms. In order to overcome the shortcomings of both exact and inexact methods, new approaches that combine reduction rules with heuristic local search algorithms were proposed recently \cite{16,22}. A very successful approach using this paradigm is the reducing-peeling framework proposed by Chang et al. \cite{14} which is based on the techniques proposed by Lamm et al. \cite{22}. Their algorithm works by computing a kernel using practically efficient reduction rules in linear and near-linear time. Additionally, they provide an extension of their reduction rules that is able to compute good initial solutions for the kernel. In particular, they greedily select vertices that are unlikely to be in a large independent set, thereby opening up the reduction space again. Thus, they are able to significantly improve the performance of the ARW local search algorithm that is applied on the kernelized graph. To speed-up kernelization, Hespe et al. \cite{20} proposed a shared-memory algorithm using partitioning and parallel bipartite matching.

3 Preliminaries

Let $G = (V = \{0, \ldots, n-1\}, E, w)$ be an undirected graph with $n = |V|$ nodes and $m = |E|$ edges. $w : V \rightarrow \mathbb{R}^+$ is the real-valued vertex weighting function such that $w(v) \in \mathbb{R}^+$ for all $v \in V$. Furthermore, for a non-empty set $S \subseteq V$ we use $w(S) = \sum_{v \in S} w(v)$ and $|S|$ to denote the weight and size of $S$. The set $N(v) = \{u : \{v, u\} \in E\}$ denotes the neighbors of $v$. We further define the neighborhood of a set of nodes $U \subseteq V$ to be $N(U) = \cup_{v \in U} N(v) \setminus U$. $N[v] = N(v) \cup \{v\}$, and $N[U] = N(U) \cup U$. A graph $H = (V_H, E_H)$ is said to be a subgraph of $G = (V,E)$ if $V_H \subseteq V$ and $E_H \subseteq E$. We call $H$ an induced subgraph when $E_H = \{\{u,v\} \in E : u,v \in V_H\}$. For a set of nodes $U \subseteq V$, $G[U]$ denotes the subgraph induced by $U$. The complement of a graph is defined as $\overline{G} = (V, \overline{E})$, where $\overline{E}$ is the set of edges not present in $G$. An independent set is a set $I \subseteq V$, such that all nodes in $I$ are pairwise non-adjacent. An independent set is maximal if it is not a subset of any larger independent set. Furthermore, an independent set $I$ has maximum weight if there is no heavier independent set, i.e. there exists no independent set $I'$ such that $w(I) < w(I')$.

The weight of a maximum independent set of $G$ is denoted by $\alpha_w(G)$. The maximum weight independent
3.1 Unweighted Reductions. In this section, we describe reduction rules for the unweighted maximum independent set problem. These reductions perform exceptionally well in practice and form the basis of our weighted reductions described in Section 5.

Vertex Folding [15]. Vertex folding was first introduced by Chen et al. [15] to reduce the theoretical running time of exact branch-and-bound algorithms for the maximum independent set problem. This reduction is applied whenever there is a vertex \( v \) with degree two and non-adjacent neighbors \( u \) and \( w \). Chen et al. [15] then showed that either \( v \) or both \( u \) and \( w \) are in some maximum independent set. Thus, we can contract \( u, v, \) and \( w \) into a single vertex \( v' \) (called a fold), forming a new graph \( G' \). Then \( \alpha(G) = \alpha(G') + 1 \) and after finding a MIS \( I' \) of \( G' \), if \( v' \notin I' \) then \( I = I' \cup \{ v \} \) is an MIS of \( G \), otherwise \( I = (I' \setminus \{ v' \}) \cup \{ u, w \} \).

Isolated Vertex Removal [11]. An isolated vertex, also called a simplicial vertex, is a vertex \( v \) whose neighborhood forms a clique. That is, there is a clique \( C \) such that \( V(C) \cap N[v] = N[v] \); this clique is called an isolated clique. Since \( v \) has no neighbors outside of the clique, by a cut-and-paste argument, it must be in some maximum independent set. Therefore, we can add \( v \) to the maximum independent set we are computing, and remove \( v \) and \( C \) from the graph. Isolated vertex removal was shown by Butenko et al. [11] to be highly effective in finding exact maximum independent sets on graphs derived from error-correcting codes. In order to work efficiently in practice, this reduction is typically limited to cliques with size at most 2 or 3 [14, 10].

Although Chang et al. [14] showed that the domination reduction (described below) captures the isolated vertex removal reduction, that reduction must be applied several times: once per neighbor in the clique.

Twin. Two non-adjacent vertices \( u \) and \( v \) are called twins if \( N(u) = N(v) \). Note that either both \( u \) and \( v \) are in some MIS, or some subset of \( N(u) \) is in some MIS. If \( |N(u)| = |N(v)| = 3 \), then either \( u \) and \( v \) are together or at least two vertices of \( N(u) \) must be in an MIS. The following case of the reduction is relevant to our result: If \( N(u) \) is independent, then we can fold \( u, v, \) and \( N(v) \) into a single vertex \( v' \) and \( \alpha(G) = \alpha(G') + 2 \).

Domination [17]. Given two vertices \( u \) and \( v \), \( u \) is said to dominate \( v \) if and only if \( N[u] \supseteq N[v] \). In this case there is an MIS in \( G \) that excludes \( u \) and therefore, \( u \) can be removed from the graph.

Critical Independent Set. A subset \( U_c \subseteq V \) is called a critical set if \( |U_c| - |N(U_c)| = \max\{|U| - |N(U)| : U \subseteq V\} \). Likewise, an independent set \( I_c \subseteq V \) is called a critical independent set if \( |I_c| - |N(I_c)| = \max\{|I| - |N(I)| : I \text{ is an independent set of } G\} \). Butenko and Trukhanov [10] show that any critical independent set is a subset of a maximum independent set. They then continue to develop a reduction that uses critical independent sets which can be computed in polynomial time. In particular, they start by finding a critical set in \( G \) by using a reduction to the maximum matching problem in a bipartite graph [2]. In turn, this problem can then be solved in \( O(|V| \sqrt{|E|}) \) time using the Hopcroft-Karp algorithm. They then obtain a critical independent set by setting \( I_c = U_c \setminus N(U_c) \). Finally, they can remove \( I_c \) and \( N(I_c) \) from \( G \).

Linear Programming (LP) Relaxation. The LP-based reduction rule by Nemhauser and Trotter [20], is based on an LP relaxation of the vertex cover problem:

\[
\begin{align*}
\text{minimize} & \quad \sum_{v \in V} x_v \\
\text{subject to} & \quad x_u + x_v \geq 1 \quad \text{for } (u, v) \in E, \\
& \quad x_v \geq 0 \quad \text{for } v \in V.
\end{align*}
\]

Nemhauser and Trotter [20] showed that there exists an optimal half-integral solution for this problem. Additionally, they prove that if a variable \( x_v \) takes an integer value in an optimal solution, then there exists an optimal integer solution where \( x_v \) has the same value. Just as in the critical set reduction, they use a reduction to the maximum bipartite matching problem to compute a half-integral solution. To develop a reduction rule for the vertex cover problem, they afterwards fix the integral part of their solution and output the remaining graph. Their approach was successively improved by Iwata et al. [21] and was shown to be effective in practice by Akiba and Iwata [23].

3.2 Critical Weighted Independent Set Reduction. We now briefly describe the critical weighted independent set reduction, which is the only reduction that has appeared in the literature for the weighted
maximum independent set problem. Similar to the unweighted case, a subset $U_c \subseteq V$ is called a critical weighted independent set if $w(U_c) - w(N(U_c)) = \max\{w(U) - w(N(U)) : U \subseteq V\}$. A weighted independent set $I_c \subseteq V$ is called a critical weighted independent set (CWIS) if $w(I_c) - w(N(I_c)) = \max\{w(I) - w(N(J)) : I \text{ is an independent set of } G\}$. Butenko and Trukhanov \cite{Butenko07} show that any CWIS is a subset of a maximum weight independent set. Additionally, they propose a weighted critical set reduction which works similar to its unweighted counterpart. However, instead of computing a maximum matching in a bipartite graph, a critical weighted set is obtained by solving the selection problem \cite{ChekuriF05}. The problem is equivalent to finding a maximum weight independent set. Additionally, they consider all vertices in the graph. For a proof of correctness, see the paper by Butenko and Trukhanov \cite{Butenko07}.

**Reduction 1. (CWIS Reduction)** Let $U \subseteq V$ be a critical weighted independent set of $G$. Then $U$ is in some MWIS of $G$. We set $G' = G[V \setminus N[U]]$ and $\alpha_w(G) = \alpha_w(G') + w(U)$.

4 Efficient Branch-and-Reduce

We now describe our branch-and-reduce framework in full detail. This includes the pruning and branching techniques that we use, as well as other algorithm details. An overview of our algorithm can be found in Algorithm 1. To keep the description simple, the pseudocode describes the algorithm such that it outputs the weight of a maximum weight independent set in the graph. However, our algorithm is implemented to actually output the maximum weight independent set. Throughout the algorithm we maintain the current solution weight as well as the best solution weight. Our algorithm applies a set of reduction rules before branching on a node. We describe these reductions in the following section. Initially, we run a local search algorithm on the reduced graph to compute a lower bound on the solution weight, which later helps pruning the search space. We then prune the search by excluding unnecessary parts of the branch-and-bound tree to be explored. If the graph is not connected, we separately solve each connected component. If the graph is connected, we branch into two cases by applying a branching rule. If our algorithm does not finish with a certain time limit, we use the currently best solution and improve it using a greedy algorithm. More precisely, our algorithm sorts the vertices in decreasing order of their weight and adds vertices in that order if feasible. We give a detailed description of the subroutines of our algorithm below.

**Algorithm 1** Branch-and-Reduce Algorithm for MWIS

```plaintext```
```
input graph $G = (V, E)$, current solution weight $c$ (initially zero), best solution weight $W$ (initially zero)

procedure Solve($G, c, W$)
  $(G, c) \leftarrow$ Reduce($G, c$)
  if $W = 0$ then $W \leftarrow c + ILS(G)$
  if $c + $UpperBound$(G) \leq W$ then return $W$
  if $G$ is empty then return $\max\{W, c\}$
  if $G$ is not connected then
    for all $G_i \in$ Components($G$) do
      $c \leftarrow c + $Solve($G_i, 0, 0$)
    return $\max(W, c)$
  $(G_1, c_1), (G_2, c_2) \leftarrow$ Branch($G, c$)
  {Run 1st case, update currently best solution}
  $W \leftarrow $Solve($G_1, c_1, W$)
  {Use updated $W$ to shrink the search space}
  $W \leftarrow $Solve($G_2, c_2, W$)

return $W$
```
```

4.1 Incremental Reductions. Our algorithm starts by running all reductions that are described in the following section. Following the lead of previous works \cite{Curtis90, Eppstein90, Hochbaum95}, we apply our reductions incrementally. For each reduction rule, we check if it is applicable to any vertex of the graph. After the checks for the current reduction are completed, we continue with the next reduction if the current reduction has not changed the graph. If the graph was changed, we go back to the first reduction rule and repeat. Most of the reductions we introduce in the following section are local: if a vertex changes, then we do not need to check the entire graph to apply the reduction again, we only need to consider the vertices whose neighborhood has changed since the reduction was last applied. The critical weighted independent set reduction defined above is the only global reduction that we use; it always considers all vertices in the graph.

For each of the local reductions there is a queue of changed vertices associated. Every time a node or its neighborhood is changed it is added to the queues of all reductions. When a reduction is applied only the vertices in its associated queue have to be checked for applicability. After the checks are finished for a particular reduction its queue is cleared. Initially, the queues of all reductions are filled with every vertex in the graph.

4.2 Pruning. Exact branch-and-bound algorithms for the MWIS problem often use weighted clique covers to compute an upper bound for the optimal solution \cite{ChekuriF05}. A weighted clique cover of $G$ is a collection of (possibly overlapping) cliques $C_1, \ldots, C_k \subseteq V$, with associated weights $W_1, \ldots, W_k$ such that $C_1 \cup C_2 \cup \cdots \cup$
$C_k = V$, and for every vertex $v \in V$, $\sum_{i:v \in C_i} W_i \geq w(v)$. The weight of a clique cover is defined as $\sum_{i=1}^k W_i$ and provides an upper bound on $\alpha_w(G)$. This holds because the intersection of a clique and any IS of $G$ is either a single vertex or empty. The objective then is to find a clique cover of small weight. This can be done using an algorithm similar to the coloring method of Brelaz [9]. However, this method can become computationally expensive since its running time is dependent on the maximum weight of the graph [32]. Thus, we use a faster method to compute a weighted clique cover which is similar to the one used in Akiba and Iwata [3].

We begin by sorting the vertices in descending order of their weight (ties are broken by selecting the vertex with higher degree). Next, we initiate an empty set of cliques $\mathcal{C}$. We then iterate over the sorted vertices and search for the clique with maximum weight which it can be added to. If there are no candidates for insertion, we insert a new single vertex clique to $\mathcal{C}$ and assign it the weight of the vertex. Afterwards the vertex is marked as processed and we continue with the next one. Computing a weighted clique cover using this algorithm has a linear running time independent of the maximum weight. Thus, we are able to obtain a bound much faster. However, this algorithm produces a higher weight clique cover than the method of Brelaz [3][9].

In addition to computing an upper bound, we also add an additional lower bound using a heuristic approach. In particular, we run a modified version of the ILS by Andrade et al. [4] that is able to handle vertex weights for a fixed fraction of our total running time. This lower bound is computed once after we apply the ILS by Andrade et al. [4] that is able to handle the ILS by Andrade et al. [4] that is able to handle

4.3 Connected Components. Solving the maximum weight independent set problem for a graph $G$ is equal to solving the problem for all connected components $G_1, \ldots, G_c$ of $G$ and then combining the solution sets $\mathcal{I}_1, \ldots, \mathcal{I}_c$ to form a solution $\mathcal{I}$ for $G$: $\mathcal{I} = \bigcup_{i=1}^c \mathcal{I}_i$. We leverage this property by checking the connectivity of $G$ after each completed round of reduction applications. If the graph disconnects due to branching or reductions then we apply our branch-and-reduce algorithm recursively to each of the connected components and combine their solutions afterwards. This technique can reduce the size of the branch-and-bound tree significantly on some instances.

4.4 Branching. Our algorithm has to pick a branching order for the remaining vertices in the graph. Initially, vertices are sorted in non-decreasing order by degree, with ties broken by weight. Throughout the algorithm, the next vertex to be chosen is the highest vertex in the ordering. This way our algorithm quickly eliminates the largest neighborhoods and makes the problem “simpler”.

5 Weighted Reduction Rules

We now develop a comprehensive set of reduction rules for the maximum weight independent set problem. We first introduce two meta reductions, which we then use to instantiate many efficient reductions similar to already-known unweighted reductions.

5.1 Meta Reductions. There are two operations that are commonly used in reductions: vertex removal and vertex folding. In the following reductions, we show general ways to detect when these operations can be applied in the neighborhood of a vertex.

**Neighbor Removal.** In our first meta reduction, we show how to determine if a neighbor can be outright removed from the graph. We call this reduction the neighbor removal reduction. (See Figure 1)

**Reduction 2. (Neighbor Removal)** Let $v \in V$. For any $u \in N(v)$, if $\alpha_w(G[N(v) \setminus N[u]]) + w(u) \leq w(v)$, then $u$ can be removed from $G$, as there is some MWIS of $G$ that excludes $u$, and $\alpha_w(G) = \alpha_w(G[V \setminus \{u\}])$.

**Proof.** Let $\mathcal{I}$ be an MWIS of $G$. We show by a cut-and-paste argument that if $u \in \mathcal{I}$ then there is another MWIS $\mathcal{I}'$ that contains $v$ instead. Let $u \in N(v)$, and suppose that $\alpha_w(G[N(v) \setminus N[u]]) + w(u) \leq w(v)$. There are two cases, if $u$ is not in $\mathcal{I}$ then it is safe to remove. Otherwise, suppose $u \in \mathcal{I}$. Then $v \in N(u)$ is not in $\mathcal{I}$, and $w(\mathcal{I} \cap N(v)) = w(\mathcal{I} \cap (N(v) \setminus N[u]) \cup \{u\}) = \alpha_w(G[N(v) \setminus N[u]]) + w(u) = w(v)$; otherwise we can swap $\mathcal{I} \cap N(v)$ for $v$ in $\mathcal{I}$ obtaining an independent set of larger weight. Thus $\mathcal{I}' = (\mathcal{I} \setminus N(v)) \cup \{v\}$ is an MWIS of $G$ excluding $u$ and $\alpha_w(G) = \alpha_w(G[V \setminus \{u\}])$.

**Neighborhood Folding.** For our next meta reduction, we show a general condition for folding a vertex with its neighborhood. We first briefly describe the intuition behind the reduction. Consider $v$ and its neighborhood $N(v)$. If $N(v)$ has a unique independent set $\mathcal{I}_{N(v)}$ with weight larger than $w(v)$, then we only need to consider two independent sets: independent sets that contain $v$ or $\mathcal{I}_{N(v)}$. Otherwise, any other independent set in $N(v)$ can be swapped for $v$ and achieve higher overall weight. By folding $v$ with $\mathcal{I}_{N(v)}$, we can solve the remaining graph and then decide which of the two options will give an MWIS of the graph. (See Figure 1)

**Reduction 3. (Neighborhood Folding)** Let $v \in V$, and suppose that $N(v)$ is independent. If $w(N(v)) >$
w(v), but \( w(N(v)) - \min_{u \in N(v)} \{ w(u) \} < w(v) \), then fold \( v \) and \( N(v) \) into a new vertex \( v' \) with weight \( w(v') = w(N(v)) - w(v) \). Let \( I' \) be an MWIS of \( G' \), then we construct an MWIS \( I \) of \( G \) as follows: If \( v' \in I' \), then \( I = (I' \setminus \{ v' \}) \cup N(v) \), otherwise if \( v \in I' \) then \( I = I' \cup \{ v \} \). Furthermore, \( \alpha_w(G) = \alpha_w(G') + w(v) \).

**Proof.** Proof can be found in Appendix A.

However, these reductions require solving the MWIS problem on the neighborhood of a vertex, and therefore may be as expensive as computing an MWIS on the input graph. We next show how to use these meta reductions to develop efficient reductions.

### 5.2 Efficient Weighted Reductions

We now construct new efficient reductions using the just defined meta reductions.

**Neighborhood Removal.** In their HILLS local search algorithm, Nogueira et al. [24] introduced the notion of a “\( (\omega, 1) \)-swap”, which swaps a vertex \( v \) into a solution if its neighbors in the current solution \( I \) have weight \( w(N(v) \cap I) < w(v) \). This can be transformed into what we call the neighborhood removal reduction.

**Reduction 4. (NEIGHBORHOOD REMOVAL)** For any \( v \in V \), if \( w(v) \geq w(N(v)) \) then \( v \) is in some MWIS of \( G \). Let \( G' = G[V \setminus N(v)] \) and \( \alpha_w(G) = \alpha_w(G') + w(v) \).

**Proof.** Since \( w(v) \geq w(N(v)) \), \( \forall u \in N(v) \) we have that

\[
\alpha_w(G[N(v) \cap N(u)]) + w(u) \leq w(N(v)) \leq w(v).
\]

Then we can remove all \( u \in N(v) \) and are left with \( v \) in its own component. Calling this graph \( G' \), we have that \( v \) is in some MWIS and \( \alpha_w(G) = \alpha_w(G') + w(v) \).

For the remaining reductions, we assume that the neighborhood removal reduction has already been applied. Thus, \( \forall v \in V, w(v) < w(N(v)) \).

**Weighted Isolated Vertex Removal.** Similar to the (unweighted) isolated vertex removal reduction, we now argue that an isolated vertex is in some MWIS—if it has highest weight in its clique.

\[
\forall u \in N(v) : w(u) := w(u) - w(v)
\]

\[
\forall u \in N(v) : w(u) \geq w(u) \quad \text{(isolated)}
\]

\[
\forall u \in N(v) : w(u) < w(u) \quad \text{(non-isolated)}
\]

**Figure 2:** Isolated weight transfer

**Reduction 5. (ISOLATED VERTEX REMOVAL.)** Let \( \forall v \in V \) be isolated and \( w(v) \geq \max_{u \in N(v)} w(u) \). Then \( v \) is in some MWIS of \( G \). Let \( G' = G[V \setminus N(v)] \) and \( \alpha_w(G) = \alpha_w(G') + w(v) \).

**Proof.** Since \( N(v) \) is a clique, \( \forall u \in N(v) \) we have that

\[
\alpha_w(G[N(v) \cap N(u)]) \leq \alpha_w(N(v)) = \max_{u \in N(v)} \{ w(u) \} \leq w(v).
\]

Similar to neighborhood removal, remove all \( u \in N(v) \) producing \( G' \) and \( \alpha_w(G) = \alpha_w(G') + w(v) \).

**Isolated Weight Transfer.** Given its weight restriction, the weighted isolated vertex removal reduction may be ineffective. We therefore introduce a reduction that supports more liberal vertex removal.

**Reduction 6. (ISOLATED WEIGHT TRANSFER)** Let \( \forall v \in V \) be isolated, and suppose that the set of isolated vertices \( S(v) \subseteq N(v) \) is such that \( \forall u \in S(v), w(v) \geq w(u) \). We

(i) remove all \( u \in N(v) \) such that \( w(u) \leq w(v) \), and let the remaining neighbors be denoted by \( N'(v) \),

(ii) remove \( v \) and \( \forall x \in N'(v) \) set its new weight to \( w'(x) = w(x) - w(v) \), and let the resulting graph be denoted by \( G' \). Then \( \alpha_w(G) = w(v) + \alpha_w(G') \) and an MWIS \( I \) of \( G \) can be constructed from an MWIS \( I' \) of \( G' \) as follows: if \( I' \cap N'(v) = \emptyset \) then \( I = I' \cup \{ v \} \), otherwise \( I = I' \).
Proof. Proof can be found in Appendix A.

**Weighted Vertex Folding.** Similar to the unweighted vertex folding reduction, we show that we can fold vertices with two non-adjacent neighbors—however, not all weight configurations permit this.

**Reduction 7. (Vertex Folding)** Let \( v \in V \) have \( d(v) = 2 \), such that \( v \)’s neighbors \( u, x \) are not adjacent. If \( w(v) < w(u) + w(x) \) but \( w(v) \geq \max\{w(u), w(x)\} \), then we fold \( v, u, x \) into vertex \( v' \) with weight \( w(v') = w(u) + w(x) - w(v) \) forming a new graph \( G' \). Then \( \alpha_w(G) = \alpha_w(G') + w(v) \). Let \( I' \) be an MWIS of \( G' \). If \( v' \in I' \) then \( I = (I' \setminus \{v\}) \cup \{u, x\} \) is an MWIS of \( G \). Otherwise, \( I = I' \cup \{v\} \) is an MWIS of \( G \).

Proof. Apply neighborhood folding to \( v \).

**Weighted Twin.** The twin reduction, as described by Akiba and Iwata [3] for the unweighted case, works for twins with 3 common neighbors. We describe our variant in the same terms, but note that the reduction supports an arbitrary number of common neighbors.

**Reduction 8. (Twin)** Let vertices \( u \) and \( v \) have independent neighborhoods \( N(u) = N(v) = \{p, q, r\} \). We have two cases:

(i) If \( w(\{u, v\}) \geq w(\{p, q, r\}) \), then \( u \) and \( v \) are in some MWIS of \( G \). Let \( G' = G[V \setminus N[\{u, v\}]] \).

(ii) If \( w(\{u, v\}) < w(\{p, q, r\}) \), but \( w(\{u, v\}) > w(\{p, q, r\}) - \min_{x \in \{p, q, r\}} w(x) \), then we can fold \( u, v, p, q, r \) into a new vertex \( v' \) with weight \( w(v') = w(u) + w(v) \).

Proof. We show by a cut-and-paste argument that there is an MWIS of \( G \) excluding \( u \). Let \( I \) be an MWIS of \( G \). If \( u \) is not in \( I \) then we are done. Otherwise, suppose \( u \in I \). Then it must be the case that \( w(v) = w(u) \), otherwise \( I' = (I \setminus \{u\}) \cup \{v\} \) is an independent set with weight larger than \( I \). Thus, \( I' \) is an MWIS of \( G \) excluding \( u \), and \( \alpha_w(G) = \alpha_w(G[V \setminus \{u\}]) \).

**6 Experimental Evaluation**

We now compare the performance of our branch-and-reduce algorithm to existing state-of-the-art algorithms on large real-world graphs. Furthermore, we examine how our reduction rules can drastically improve the quality of existing heuristic approaches.
6.1 Methodology and Setup. All of our experiments were run on a machine with four Octa-Core Intel Xeon E5-4640 processors running at 2.4 GHz, 512 GB of main memory, 420 MB L3-Cache and 48256 KB L2-Cache. The machine runs Ubuntu 14.04.3 and Linux kernel version 3.13.0-77. All algorithms were implemented in C++11 and compiled with g++ version 4.8.4 with optimization flag -O3. Each algorithm was run sequentially for a total of 1000 seconds. We present two kinds of data: (1) the best solution found by each algorithm and the time (in seconds) required to obtain it, (2) convergence plots, which show how the solution quality changes over time. In particular, whenever an algorithm finds a new best independent set \( S \) at time \( t \), it reports a tuple \( (t, |S|) \).

Algorithms Compared. We use two different variants of our branch-and-reduce algorithm. The first variant, called B & R full, uses our full set of reductions each time we branch. The second variant, called B & R dense, omits the execution of the remaining reductions faster than B & R full. In particular, this configuration completely omits the weighted critical set reductions from both the initialization and recursion. Additionally, we also omit the weighted clique reduction from the first reduction call and use a faster version that only considers triangles during recursion. Finally, we do not use the generalized neighborhood folding during recursion. This configuration finds solutions more quickly on dense graphs.

We also include the state-of-the-art heuristics HILS by Nogueria et al. [27] and both versions of DynWVC by Cai et al. [13] (see Section 2 for a short explanation of these algorithms). Finally, we do not include any other exact algorithms (e.g. [10, 32]) as their code is not available. Also note that these exact algorithms are either not tested in the weighted case [10] or the largest instances reported consist of a few hundred vertices [32].

To further evaluate the impact of reductions on existing algorithms, we also propose combinations of the heuristic approaches with reductions (Red + HILS and Red + DynWVC). We do so by first computing a kernel graph using our set of reductions and then run the existing algorithms on the resulting graph.

Instances. We test all algorithms on a large corpus of sparse data sets. For this purpose, we include a set of real-world conflict graphs obtained from Open-StreetMap [11] files of North America, according to the method described by Barth et al. [7]. More specifically, these graphs are generated by identifying map labels with vertices that have a weight corresponding to their importance. Edges are then inserted between vertices if their labels overlap each other. Conflict graphs can also be used in a dynamic setting by associating vertices with intervals that correspond to the time they are displayed. Furthermore, solving the MWIS problem on these graphs eliminates label conflicts and maximizes the importance of displayed labels. Finally, different activity models (AM1, AM2 and AM3) are used to generate different conflict graphs. The instances we use for our experiments are the same ones used by Cai et al. [13]. We omit all instances with less than 1000 vertices from our experiments, as these are easy to solve and our focus is on large scale networks [13].

In addition to the OSM networks, we also include collaboration networks, communication networks, additional road networks, social networks, peer-to-peer networks, and Web crawl graphs from the Stanford Large Network Dataset Repository [24] (SNAP).

These networks are popular benchmark instances commonly used for the maximum independent set problem [3, 16, 22]. However, all SNAP instances are unweighted and comparable weighted instances are very scarce. Therefore, a common approach in literature is to assign vertex weights uniformly at random from a fixed size interval [13, 25]. To keep our results in line with existing work, we thus decided to select vertex weights uniformly at random from [1, 200]. Basic properties of our benchmark instances can be found in Table 4.

6.2 Comparison with State-of-the-Art. A representative sample of our experimental results for the OSM and SNAP networks is presented in Table 1. For a full overview of all instances, we refer to Table 4 (OSM) and Table 5 (SNAP) respectively. For each instance, we list the best solution computed by each algorithm \( w_{\text{Algo}} \) and the time in seconds required to find it \( t_{\text{Algo}} \). For each data set, we highlight the best solution found across all algorithms in bold. Additionally, if any version of our algorithm is able to find an exact solution, the corresponding row is highlighted in gray. Finally, recall that our algorithm computes a solution on unsolved instances once the time-limit is reached by additionally running a greedy algorithm as post-processing.

Examining the OSM graphs, B & R is able to solve 15 out of the 34 instances we tested. However, HILS is also able to compute a solution with the same weight on all of these instances. Furthermore, HILS obtains a higher or similar quality solution than both versions of DynWVC and B & R for all remaining unsolved instances. Overall, HILS is able to find the best solution on all OSM instances that we tested. Additionally, on
most of these instances it does so significantly faster than all of its competitors. Note though, that neither HILS nor DynWVC provide any optimality guarantees (in contrast to B & R).

Looking at both versions of DynWVC, we see that DynWVC1 performs better than DynWVC2, which is also reported by Cai et al. [13]. Comparing both variants of our branch-and-reduce algorithm, we see that they are able to solve the same instances. Nonetheless, B & R dense is able to compute better solutions on roughly half of the remaining instances. Additionally, it almost always requires significantly less time to achieve its maximum compared to B & R full.

For the SNAP networks, we see that B & R solves 25 of the 31 instances we tested.  Most notable, on seven of these instances where either HILS or DynWVC1 also find a solution with optimal weight, it does so up to two orders of magnitude faster. This difference in performance compared to the OSM networks can be explained by the significantly lower graph density and less uniform degree distribution of the SNAP networks. These structural differences seem to allow for our reduction rules to be applicable more often, resulting in a significantly smaller kernel (as seen in Table 7). This is similar to the behavior of unweighted branch-and-reduce. Therefore, except for a single instance, our algorithm is able to find the best solution on all graphs tested.

Comparing the heuristic approaches, both versions of DynWVC perform better than HILS on most instances, with DynWVC2 often finding better solution than DynWVC1. Nonetheless, HILS finds higher weight solutions than DynWVC1 and DynWVC2.

### 6.3 The Power of Weighted Reductions

We now examine the effect of using reductions to improve existing heuristic algorithms. For this purpose, we compare the combined approaches Red + HILS and Red + DynWVC with their base versions as well as our branch-and-reduce algorithm. Our sample of results for the OSM and SNAP networks is given in Table 2. In addition to the data used in our state-of-the-art comparison, we now also report speedups between the modified and base versions of each local search. Additionally, we give the percentage of instances solved by B & R, as well as the percentage of solutions with optimal weight found by the inexact algorithms compared to B & R. For a full overview of all instances, we refer to Table 6 and Table 7 respectively.

When looking at the speedups for the SNAP graphs, we can see that using reductions allows local search to find optimal solutions orders of magnitude faster. Additionally, they are now able to find an optimal solution more often than without reductions. DynWVC2 in particular achieves an increase of 50% of optimal solutions when using reductions. Overall, we achieve a speedup of up to three orders of magnitude for the SNAP instances. Thus, the additional costs for computing the kernel can be neglected for these instances. However, for the OSM instances our reduction rules are less applicable and reducing the kernel comes at a significant cost compared to the unmodified local searches.

To further examine the influence of using reductions, Figure 3 shows the solution quality over time for

---

3 Using a longer time limit of 48 hours we are able to solve 27 out of 31 instances.
all algorithms and four instances. For additional convergence plots, we refer to Figure 7. For the OSM instances, we can see that initially DynWVC and HILS are able find good quality solutions much faster compared to their combined approaches. However, once the kernel has been computed, regular DynWVC and HILS are quickly outperformed by the hybrid algorithms.

A more drastic change can be seen for the SNAP instances. Instances were both DynWVC and HILS examine poor performance, Red + DynWVC and Red + HILS now rival our branch-and-reduce algorithm and give near-optimal solutions in less time. Thus, using reductions for instances that are too large for traditional heuristic approaches allows for a drastic improvement.

7 Conclusion and Future Work

In this paper, we engineered a new branch-and-reduce algorithm as well as a combination of kernelization with local search for the maximum weight independent set problem. The core of our algorithms are a full suite of new reductions for the maximum weight independent set problem. We performed extensive experiments to show the effectiveness of our algorithms in practice on real-world graphs of up to millions of vertices and edges. Our experimental evaluation shows that our branch-and-reduce algorithm can solve many large real-world instances quickly in practice, and that kernelization has a highly positive effect on local search algorithms.

As HILS often finds optimal solutions in practice, important future works include using this algorithm for the lower bound computation within our branch-and-reduce algorithm. Furthermore, we would like to extend our discussion on the effectiveness of novel reduction rules. In particular, we want to evaluate how much quality we gain from applying each individual rule and how the order we apply them in changes the resulting kernel size.
References

[1] OpenStreetMap. URL https://www.openstreetmap.org

[2] A. A. Ageev. On finding critical independent and vertex sets. *SIAM Journal on Discrete Mathematics*, 7(2):293–295, 1994.

[3] 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.

[4] 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.

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

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

[7] 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.

[8] U. Benlic and J.-K. Hao. Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation*, 219(9):4800–4815, 2013.

[9] D. Brélaz. New methods to color vertices of a graph. *Commun. ACM*, 22(4):251–256, 1979. doi:10.1145/359094.359101.

[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. Butenko, P. Pardalos, I. Sergienko, V. Shylo, and P. Stetsyuk. Estimating the size of correcting codes using extremal graph problems. In C. Pearce and E. Hunt, editors, *Optimization*, volume 32 of *Springer Optimization and Its Applications*, pages 227–243. Springer, 2009. doi:10.1007/978-0-387-98096-6_12.

[12] S. Cai and J. Lin. Fast solving maximum weight clique problem in massive graphs. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 568–574. AAAI Press, 2016. URL http://www.ijcai.org/Proceedings/16/Papers/087.pdf

[13] 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.

[14] L. Chang, W. Li, and W. Zhang. Computing a near-maximum independent set in linear time by reducing-peeling. *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD ’17)*, pages 1181–1196, 2017. doi:10.1145/3035918.3035939.

[15] J. Chen, I. A. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. *Journal of Algorithms*, 41(2):280–301, 2001. doi:10.1006/jagm.2001.1186.

[16] J. Dahlum, S. Lamm, P. Sanders, C. Schulz, D. Strash, and R. F. Werneck. Accelerating local search for the maximum independent set problem. In A. V. Goldberg and A. S. Kulikov, editors, *Experimental Algorithms (SEA 2016)*, volume 9685 of *LNCS*, pages 118–133. Springer, 2016. doi:10.1007/978-3-319-38851-9_9.

[17] F. V. Fomin, F. Grandoni, and D. Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5):25:1–25:32, 2009. doi:10.1145/1552285.1552286.

[18] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some Simplified NP-Complete Problems. In *Proceedings of the 6th ACM Symposium on Theory of Computing*, STOC ’74, pages 47–63. ACM, 1974.

[19] A. Gemsa, M. Nöllenburg, and I. Rutter. Evaluation of labeling strategies for rotating maps. In *Experimental Algorithms (SEA’14)*, volume 8504 of *LNCS*, pages 235–246. Springer, 2014. doi:10.1007/978-3-319-319-3859-9_9.

[20] D. Hespe, C. Schulz, and D. Strash. Scalable kernelization for maximum independent sets. In *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 223–237. SIAM, 2018. doi:10.1137/1.9781611975055.19.

[21] Y. Iwata, K. Oka, and Y. Yoshida. Linear-time FPT algorithms via network flow. In *Proc. 25th ACM-SIAM Symposium on Discrete Algorithms (SODA ’14)*, pages 1749–1761. SIAM, 2014. doi:10.1137/1.9781611973402.127.

[22] 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.
A Omitted Proofs

REDUCTION 3. (NEIGHBORHOOD FOLDING) Let \( v \in V \), and suppose that \( N(v) \) is independent. If \( w(N(v)) > w(v) \), but \( w(N(v)) - \min_{u \in N(v)} w(u) < w(v) \), then fold \( v \) and \( N(v) \) into a new vertex \( v' \) with weight \( w(v') = w(N(v)) - w(v) \). Let \( T' \) be an MWIS of \( G' \), then we construct an MWIS \( T \) of \( G \) as follows: If \( v' \in T' \) then \( T = (T' \setminus \{v'\}) \cup N(v) \), otherwise if \( v \in T' \) then \( T = T' \cup \{v\} \). Furthermore, \( \alpha_w(G) = \alpha_w(G') + w(v) \).

Proof. First note that after folding, the following graphs are identical: \( G'[V' \setminus N_{G'}[v']] = G[V \setminus N[N[v]] \) and \( G'[V' \setminus \{v'\}] = G[V \setminus N[v]] \). Let \( T' \) be an MWIS of \( G' \). We have two cases.

Case 1 \((v' \in T')\): Suppose that \( v' \in T' \). We show that \( w(N(v)) + \alpha_w(G[V \setminus N[N[v]]) \geq w(v) + \alpha_w(G[V \setminus N[v]]) \), which shows that the vertices of \( N(v) \) are together in some MWIS of \( G \). Since \( v' \in T' \), we have that

\[
w(v) + \alpha_w(G') = w(v) + w(v') + \alpha_w(G'[V' \setminus N_{G'}[v']]) = w(v) + w(N(v)) - w(v') + \alpha_w(G'[V' \setminus N_{G'}[v']]) = w(N(v)) + \alpha_w(G[V \setminus N[N[v]])).
\]

But since \( T' \) is an MWIS of \( G' \), we have that

\[
w(v) + \alpha_w(G') \geq w(v) + \alpha_w(G'[V' \setminus \{v'\}]) = w(v) + \alpha_w(G[V \setminus N[v]])\]

Thus, \( w(N(v)) + \alpha_w(G[V \setminus N[N[v]]) \geq w(v) + \alpha_w(G[V \setminus N[v]]) \) and the vertices of \( N(v) \) are together in some MWIS of \( G \). Furthermore, we have that

\[
\alpha_w(G) = w(N(v)) + \alpha_w(G[V \setminus N[N[v]]) = \alpha_w(G') + w(v).
\]

Case 2 \((v' \notin T')\): Suppose that \( v' \notin T' \). We show that \( w(v) + \alpha_w(G[V \setminus N[v]]) \geq w(N(v)) + \alpha_w(G[V \setminus N[N[v]]) \), which shows that \( v \) is in some MWIS of \( G \). Since \( v' \notin T' \), we have that

\[
w(v) + \alpha_w(G') = w(v) + \alpha_w(G'[V' \setminus \{v'\}]) = w(v) + \alpha_w(G[V \setminus N[v]])\]

But since \( T' \) is an MWIS of \( G' \), we have that

\[
w(v) + \alpha_w(G') \geq w(v) + w(v') + \alpha_w(G'[V' \setminus N_{G'}[v']]) = w(v) + w(N(v)) - w(v') + \alpha_w(G[V \setminus N[N[v]]) = w(N(v)) + \alpha_w(G[V \setminus N[N[v]])\]

Thus, \( w(v) + \alpha_w(G[V \setminus N[N[v]]) \geq w(u) + w(x) + \alpha_w(G[V \setminus N[u \cup x]]) \) and \( v \) is in some MWIS of \( G \). Lastly,

\[
\alpha_w(G) = w(v) + \alpha_w(G[V \setminus N[v]]) = \alpha_w(G') + w(v).
\]
Reduction 6. (Isolated weight transfer) Let $v \in V$ be isolated, and suppose that the set of isolated vertices $S(v) \subseteq N(v)$ is such that $\forall u \in S(v), w(u) \geq w(u)$. We

(i) remove all $u \in N(v)$ such that $w(u) \leq w(v)$, and let the remaining neighbors be denoted by $N'(v)$.
(ii) remove $v$ and $\forall x \in N'(v)$ set its new weight to $w'(x) = w(x) - w(v)$, and let the resulting graph be denoted by $G'$. Then $\alpha_w(G) = w(v) + \alpha_w(G')$ and an MWIS $I$ of $G$ can be constructed from an MWIS $I'$ of $G'$ as follows: if $I' \cap N'(v) = \emptyset$ then $I = I' \cup \{v\}$, otherwise $I = I'$.

Proof. For (i), note that it is safe to remove all $u \in N(v)$ such that $w(u) \leq w(v)$ since these vertices meet the criteria for the neighbor removal reduction. All vertices $x \in N'(v)$ that remain have weight $w(v) > w(v)$ and are not isolated.

Case 1 ($I' \cap N'(v) = \emptyset$): Let $I'$ be an MWIS of $G'$, we show that if $I' \cap N'(v) = \emptyset$ then $I = I' \cup \{v\}$. To show this, we show that $w(v) + \alpha_w(G[V\setminus N[v]]) \geq \alpha_w(G[V\setminus N[v]])$.

Let $x \in N'(v)$. Since $x \notin I'$, we have that

$$w(v) + \alpha_w(G') = w(v) + \alpha_w(G'[V'\setminus N'(v)]) = w(v) + \alpha_w(G[V\setminus N[v]])$$

and

$$w(v) + \alpha_w(G') \geq w(v) + w'(x) + \alpha_w(G'[V'\setminus N[x]]) = w(v) + w(x) - w(v) + \alpha_w(G'[V'\setminus N[x]]) = w(x) + \alpha_w(G[V\setminus N[x]])$$

Thus, for any $x \in N'(v)$, we have that

$$w(v) + \alpha_w(G') = w(v) + \alpha_w(G[V\setminus N[v]]) \geq w(x) + \alpha_w(G[V\setminus N[x]])$$

and therefore the heaviest independent set containing $v$ is at least the weight of the heaviest independent containing any neighbor of $v$. Concluding, we have that

$$w(v) + \alpha_w(G[V\setminus N[v]]) \geq \alpha_w(G[V\setminus N[v]])$$

and therefore $I = I' \cup \{v\}$ is an MWIS of $G$.

Case 2 ($I' \cap N'(v) \neq \emptyset$): Let $I'$ be an MWIS of $G'$, we show that if $I' \cap N'(v) \neq \emptyset$ then $I = I'$. To show this, let $\{x\} = I' \cap N'(v)$. Define $G''$ as the graph resulting from increasing the weight of $N'(v)$ by $w(v)$, i.e. $\forall u \in N'(v)$ we set $w''(u) = w'(u) + w(v) = w(u)$. We first show that $I'' = I$ is an MWIS of $G''$. Therefore, assume that $I^*$ is an MWIS of $G''$ with $w(I^*) > w(I')$ that does not contain $x$. However, then $I^*$ is also a better MWIS on $G'$ which contradicts our initial assumption. Finally, we have that $w(I''') = w(I') + w(v)$, since exactly one node in $N'(v)$ is in $I'$. Next, we define $G'''$ as the graph resulting from adding back $v$ to $G'$ and show that $I''' = I''$ is a MWIS of $G'''$. For this purpose, we assume that $I^*$ is a MWIS of $G'''$ with $w(I^*) > w(I''')$. Then, $v \in I^*$ since we only added this node to $G''$. Likewise, $x \notin I^*$ since its a neighbor of $v$.

Since $w(I^*) > w(I''')$, we have that:

$$w(I^* \setminus \{v\}) = w(I^*) - w(v) > w(I''') - w(v) = w(I') + w(v) - w(v) = w(I').$$

However, since $I^* \setminus \{v\}$ does neither include $v$ nor any neighbor of $v$ it is also an IS of $G'$ that is larger than $I'$. This contradicts our initial assumption and thus $I''' = I'' = I$. Furthermore, since $G''' = G$, we have that $I''' = I = I'$.

\[ \Box \]
### B Graph Properties, Kernel Sizes, Full Tables, Convergence Plots

| Graph                  | $|V|$  | $|E|$  | $K_{\text{dense}}$ | $K_{\text{full}}$ |
|------------------------|------|------|---------------------|---------------------|
| alabama-AM2            | 1164 | 38,772 | 173                 | 173                 |
| alabama-AM3            | 3,604 | 619,928 | 1,614               | 1,614               |
| district-of-columbia-AM1 | 2,500 | 49,302 | 800                 | 800                 |
| district-of-columbia-AM2 | 13,597 | 3,219,590 | 6,360              | 6,360              |
| district-of-columbia-AM3 | 46,221 | 55,458,274 | 33,367             | 33,367             |
| florida-AM2           | 1,254 | 33,872 | 41                  | 41                  |
| florida-AM3           | 2,985 | 308,086 | 1,069               | 1,069               |
| georgia-AM3           | 1,680 | 148,252 | 861                 | 861                 |
| greenland-AM3         | 4,986 | 7,304,722 | 3,942              | 3,942              |
| hawaii-AM2             | 2,875 | 530,316 | 428                 | 428                 |
| hawaii-AM3             | 28,006 | 98,889,842 | 24,436            | 24,436            |
| idaho-AM3             | 4,064 | 7,848,160 | 3,208              | 3,208              |
| kansas-AM3             | 2,732 | 1,613,824 | 1,605              | 1,605              |
| kentucky-AM2          | 2,453 | 1,286,856 | 442                | 442                |
| kentucky-AM3          | 19,095 | 119,067,260 | 16,871           | 16,871           |
| louisiana-AM3         | 1,162 | 74,154 | 382                 | 382                 |
| maryland-AM3          | 1,018 | 190,830 | 187                | 187                |
| massachusetts-AM2      | 1,339 | 70,898 | 196                 | 196                 |
| massachusetts-AM3      | 3,703 | 1,022,982 | 2,008              | 2,008              |
| mexico-AM3            | 1,096 | 94,262 | 620                 | 620                 |
| new-hampshire-AM3      | 1,107 | 36,042 | 247                 | 247                 |
| north-carolina-AM3    | 1,557 | 473,478 | 1,178              | 1,178              |
| oregon-AM2            | 1,325 | 115,034 | 35                  | 35                  |
| oregon-AM3            | 5,588 | 5,825,402 | 3,670              | 3,670              |
| pennsylvania-AM3      | 1,148 | 52,928 | 315                 | 315                 |
| rhode-island-AM2      | 2,866 | 590,976 | 1,103              | 1,103              |
| rhode-island-AM3      | 15,124 | 25,244,438 | 13,031           | 13,031           |
| utah-AM3              | 1,339 | 85,744 | 568                 | 568                 |
| vermont-AM3           | 3,436 | 2,272,328 | 2,630              | 2,630              |
| virginia-AM2          | 2,279 | 120,080 | 237                | 237                |
| virginia-AM3          | 6,185 | 1,331,806 | 3,867             | 3,867             |
| washington-AM2        | 3,025 | 304,898 | 382                | 382                |
| washington-AM3        | 10,022 | 4,692,420 | 8,030             | 8,030             |
| west-virginia-AM3     | 1,185 | 251,240 | 991                | 991                |

| Graph                  | $|V|$  | $|E|$  | $K_{\text{dense}}$ | $K_{\text{full}}$ |
|------------------------|------|------|---------------------|---------------------|
| as-skitter             | 4,964,115 | 22,190,596 | 27,318             | 9,180             |
| ca-AstroPh             | 18,772 | 396,100 | 0                   | 0                  |
| ca-CondMat             | 23,133 | 186,878 | 0                   | 0                  |
| ca-GrQc                | 5,242  | 28,968  | 0                   | 0                  |
| ca-HepPh               | 12,008 | 236,978 | 0                   | 0                  |
| ca-HepTh               | 9,877  | 51,946  | 0                   | 0                  |
| email-Enron            | 36,092 | 367,662 | 0                   | 0                  |
| email-EuAll            | 265,214 | 728,926  | 0                   | 0                  |
| p2p-Gnutella04         | 10,876 | 79,988  | 0                   | 0                  |
| p2p-Gnutella05         | 8,846  | 63,678  | 0                   | 0                  |
| p2p-Gnutella06         | 8,717  | 63,050  | 0                   | 0                  |
| p2p-Gnutella08         | 6,301  | 41,554  | 0                   | 0                  |
| p2p-Gnutella09         | 8,114  | 52,026  | 0                   | 0                  |
| p2p-Gnutella24         | 26,518 | 130,738 | 0                   | 0                  |
| p2p-Gnutella25         | 22,687 | 109,410 | 11                  | 0                  |
| p2p-Gnutella30         | 36,682 | 176,656 | 10                  | 0                  |
| p2p-Gnutella31         | 62,586 | 295,784 | 0                   | 0                  |
| roadNet-CA             | 1,965,206 | 5,533,214 | 233,083           | 63,926            |
| roadNet-PA             | 1,088,092 | 3,083,796 | 115,536           | 38,080            |
| roadNet-TX             | 1,379,917 | 3,843,320 | 151,570           | 39,433            |
| soc-Epinions1          | 75,879 | 811,480 | 6                   | 0                  |
| soc-LiveJournal1       | 4,847,571 | 85,702,474 | 61,690            | 29,779            |
| soc-Slashdot0811       | 77,360 | 938,360 | 0                   | 0                  |
| soc-Slashdot0902       | 82,168 | 1,008,460 | 20                 | 0                  |
| soc-pokec-relationships | 1,632,803 | 44,603,928 | 927,214           | 902,748           |
| web-BerkStan           | 685,230 | 13,298,940 | 37,004            | 17,482            |
| web-Google             | 875,713 | 8,444,102 | 2,892             | 1,178             |
| web-NotreDame          | 325,729 | 2,180,216 | 14,038            | 6,760             |
| web-Stanford           | 281,903 | 3,985,272 | 14,280            | 2,640             |
| wiki-Talk              | 2,394,385 | 9,319,130 | 0                 | 0                 |
| wiki-Vote              | 7,115  | 201,524 | 246                | 237               |

Table 3: Basic properties as well as kernel sizes computed by both variants of our branch-and-reduce algorithm for the OSM networks (top) and SNAP networks (bottom).
Table 4: Best solution found by each algorithm and time (in seconds) required to compute it. The global best solution is highlighted in **bold**. Rows are highlighted in gray if B & R is able to find an exact solution.

| Graph               | DynWVC1 | DynWVC2 | HILS | B & R_{dense} | B & R_{full} |
|---------------------|---------|---------|------|----------------|--------------|
| as-skitter          | 595.70  | 824.09  | 187.5  | 0.76           | 0.67          |
| ca-AstroPh          | 207.97  | 796.46  | 108.3  | 46.17          | 46.67         |
| ca-CondMat          | 71.54   | 114.43  | 222.30 | 45.07          | 45.07          |
| ca-GroC             | 1.75    | 289.48  | 0.82   | 0.60           | 0.60           |
| ca-HepPh            | 26.36   | 579.04  | 17.31  | 11.44          | 11.44          |
| ca-Th                | 9.87    | 560.64  | 12.64  | 94.19          | 94.19          |
| email-Em            | 295.07  | 144.37  | 90.50  | 79.40          | 79.40          |
| emall-Eu            | 180.92  | 253.30  | 156.23 | 137.24         | 137.24         |
| p2p-Gnutella04      | 2.46    | 667.49  | 86.88  | 2.64           | 2.64           |
| p2p-Gnutella05      | 24.23   | 556.59  | 3.54   | 0.56           | 0.56           |
| p2p-Gnutella25      | 532.67  | 547.85  | 1.38   | 1.47           | 1.47           |
| p2p-Gnutella30      | 10.06   | 275.92  | 22.66  | 7.36           | 7.36           |
| p2p-Gnutella31      | 169.03  | 470.50  | 43.15  | 34.33          | 34.33          |
| roadNet-CA          | 1.001.61| 109.028 | 1.001.61| 3.121.90       | 3.121.90       |
| roadNet-TX          | 720.57  | 60.940  | 787.59 | 998.56         | 998.56         |
| soc-Slashdot0811    | 809.97  | 9.5511  | 10.18  | 7.61           | 7.61           |
| soc-Slashdot0902    | 783.10  | 9.55.052| 271.12 | 786.70         | 786.70         |
| soc-poke-relationships | 999.99 | 82.822.272 | 1.001.42 | 2.482.82 | 2.482.82 |
| web-Berkan          | 347.17  | 459.13  | 372.33 | 994.73         | 994.73         |
| web-Geographica     | 758.75  | 563.393 | 68.63  | 598.48         | 598.48         |
| web-NotreDame       | 963.44  | 25.975.765| 875.22 | 98.79         | 98.79          |
| web-Slashdot0811    | 1.001.61| 109.028 | 1.001.61| 3.121.90       | 3.121.90       |
| wiki-Talk           | 961.05  | 235.874 | 991.31 | 996.02         | 996.02         |
| wiki-Vote           | 0.74    | 500.436 | 0.75   | 500.436        | 500.436        |

Table 5: Best solution found by each algorithm and time (in seconds) required to compute it. The global best solution is highlighted in **bold**. Rows are highlighted in gray if B & R is able to find an exact solution.
Table 6: Best solution found by each algorithm and time (in seconds) required to compute it. The global best solution is highlighted in **bold**. Rows are highlighted in gray if B & R is able to find an exact solution.

| Graph          | Red + DynWVC1 | Red + DynWVC2 | Red + HILS | B & R_dense | B & R_full |
|----------------|---------------|---------------|------------|-------------|------------|
|               | $t_{max}$    | $w_{max}$    | $t_{max}$  | $w_{max}$   | $t_{max}$  |
| as-skitter     | 64.52        | 123,995,644  | 85.60      | 123,995,608 | 84.70      |
|               | 613,848     | 123,996,322  | 641.38     | 123,997,824 | 746.93     |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 1.143,480   | 1.143,480   | 0.01       | 1.143,480  | 0.02       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
|               | 0.02         | 792,056     | 0.02       | 796,656    | 0.03       |
Figure 7: Solution quality over time for our sample of eight OSM instances (upper two rows) and eight SNAP instances (lower two rows) as given in Section 6.2.