Concurrent Iterated Local Search for the Maximum Weight Independent Set Problem

Ernestine Großmann
Faculty of Mathematics and Computer Science, Heidelberg University, Germany

Kenneth Langedal[1]
Department of Informatics, University of Bergen, Norway

Christian Schulz
Faculty of Mathematics and Computer Science, Heidelberg University, Germany

Abstract

The Maximum Weight Independent Set problem is a fundamental NP-hard problem in combinatorial optimization with several real-world applications. Given an undirected vertex-weighted graph, the problem is to find a subset of the vertices with the highest possible weight under the constraint that no two vertices in the set can share an edge. This work presents a new iterated local search heuristic called CHILS (Concurrent Hybrid Iterated Local Search). The implementation of CHILS is specifically designed to handle large graphs of varying densities. CHILS outperforms the current state-of-the-art on commonly used benchmark instances, especially on the largest instances. As an added benefit, CHILS can run in parallel to leverage the power of multicore processors. The general technique used in CHILS is a new concurrent metaheuristic called Concurrent Difference-Core Heuristic that can also be applied to other combinatorial problems.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Randomized Local Search, Heuristics, Maximum Weight Independent Set, Algorithm Engineering, Parallel Computing

Digital Object Identifier 10.4230/LIPIcs.SEA.2025.22

Related Version *Full Version*: `https://arxiv.org/abs/2412.14198` [12]

Supplementary Material *Software*: `https://github.com/KennethLangedal/CHILS` [21]
archived at `swh:1:dir:7dbe8b0e824fd80a8ef6eb2747291823f2f4d973`

Funding *Ernestine Großmann*: Supported by DFG grant SCHU 2567/3-1.
Kenneth Langedal: Supported by the Research Council of Norway under contract 303404 and Meltzer Research Fund, project number 104066111.

Acknowledgements The inspiration for this work and collaboration started at the Dagstuhl Seminar 23491 on Scalable Graph Mining and Learning [20]. We also thank Fredrik Manne for his helpful feedback throughout the writing process.

1 Introduction

Consider an undirected vertex-weighted graph $G = (V, E, \omega)$, where V is the set of vertices, E is the set of edges, and $\omega : V \to \mathbb{R}^+$ is a function that maps each vertex to a positive weight. An *independent set* $S \subseteq V$ is a subset of the vertices such that no two members of the independent set share an edge, i.e. for all $u, v \in S$ it holds that $\{u, v\} \notin E$. The Maximum Weight Independent Set (MWIS) problem asks for an independent set S of maximum weight, where the weight of S is defined as the sum $\sum_{v \in S} \omega(v)$ of the vertices. MWIS is a generalization of the classical NP-hard problem Maximum Independent Set (MIS), where all weights equal one.

[1] Corresponding author

23rd International Symposium on Experimental Algorithms (SEA 2025).
Editors: Petra Mutzel and Nicola Prezza; Article No. 22; pp. 22:1–22:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The MWIS problem and other closely related problems have several practical applications ranging from matching molecular structures to wireless networks [4]. Recently, Dong et al. [8] introduced a new collection of instances based on real-life long-haul vehicle routing problems at Amazon. The problem they want to solve is to find a subset of vehicle routes such that no two routes share a driver or a load. Each route has a weight, and the objective is to maximize the sum of the route weights. To state this problem as an MWIS, they build a conflict graph where vertices correspond to routes and the route weights are modeled by vertex weights. Furthermore, they connect two vertices by an edge if the corresponding routes have a conflict, i.e. the routs share a driver or a load. For a more detailed overview of other applications, see the collection by Butenko [4].

It is well known that data reduction rules can speed up algorithms for NP-hard problems. Reduction rules reduce an instance in such a way that an optimal solution for the reduced instance can be lifted to an optimal solution for the original instance. Several reduction rules have been developed for MIS and MWIS. Additionally, reduction rules have also improved many heuristic approaches for MWIS and associated problems. Such reduction rules are often used as a preprocessing step before running an exact algorithm or heuristic on the reduced graph.

Our Results

Our contribution is a new concurrent iterated local search heuristic CHILS for the Maximum Weight Independent Set problem. Our CHILS heuristic works by alternating between running local search on the full graph and the Difference-Core (D-Core) which is a subgraph constructed using multiple heuristic solutions. With CHILS we are able to outperform existing heuristics across a wide variety of real-world instances. On vehicle routing instances, we compare CHILS with two recent heuristics, METAMIS [9] and a Bregman-Sinkhorn algorithm [16], both designed specifically for these instances. In contrast, CHILS is not optimized for vehicle routing instances specifically, and does not make use of the additional metainformation provided for these instances. Despite this, it still finds the best solution on 31/37 instances while being significantly closer to the best known solutions in the cases where CHILS is not best. Running CHILS in parallel significantly improves performance to the point where CHILS computes the best solution on 35/37 instances. These results are with one-hour time limit and the same 16-core CPU used to evaluate all the heuristics. For parallel scalability, we include experiemnts on a 128-core CPU where the parallel version of CHILS reaches speedups of up to 104 compared to the sequential version.

2 Preliminaries

In this work, a graph $G = (V, E, \omega)$ is an undirected vertex-weighted graph with $n = |V|$ and $m = |E|$, where $V = \{0, \dots, n-1\}$ and $\omega : V \to \mathbb{R}^+$. The neighborhood $N(v)$ of a vertex $v \in V$ is defined as $N(v) = \{u \in V \mid \{u, v\} \in E\}$. Additionally, we define $N[v] = N(v) \cup \{v\}$. The same sets are defined for the neighborhood $N(U)$ of a set of vertices $U \subseteq V$, i.e. $N(U) = \cup_{v \in U} N(v) \setminus U$ and $N[U] = N(U) \cup U$. The degree $\deg(v)$ of a vertex is defined as the number of its neighbors $\deg(v) = |N(v)|$. The complement graph is defined as $\overline{G} = (V, \overline{E})$, where $\overline{E} = \{\{u, v\} \mid \{u, v\} \notin E\}$ is the set of edges not present in G. Furthermore, for a graph $G = (V, E)$ we define an induced subgraph $G[H]$ on a subset of vertices $H \subseteq V$ by $G[H] = (H, \{\{u, v\} \in E \mid u, v \in H\})$. A set $S \subseteq V$ is called an *independent set* (IS) if for all vertices $u, v \in S$ there is no edge $\{u, v\} \in E$. For a given IS S, a vertex $v \in V \setminus S$ is called *free* if $S \cup \{v\}$ is still an independent set. An IS is called *maximal* if there are no free vertices.

The number of neighbors of a vertex u that are in the solution, i.e. $|N(u) \cap S|$, is called the *tightness* of u. The MAXIMUM INDEPENDENT SET problem (MIS) is that of finding an IS with maximum cardinality. Similarly, the MAXIMUM WEIGHT INDEPENDENT SET problem (MWIS) is that of finding an IS with maximum weight. The weight of an independent set S is defined as $\omega(S) = \sum_{v \in S} \omega(v)$. The complement of a maximal 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 at least one vertex in the set C. The MINIMUM VERTEX COVER problem, defined as computing a vertex cover with minimum cardinality, is thereby dual to the MIS problem. The decision version of the MINIMUM VERTEX COVER (MVC) problem was one of Karp's original 21 NP-complete problems [18]. Another closely related concept is 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 $v \in V$ is called *simplicial* when its neighborhood $N[v]$ forms a clique.

Our new CONCURRENT DIFFERENCE CORE HEURISTIC is based on the concept of a DIFFERENCE-CORE (D-CORE). It is defined using a set of solutions $S = \{S_1, S_2, \ldots, S_k\}$ for a combinatorial problem on a graph G. The D-CORE is an induced subgraph $G[D]$ with the property that for every vertex $v \in D$ there exist two solutions $S_i, S_j \in S$ such that $v \in S_i$ and $v \notin S_j$.

3 Related Work

We give an overview of previous work on heuristic procedures for the MWIS problem. For a full overview of the related work on MWIS, MWVC, and MAXIMUM WEIGHTED CLIQUE solvers, as well as an extensive collection of known reduction rules for the MWIS and MWVC problems, we refer to the survey by Großmann et al. [13]. For more details on data reduction rules, we direct the reader to the survey by Abu-Khzam et al. [1].

Local search is a widely used heuristic approach for MWIS. It starts from any feasible solution and then tries to improve it by simple insertion, removal, or swap operations. Local search generally offers no theoretical guarantees for the solution quality. However, in practice, it often finds high-quality solutions significantly faster than exact procedures. For unweighted graphs, the iterated local search (ARW) by Andrade et al. [2] is a very successful heuristic. It is based on $(1, 2)$-swaps that remove one vertex from the solution and add two new vertices, thus improving the current solution by one. The ARW heuristic uses special data structures that find such a $(1, 2)$-swap in time $\mathcal{O}(m)$ or prove that none exists. It is able to find near-optimal solutions for small to medium-size instances in milliseconds but struggles on large sparse instances with millions of vertices.

The hybrid iterated local search (HILS) by Nogueira et al. [26] 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 neighbors. These two types of neighborhoods are explored separately using variable neighborhood descent (VND). When it was introduced, HILS found optimal solutions on well-known benchmark instances within milliseconds and outperformed other state-of-the-art local search heuristics.

Two other local search heuristics, DYNWVC1 and DYNWVC2, for the complementary MINIMUM WEIGHT VERTEX COVER (MWVC) problem were presented by Cai et al. [5]. Their heuristics extend the existing FASTWVC heuristic [25] 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. DYNWVC2 provides further improvements on large-scale networks but performs worse on map labeling instances.

Li et al. [24] presented a local search heuristic NuMWVC for the MWVC problem. Their heuristic applies reduction rules during the construction phase of the initial solution. Furthermore, they adapt the configuration checking approach [6] 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 showed that NuMWVC outperformed state-of-the-art approaches on graphs of up to millions of vertices.

A hybrid method called GNN-VC was introduced by Langedal et al. [22] to solve the MWVC problem. For this approach, they combined elements from exact methods with local search, data reductions, and Graph Neural Networks. In the experimental evaluation, GNN-VC achieved clear improvements compared to DynWVC2, HILS, and NuMWVC in both solution quality and running time.

Another reduction-based heuristic HtWIS was presented by Gu et al. [15] for the MWIS problem. HtWIS repeatedly applies reductions exhaustively and then chooses one vertex by a tie-breaking policy to add to the solution. Once this vertex and its neighbors have been removed from the graph, the reduction rules can be applied again. Experimental evaluation showed a significant improvement in running time.

Großmann et al. [14] introduced a heuristic called M^2WIS that combines reductions with an evolutionary approach. Here, the authors made use of exact data reductions and heuristic reductions derived from the population to reduce the graph iteratively. With this technique, M^2WIS was able to achieve near-optimal solutions for a wide set of instances.

A metaheuristic METAMIS was introduced by Dong et al. [9] for the vehicle routing instances introduced by Dong et al. [8]. METAMIS is a local search heuristic that uses a new variant of path-relinking to escape local optima. In the experiments, METAMIS outperforms HILS on a wide range of instances, both in terms of time and solution quality. The vehicle routing instances come equipped with initial warm start solutions and clique information derived from the application. Using this clique information, Haller and Savchynskyy [16] proposed a Bregman-Sinkhorn algorithm (BSA) that addresses a family of clique cover LP relaxations. In addition to solving the relaxed dual problem, BSA utilizes a simple and efficient primal heuristic to obtain feasible integer solutions for the initial non-relaxed problem. In the experiments, BSA outperforms METAMIS on time and solution quality, but only in the cold-start configuration where METAMIS does not use the precomputed solutions.

4 Novel Concurrent Local Search

The proposed heuristic consists of two parts: (1) an iterated local search procedure we refer to as BASELINE, and (2) a new, concurrent heuristic called CHILS (Concurrent Hybrid Iterated Local Search). In the following section, we first give a high-level overview of the proposed approach and then provide a detailed description of each component of our heuristic.

4.1 High-Level Description

A fast heuristic implemented to run efficiently is often better in practice than a more complicated and slow heuristic, especially when the heuristic makes heavy use of random decisions. This can be seen from the results of programming competitions such as PACE (Parameterized Algorithms and Computational Experiments), where fast randomized local search has become the default heuristic strategy among the winning solvers [3, 11, 19]. We also use this approach for our BASELINE local search. First, it makes a small number of

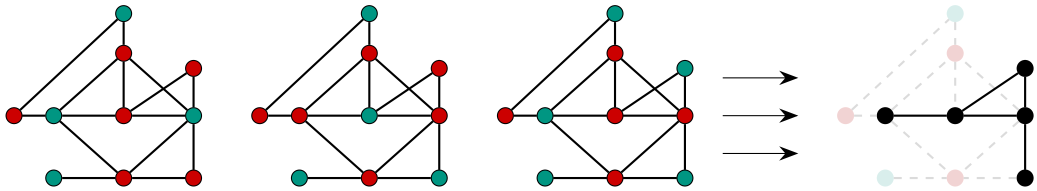


Figure 1 D-Core illustration. Vertices part of all or non of the solutions are not in the D-Core.

random changes in a local area of the current solution. Then, it applies greedy improvements in random order until a local optimum is found. Finally, if the new local optimum is worse than the previous, it backtracks the changes and repeats. Compared to more complicated heuristics, the main benefit of our BASELINE heuristic is that it is inherently local. Regardless of the graph size, one iteration of the search typically only touches vertices a few edges away from where the random alteration started. Backtracking to the best solution is also local, as long as queues are used to track changes. Using queues also differentiates the implementation of our BASELINE from other heuristics, such as HILS [26], that make a copy of the entire solution instead.

The high-level idea of CHILS is a new metaheuristic we call Concurrent Difference-Core Heuristic. Instead of only trying to improve one solution, we maintain several and update each one concurrently. When used sequentially, each solution is updated round-robin. At fixed intervals, the solutions are used to create a Difference-Core (D-Core) instance based on where the solutions differ. More specifically, if a vertex is part of all or none of the solutions, it is not part of the D-Core. A formal definition of the D-Core is presented in the preliminaries in Section 2. The intuition is that the intersection of the solutions is likely to be part of an optimal solution, and where the solutions differ indicates areas where further improvements could be made. Since there is no guarantee that the intersection of the solutions is part of an optimal solution, the Concurrent Difference-Core Heuristic alternates between looking for improvements on the original instance and the D-Core. The D-Core is always constructed according to the current set of solutions. While the general Concurrent Difference-Core Heuristic can work using any heuristic method, our CHILS heuristic uses the BASELINE iterated local search.

4.2 Baseline Local Search

The outline of the BASELINE local search is shown in Algorithm 1. Each search iteration starts by picking a random vertex u from the graph. If the vertex is already in the solution $u \in S$, or the tightness of u is one, i.e. $|N(u) \cap S| = 1$, then an alternating augmenting walk (AAW) is used to perturb the current solution. If u is not currently in the solution, it is added along with a random number of additional changes in its close proximity. The vertices that see a change in their neighborhood are considered “close proximity” to u. These vertices are continuously queued up in a queue Q to find greedy improvements more efficiently later. We also use the size of Q to control the amount of random changes. We stop perturbing the solution once $|Q|$ exceeds m_q, where m_q is a hyperparameter for the BASELINE heuristic. The benefit of using Q and m_q is that it stabilizes the computational cost for one iteration across different graph densities. For instance, the number of changes needed to fill Q in a sparse area is higher than in a dense area.

Algorithm 1 BASELINE Local Search.

Data: Graph $G = (V, E, w)$, independent set S, max queue size m_q, and time limit t
Result: Improved independent set S

```
Q = ∅    /* Always filled with vertices that observe changes to S    */
while time spent < t do
    cost = ω(S)
    u = uniform random from [0, |V| − 1]
    if u ∈ S or |N(u) ∩ S| = 1 then
        S, Q = AAW-moves(G, S, Q, u)
    else
        S = {u} ∪ S \ N(u)
        Q = Q ∪ N[u]
        while |Q| < m_q do
            v = random element from Q
            if v ∈ S then
                S = S \ {v}
            else
                S = {v} ∪ S \ N(v)
            end
            Q = Q ∪ N[v]
        end
    end
    S, Q = GREEDY(G, S, Q)
    if w(S) < cost then
        undo changes to S
    end
end
return S
```

After perturbing the current solution, greedy improvements are made to find a new local optimum. Having stored the candidate vertices in Q speeds up the search for this new local optima in sparse graphs. We incorporate three greedy improvement operators for GREEDY in BASELINE described in the following.

Neighborhood Swap

For a vertex $u \notin S$, if $\omega(u) > \omega(N(u) \cap S)$, then the independent set obtained by inserting the vertex u and removing all neighbors of u that are currently in the solution, i.e. $S = \{u\} \cup (S \setminus N(u))$, leads to an independent set of higher weight. For each vertex $u \in V$, the combined weight of the neighbors that are currently in the solution $\Sigma_{v \in N(u) \cap S}\, w(v)$ is maintained at all times. This additional data allows the neighborhood swap to be checked in $\mathcal{O}(1)$ time.

One-Two Swap

For a vertex in the current solution $u \in S$, consider the set $T = \{v \in N(u) \mid N(v) \cap S = \{u\}\}$ with one-tight vertices in the neighborhood of u. If there exists a pair of vertices $\{x, y\} \subseteq T$ such that $\{x, y\} \notin E$ and where $w(u) < w(x) + w(y)$, then swapping u for x and y leads to a better solution. Examining all one-two swaps can be done in $\mathcal{O}(m)$ time [2].

Alternating Augmenting Walk

We use a slightly modified version of the alternating augmenting path (AAP) introduced by Dong et al. [9]. An alternating augmenting walk (AAW) is a sequence of vertices that alternate between being in and out of the solution S. AAW moves are used both for perturbation and finding greedy improvements. When used for perturbation, the AAW is always extended in a random direction. After no more vertices can be added to the walk, the entire AAW is applied to the solution unless a strictly improving prefix of the walk exists. When searching for greedy improvements, the walk always starts from a one-tight vertex and extends in the best direction possible.

Let U be the set of vertices on the AAW in S, and $\overline{U}$ be the vertices on the AAW not in S. A valid AAW has the property that the set S' obtained by applying the AAW, i.e. $S' = (S \setminus U) \cup \overline{U}$ is also an independent set. This means $\overline{U}$ must also be an independent set where $N(\overline{U}) \cap S = U$. The main purpose of AAWs is to find improving walks where $\omega(\overline{U}) > \omega(U)$, and swap the vertices in U for those in $\overline{U}$ to obtain a heavier independent set. As a secondary use case, they can also perturb the current solution. The benefit of using AAWs instead of random perturbation is that the cardinality of the new solution can only be one less than the original. Constructing an AAW starts with either a single vertex $u \in S$ or a pair of vertices $v \notin S, u \in S$, such that $N(v) \cap S = \{u\}$. The AAW is then extended from the last vertex u on the walk two vertices $x \in N(u), y \in N(x)$ at a time, such that the following three conditions are met.

1. x is not adjacent to any vertices in $\overline{U}$
2. x is not currently in $\overline{U}$
3. x is adjacent to precisely two vertices in the solution $N(x) \cap S = \{u, y\}$

The main difference to the definition of an AAP given for METAMIS [9] is that y is allowed to already be on the path. This results in a walk rather than a path. With the path constraint as in METAMIS, an x, y pair that loops back to an earlier vertex on the path is not a valid extention of the AAP. After applying this AAP (without x, y), x would be a free vertex. In our relaxed definition, we can pick up these free vertices directly. A downside with our definition is that the walks could remain more local than a straight path away from the source vertex.

4.3 CHILS

We give an overview of our concurrent heuristic CHILS in Algorithm 2. First, we run BASELINE on P different solutions for the full graph with a time limit of t_G seconds. Each solution is assigned a different random seed and slightly modified max queue size (m_q) to increase their difference. We also assign each solutions an *id* and always keep track of the best solution found so far. Note that the id of the best solution can change during the execution of the algorithm. After improving the solutions on the full graph, i.e. after $P \times t_G$ seconds, we construct the D-Core instance. This is done by removing vertices that are part of all or none of the P independent sets, see Figure 1 for an example.

On the D-Core, we again start our BASELINE local search P times with a time limit of t_C to generate new solutions. We extend an independent set on the D-Core to the full graph by also including the vertices that were in the intersection of all P solutions. This independent set can replace the previous solution with the same id. However, for solutions with an even id as well as for the best solution found so far, replacements are only made if the new solution is of higher weight. Letting half of the solutions always accept the D-Core solution helps diversify the search.

Algorithm 2 The CHILS Algorithm.

Data: Graph $G = (V, E, \omega)$, number of solutions P, max queue size m_q, time limit for the full graph t_G, time limit for the D-Core t_C, and overall time limit t
Result: Independent set S

1 $C = \{S_1, S_2, ..., S_P\}$ /* using GREEDY$(G, \emptyset, V)$ with different seeds */
2 **while** *time spent* $< t$ **do**
3   **parallel for** $S_i \in C$ **do**
4     $S_i = \text{BASELINE}(G, S_i, m_q + 4i, t_G)$
5   **end**
6   G' = compute D-Core using C
7   **parallel for** $S_i \in C$ **do**
8     $S' = \text{BASELINE}(G', \emptyset, m_q + 4i, t_C)$
9   **if** $\omega(S') + \textit{offset} \geq \omega(S_i)$ *OR (i is odd AND S_i is not best)* **then**
10   apply S' to S_i
11   **end**
12   **end**
13   **if** $|V'| < 500$ **then**
14   C = PARALLEL_PERTURB(C)
15   **end**
16 **end**
17 **return** *$S \in C$ with largest weight*

Using the D-Core helps concentrate the local search on the more difficult regions of the graph, where our P solutions differ. However, since the areas where they agree are not necessarily part of an optimal solution, CHILS alternates between using the BASELINE on the original instance and the recomputed D-Core based on the current P solutions.

If the number of vertices in the D-Core falls below some small constant value, it indicates that the P solutions are all quite similar. Since this reduces the benefit of our approach, we perturb all solutions with an odd id, except for the best solution, and without backtracking in the case where the new local optima is worse. In Algorithm 2, this is shown as PARALLEL_PERTURB on line 14, where perturbing one solution is done as described in lines 10-18 of Algorithm 1. As with accepting replacement solutions from the D-Core, perturbing only half the solutions here helps to diversify the search and escape local optima.

Parallel CHILS

Our CHILS approach is easily parallelizable, with a natural choice for the number of solutions being exactly the number of cores available on the machine, allowing each solution to be improved simultaneously. In this configuration, the worst-case scenario is similar to running the underlying BASELINE local search sequentially, at least when technical details such as memory bandwidth and dynamic clock speeds are ignored. For larger numbers of solutions, the parameter P should be divisible by the number of threads running to ensure that no threads are idle.

5 Experimental Evaluation

The following section introduces the experimental setup and establishes the dataset used for evaluating the proposed approaches. Then, we present state-of-the-art comparison for BASELINE and CHILS. Finally, we present results for the parallel scalability of CHILS.

5.1 Methodology

All the experiments were run on a machine with an Intel Xeon w5-3435X 16-core processor and 132 GB of memory, running Ubuntu 22.04.4 with Linux kernel 5.15.0-113. Both CHILS and BASELINE were implemented in C and compiled with GCC version 11.4.0 using the -O3 flag. OpenMP was used for the parallel implementations. The source code is openly available on GitHub[2].

We evaluate eight instances in parallel for the sequential experiments. To ensure fairness between the algorithms, the instances start in the same order for each program, and only one program is evaluated at a time. We evaluate each heuristic once for each instance.

We compare our heuritsics BASELINE and CHILS to the state-of-the-art metaheuristic METAMIS by Dong et al. [9], and the Bregman-Sinkhorn algorithm BSA by Haller and Savchynskyy [16]. The source code for METAMIS is not publicly available, and therefore, we had to use the numbers reported in [9]. METAMIS was evaluated using the Amazon Web Service r3.4xlarge compute node running Intel Xeon Ivy Bridge Processors and was implemented in Java. The authors also ran their heuristic five times with different seeds and reported the best solution found. For the Bregman-Sinkhorn algorithm BSA, we use the variation that produces integer solutions only after reaching 0.1 % relative duality gap for the LP relaxation by recommendation from the authors [17][3].

For the comparison of different algorithms, we use performance profiles [7]. At a high level, performance profiles show the relationship between the solution size or running time of each algorithm and the corresponding result produced by the best or fastest solution given by any of the algorithms. The performance profile gives each algorithm a non-decreasing, piecewise constant function. In general, the y-axis represents the fraction of instances where the objective function is better than or equal to τ times the best objective function value. For our solution quality comparison the y-axis shows the fraction of instances $\#\{\text{weight} \geq \tau * \text{best}\}/\#G$. Here, *weight* refers to the independent set weight computed by an algorithm on an instance, and *best* corresponds to the best result among all the algorithms shown in the plot. $\#G$ is the number of graphs in the dataset. The parameter τ is plotted on the x-axis. For maximization problems we have $0 < \tau \leq 1$. In general, algorithms are considered to perform well if a high fraction of instances are solved within a factor of τ as close to 1 as possible, indicating that many instances are solved close to or better/faster than the optimum/fastest solution found by all competing algorithms.

5.2 Datasets

Our set of instances consists of 37 vehicle routing instances introduced by Dong et al. [8], see Table 3 in the Appendix. Initial warm-start solutions derived from the application and clique information for the graphs are also provided for these instances. The clique information is a clique cover of the graph, i.e. a collection of potentially overlapping cliques that cover the entire graph. METAMIS [9] and BSA [16] use this clique information.

[2] `https://github.com/KennethLangedal/CHILS`

[3] The exact command the authors proposed and we used was `mwis_json -l temp_cont -B 50 --initial-temperature 0.01 -g 50 -b 100000000 -t [seconds] [instance]`

Table 1 Geometric mean weight computed by the sequential CHILS after one hour for different configurations for the number of solutions P and time limits for the local search $t = t_G = t_C$. Note that the time t is spent per solution, and not divided between them.

	$P = 8$	$P = 16$	$P = 32$	$P = 64$
$t = 0.1$	14 119 824.66	14 119 386.43	14 119 515.58	14 100 937.28
$t = 1$	14 124 084.06	14 126 354.36	14 127 387.83	14 117 718.85
$t = 2.5$	14 127 282.12	14 128 073.98	14 130 823.31	14 118 203.18
$t = 5$	14 127 912.55	14 129 544.84	14 129 347.25	14 117 284.05
$t = 10$	14 127 684.40	**14 132 119.66**	14 124 680.31	14 116 758.39
$t = 20$	14 130 214.46	14 125 181.03	14 119 467.07	14 110 491.07
$t = 30$	14 126 895.82	14 118 546.91	14 115 471.69	14 112 363.89

The vehicle routing instances are significantly harder than previously used datasets. The authors of METAMIS report results for HILS that are significantly worse than the new METAMIS heuristic on these instances [9]. The authors of BSA also make a similar observation [16]. Therefore, this experiment section only focuses on the two heurstics METAMIS and BSA, and only on these new vechicle routing instances. For an extended experiment section including comparisons with older heuristics and a wider variety of test instances, see the extended version [12].

5.3 Parameter Tuning

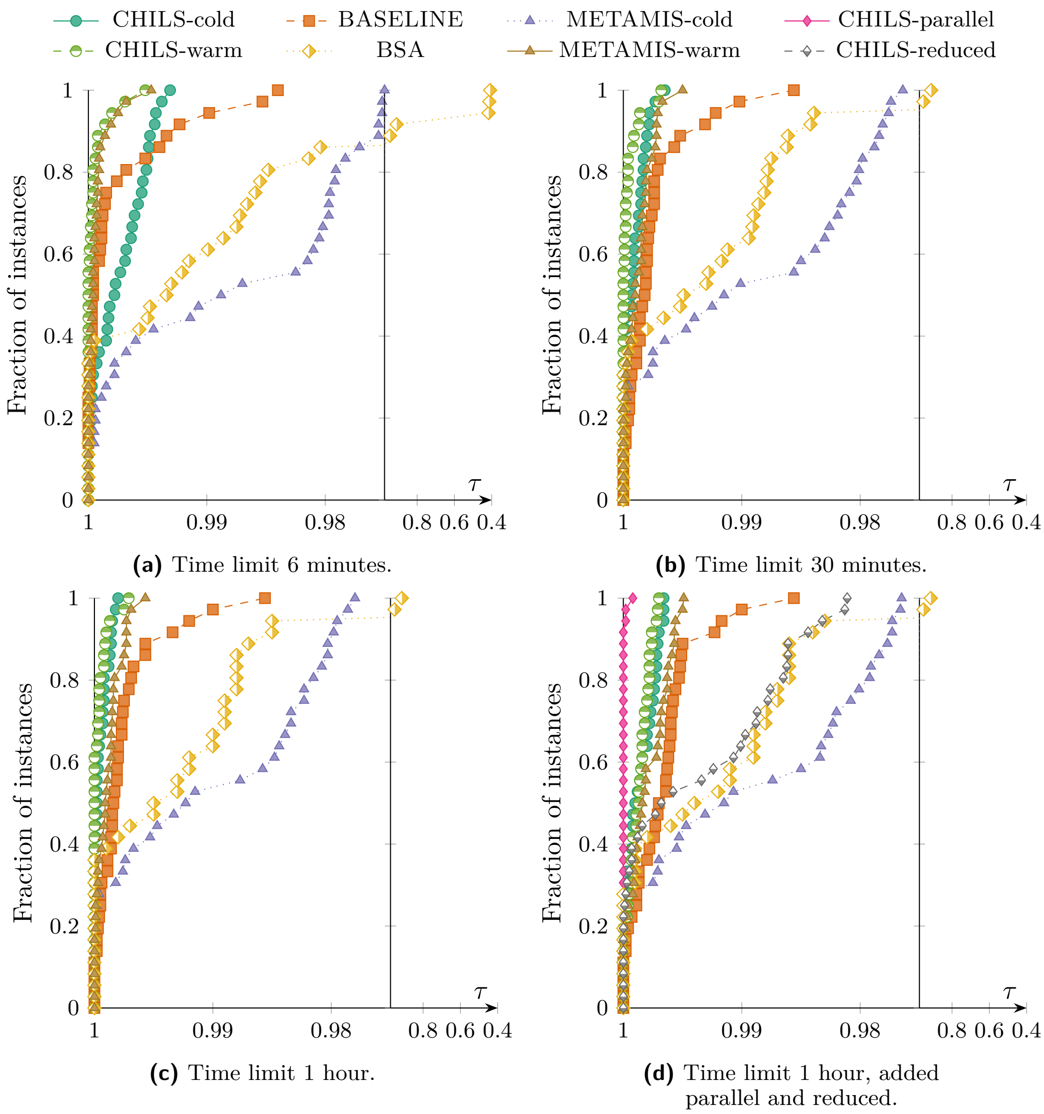
In this section, we perform experiments to find good choices for the number of concurrent solutions P and the time t spent improving them in the sequential version of CHILS. We choose a subset of instances for parameter tuning. For the vehicle routing instances, we used three graphs with more than 500 000 vertices and three with less than 500 000 vertices. In addition to the vehicle routing instances, we also include six other instances that stood out as particularly challenging in previous works [10, 14]. These include 3d meshes derived from simulations using the finite element method (FE) [28], OpenStreetMap (OSM) instances [27], and graphs from the Stanford Large Network Dataset Repository (SNAP) [23]. The instances chosen for these experiments are marked with a $\star$ in Table 3.

We begin our experiments with the parameter defining the number of concurrent solutions $P \in \{8, 16, 32, 48, 64\}$. The second parameter, highly correlating with P, is the time spent per local search run. For this experiment we set $t = t_G = t_C$ and test all $t \in \{0.1, 1, 2.5, 5, 10, 20, 30\}$ seconds. Recall that t_G is the time spent on the original graph and t_C is the time spent on the D-Core. We present the geometric mean solution weight after one hour for the different configurations of these two parameters in Table 1.

In general, we can see that the best choice for t gets lower as the number of solutions P increase. Comparing the configurations with the most solutions, i.e. $P = 64$, the solution quality is worse than using fewer concurrent local search runs for all t values tested. The best configuration we found is $P = 16$ and $t = 10$ seconds, resulting in a geometric mean weight of 14 132 199.66. Note that these choices are only optimizing running CHILS sequentially with one hour time limit.

Observation 1 The best parameter configuration we found experimentally was $P = 16$ and $t = 10$ for running CHILS sequentially with a time limit of one hour per instance.

5.4 State-of-the-Art Comparison



(a) Time limit 6 minutes.

(b) Time limit 30 minutes.

(c) Time limit 1 hour.

(d) Time limit 1 hour, added parallel and reduced.

Figure 2 Performance profiles for state-of-the-art comparison on solution quality on the vehicle routing instances with different time limits. In (d), results computed with parallel CHILS and reduction rules are added.

For the state-of-the-art comparison in this section we compare our heuristics with the results of METAMIS[4] [9] and BSA [16]. The other state-of-the-art heuristics are not designed or implemented for the vechicle routing instances and perform significantly worse. Furthermore, the vertex weights for these instances are very large, which leads to problems for other heuristics that only accept weights that can be represented with 32 bits. In the extended version of this paper we provide additional experiments comparing BASELINE and CHILS against these other heuristics evaluated on a larger set of commonly used test instances [12].

[4] The code is not publicly available, therefore, we could not rerun the experiments.

Due to the size of the weights, we only see small percentage improvements despite significant differences between the solutions. However, most of the solutions found by the sequential heuristics are still not optimal, evident by the significant uplift in solution quality we get using our parallel CHILS. It is also worth mentioning that even small improvements in solution quality, especially for the vehicle routing application, can result in substantial cost reductions [9].

We include a warm-start configuration for CHILS where we start with the provided initial solutions. The METAMIS numbers for cold and warm-starts are taken from [9]. In Figure 2, we present performance profiles to compare the solution quality achieved by the algorithms with different time limits. The first two show the state-of-the-art comparison of solution quality achieved with a 6 and 30 minutes time limit respectively. As expected, the configurations with the warm start perform best with the shortest time limit, see Figure 2a. However, if no initial solution is known, our two variants, BASELINE and CHILS, significantly outperform all competitor configurations. Note that BASELINE is performing better than CHILS on around 80 % of the instances with the 6 minute time limit, since the configuration for CHILS is optimized for running for one hour, see Section 5.3. Compared to the second profile on the right, we can see that with more time, the CHILS solutions improve most, even surpassing the METAMIS-warm results on most instances. Within this time limit, CHILS-warm finds the best solutions on almost 60 % of the instances. Furthermore, both CHILS configurations compute solutions that are at most 0.3 % worse than the best-found solution. On the other hand, on more than 38 % of the instances, BSA and METAMIS-cold find solutions which are more than 1 % worse than the best solutions found on the respective instances.

In Figure 2c, we present the state-of-the-art comparison where all algorithms have one hour time limit and run on the original instances. Here, we can see that CHILS – with and without the initial solutions – generally performs best. The results shown here are very similar to the results computed within the shorter limit of 30 minutes; see Figure 2b.

For the profile in Figure 2d, however, this changes. Here, the time limit is also one hour, but we added a configuration called CHILS-parallel, which utilizes the warm start solution and runs in parallel with the configuration of $P = 64$ and $t = 5$ seconds, using 16 parallel threads. Additionally, we include a configuration that uses reduction rules and run CHILS on the reduced instances. Note that Dong et al. already showed that the vehicle routing instances are not easy to reduce [9]. In the experiments here, we use a new reduction approach with an extended set of reductions introduced in [12]. However, our configuration with reductions, CHILS-reduced, cannot compete with the other configurations on most instances. This means that spending time to reduce these instances is not worthwhile compared to running local search. Note that we do not evaluate BSA on the reduced instances since the clique information after the reduction process is lost. Testing even the fast reduction rules on these graphs takes considerable time. However, on *MR-W-FN* or *MW-D-01*, for example, using reduction rules does help in finding better solutions, see Table 2. Considering all of our variants, we outperform all other schemes in all but two instances, as shown in Figure 2 and Table 2. Furthermore, compared to METAMIS, our approach does not depend as much on having good initial solutions. Nevertheless, using a warm start solution can improve the performance of CHILS further.

Observation 2 On the vehicle routing instances without initial solutions, even our BASELINE approach outperforms the state-of-the-art heuritsics. The BASELINE approach is especially good with a low time limit. Overall, CHILS outperforms *all* the other heuristics regarding solution quality for all time limits tested.

Table 2 Results for vehicle routing instances by Dong et al. [8]. The results for METAMIS were taken from the paper by Dong et al. [9] since the code is not publicly available. Note that the authors of METAMIS used the best out of four runs, while the results for BASELINE, CHILS, and BSA were only run once. All the results show the best solution found after one hour.

	Cold Start				Warm Start		Reduced	Parallel
Instance	METAMIS	BSA	BASELINE	CHILS	METAMIS	CHILS	CHILS	CHILS
CR-S-L-1	5 588 489	5 639 292	5 694 508	5 698 608	5 692 891	5 701 015	5 610 201	**5 718 230**
CR-S-L-2	5 691 892	5 729 194	5 785 140	5 798 470	5 784 034	5 799 458	5 715 463	**5 813 362**
CR-S-L-4	5 681 336	5 725 525	5 781 519	5 791 447	5 777 081	5 792 758	5 698 425	**5 806 975**
CR-S-L-6	3 859 513	3 900 370	3 936 944	3 938 946	3 936 137	3 946 157	3 890 733	**3 952 205**
CR-S-L-7	1 989 879	2 006 810	2 017 624	2 021 238	2 019 428	2 022 339	2 006 857	**2 025 681**
CR-T-C-1	4 654 419	4 717 754	4 742 508	4 744 119	4 743 040	4 750 570	4 702 548	**4 760 428**
CR-T-C-2	4 874 346	4 934 488	4 969 512	4 975 069	4 968 952	4 976 613	4 918 257	**4 987 562**
CR-T-D-4	4 817 281	4 875 268	4 912 984	4 919 079	4 911 646	4 922 752	4 864 858	**4 932 826**
CR-T-D-6	2 970 011	3 009 020	3 024 044	3 027 211	3 024 523	3 029 782	2 999 231	**3 033 189**
CR-T-D-7	1 440 281	1 453 990	1 460 328	1 461 099	1 460 240	1 460 584	1 451 070	**1 462 239**
CW-S-L-1	1 634 950	1 645 459	1 660 063	1 660 472	1 660 815	1 662 580	1 646 548	**1 663 763**
CW-S-L-2	1 708 820	1 713 535	1 737 052	1 738 307	1 738 128	1 739 670	1 723 914	**1 743 532**
CW-S-L-4	1 725 591	1 730 641	1 751 204	1 754 409	1 753 803	1 756 602	1 735 633	**1 759 452**
CW-S-L-6	1 158 925	1 162 552	1 175 335	1 177 272	1 177 156	1 176 354	1 166 774	**1 178 467**
CW-S-L-7	587 288	588 279	594 312	594 598	593 891	593 807	590 834	**594 757**
CW-T-C-1	1 317 775	1 325 862	1 336 232	1 338 085	1 338 064	1 340 085	1 323 170	**1 341 888**
CW-T-C-2	931 802	935 238	944 902	946 217	945 886	945 913	935 907	**947 644**
CW-T-D-4	457 185	459 575	460 955	460 734	461 056	461 108	459 543	**461 475**
CW-T-D-6	457 790	459 238	461 088	461 354	461 312	461 101	460 227	**461 709**
MR-D-03	1 754 110 286	1 757 141 345	1 752 894 190	1 758 762 733	1 757 227 519	1 758 429 721	1 758 775 738	**1 759 255 435**
MR-D-05	1 786 342 921	1 788 812 740	1 781 868 583	1 789 601 100	1 787 849 777	1 789 537 256	1 789 944 187	**1 790 776 639**
MR-D-FN	1 797 573 192	1 801 983 754	1 794 240 970	1 801 499 264	1 799 452 160	1 800 375 890	1 801 727 328	**1 802 925 854**
MR-W-FN	5 358 386 615	5 386 842 780	5 385 799 979	5 385 799 979	**5 386 842 781**	**5 386 842 781**	**5 386 842 781**	**5 386 842 781**
MT-D-01	**238 166 485**	**238 166 485**	**238 166 485**	**238 166 485**	**238 166 485**	**238 166 485**	**238 166 485**	**238 166 485**
MT-D-200	287 048 909	**287 155 108**	287 042 596	287 086 442	287 048 081	287 042 596	287 086 442	287 086 442
MT-D-FN	**290 866 943**	**290 866 943**	**290 866 943**	**290 866 943**	290 771 450	290 771 450	**290 866 943**	**290 866 943**
MT-W-01	**312 121 568**	**312 121 568**	**312 121 568**	**312 121 568**	**312 121 568**	**312 121 568**	**312 121 568**	**312 121 568**
MT-W-200	383 961 323	384 056 011	383 974 084	384 052 157	383 985 408	384 052 017	384 052 157	**384 056 012**
MT-W-FN	390 854 593	390 869 890	**390 869 891**	**390 869 891**	**390 869 891**	**390 869 891**	**390 869 891**	**390 869 891**
MW-D-01	476 334 711	476 298 607	475 180 516	476 328 138	475 987 082	476 120 423	**476 440 656**	476 360 408
MW-D-20	525 124 575	526 857 183	523 423 307	526 883 302	525 486 034	526 333 489	526 648 329	**527 498 481**
MW-D-40	536 520 199	**539 036 121**	533 671 730	538 302 190	536 735 155	537 485 389	538 409 586	538 596 069
MW-D-FN	541 918 916	545 554 192	541 193 604	544 451 893	543 857 187	544 205 239	544 246 489	**545 711 017**
MW-W-01	**1 270 305 952**	**1 270 305 952**	**1 270 305 952**	1 270 235 200	1 269 344 846	1 269 344 846	**1 270 305 952**	**1 270 305 952**
MW-W-05	**1 328 958 047**	1 326 236 043	1 327 478 708	1 328 043 785	**1 328 958 047**	**1 328 958 047**	1 328 043 785	**1 328 958 047**
MW-W-10	1 342 899 725	1 280 286 209	1 340 268 013	1 342 808 634	**1 342 915 691**	**1 342 915 691**	1 342 809 954	**1 342 915 691**
MW-W-FN	**1 350 818 543**	1 235 306 258	1 331 333 002	**1 350 818 543**	**1 350 818 543**	**1 350 818 543**	**1 350 818 543**	**1 350 818 543**

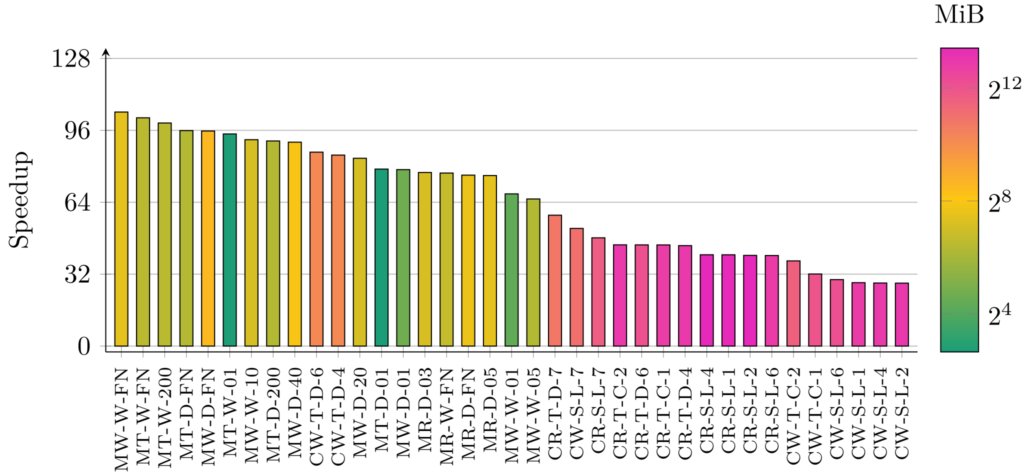


Figure 3 Bar chart showing the speedup on each instance using 128 threads compared to the sequential CHILS. The colors indicate the amount of allocated memory for each instance. The total amount of L3 cache for this machine is 256 MiB, and smaller instances fit entirely in the cache.

5.5 Parallel Scalability

For the parallel results shown in Figure 2d, we used the same machine as the other heuristics to ensure fairness between each program with regard to solution quality. To gain more detailed insight into the parallel scalability of our approach, we utilize a larger machine with an AMD EPYC 9754 128-core processor for our scalability experiments. Furthermore, CHILS as defined in Section 4 relies on wall time to alternate between local search on the full graph and the D-CORE. This introduces variance between runs and makes it difficult to conduct scalability experiments. Therefore, we change the implementation for this section to perform a fixed number of local search iterations instead, where one iteration refers to the while loop starting on Line 2 in Algorithm 1. We also set a fixed number of CHILS iterations, referring to the while loop starting on Line 2 in Algorithm 2. By removing the wall-clock measures and using the same random seed, we ensure that the parallel and sequential versions perform the exact same computations and reach the same solution in the end.

The configuration we use for these experiments consists of 1 000 local search iterations, 10 CHILS iterations, and $P = 128$. Depending on the instance, this ratio of local search on the full graph and D-CORE is reasonably close to the wall-clock version with $t_G = t_C = 10$. Each run was repeated five times, and the best measurement is used. We use the best measurement because, in this configuration, there is no randomness between runs. Any difference we observe in execution time is solely due to factors outside our control, such as fluctuations in clock speed and other programs running on the machine. As such, the best measure is the closest to the true execution time.

The speedup numbers for each instance are shown in Figure 3. The best scaling instance reaches a speedup of 104, while the worst is still 28 times faster than the sequential program. Figure 3 also shows the amount of memory allocated for each instance. This includes the graph, which we store using the compressed sparse row format, and additional data structures required by the heuristic. All memory allocations are done up front before starting the heuristic, and the appropriate thread initializes any thread-local data. The CPU we use has a combined L3 cache of 256 MiB. There is a clear drop in speedup around the point when the data no longer fits in the cache. This indicates that the memory bandwidth is the main bottleneck for larger instances. In terms of load balancing, there could be variations in how much work it takes to perform 1 000 local search iterations for each solution. This is because each solution has a different random seed and m_q parameter. Note that this is not an issue for the version presented in Section 4, since that version uses wall time instead of local search iterations. Table 3 in the Appendix gives detailed execution time, speedup, and the amount of memory used for each instance.

6 Conclusion and Future Work

We introduce a new heuristic called CHILS (Concurrent Hybrid Iterated Local Search) that expands on the HILS heuristic. This new heuristic outperforms all known heuristics across a wide range of test instances in a sequential environment. As an added benefit, CHILS can also leverage the power offered by multicore processors. For the state-of-the-art comparison, letting CHILS use all 16 cores available on the machine significantly improves the solution quality on the vehicle routing instances. Using another 128-core machine for scalability experiments, we show that CHILS reaches speedups up to 104 for the same instances.

These vehicle routing instances by Dong et al. [8] offer a significant challenge for practical MWIS algorithms. Our result marks the third iteration of improvements to this dataset, after METAMIS [9] and BSA [16], and yet, we believe we are still far away from optimal solutions on these instances. This is indicated by the significant uplift in solution quality achieved by running CHILS in parallel. It is especially interesting to improve these results further, as even small improvements in solution quality can result in substantial cost reductions [9].

There are several directions for future research, including finding efficient data reductions that work on these hard instances or using the clique information in the CHILS heuristic. If the use of clique information leads to improvements, then a natural continuation would be to compute clique covers for other hard instances.

CHILS is based on the proposed metaheuristic Concurrent Difference-Core Heuristic. This metaheuristic could lead to improvements for solving other problems as well. As a metaheuristic, it only requires that solutions can be compared to find the Difference-Core; otherwise, any heuristic method can be used internally. Examples of possible problems include Vertex Cover, Dominating set, Graph Coloring, and Connectivity Augmentation. As an added benefit, the Concurrent Difference-Core Heuristic is trivially parallelizable, which could enable improvements in the parallel setting too.

References

1 Faisal N. Abu-Khzam, Sebastian Lamm, Matthias Mnich, Alexander Noe, Christian Schulz, and Darren Strash. Recent advances in practical data reduction. *Algorithms for Big Data*, pages 97–133, 2022.

2 Diogo V. Andrade, Mauricio G. C. Resende, and Renato F. Werneck. Fast local search for the maximum independent set problem. *Journal of Heuristics*, 18:525–547, 2012. `doi:10.1007/S10732-012-9196-4`.

3 Max Bannach and Sebastian Berndt. PACE solver description: The PACE 2023 parameterized algorithms and computational experiments challenge: Twinwidth. In Proceedings of the *18th International Symposium on Parameterized and Exact Computation (IPEC)*, pages 35:1–35:14. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023. `doi:10.4230/LIPICS.IPEC.2023.35`.

4 Sergiy Butenko. *Maximum independent set and related problems, with applications*. University of Florida, 2003.

5 Shaowei Cai, Wenying Hou, Jinkun Lin, and Yuanjie Li. Improving local search for minimum weight vertex cover by dynamic strategies. In Proceedings of the *27th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1412–1418, 2018. doi:10.24963/IJCAI.2018/196.

6 Shaowei Cai, Kaile Su, and Abdul 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.

7 Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002. doi:10.1007/S101070100263.

8 Yuanyuan Dong, Andrew V. Goldberg, Alexander Noe, Nikos Parotsidis, Mauricio G. C. Resende, and Quico Spaen. New instances for maximum weight independent set from a vehicle routing application. In Proceedings of the *Operations Research Forum*, volume 2, pages 1–6. Springer, 2021. doi:10.1007/S43069-021-00084-X.

9 Yuanyuan Dong, Andrew V. Goldberg, Alexander Noe, Nikos Parotsidis, Mauricio G. C. Resende, and Quico Spaen. A metaheuristic algorithm for large maximum weight independent set problems. *Networks*, 81:91–112, 2024. doi:10.1002/NET.22247.

10 Alexander Gellner, Sebastian Lamm, Christian Schulz, Darren Strash, and Bogdán Zaválnij. Boosting data reduction for the maximum weight independent set problem using increasing transformations. In Proceedings of the *Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 128–142. SIAM, 2021. doi:10.1137/1.9781611976472.10.

11 Ernestine Großmann, Tobias Heuer, Christian Schulz, and Darren Strash. The PACE 2022 parameterized algorithms and computational experiments challenge: directed feedback vertex set. In Proceedings of the *17th International Symposium on Parameterized and Exact Computation (IPEC)*, pages 26:1–26:18. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2022. doi:10.4230/LIPICS.IPEC.2022.26.

12 Ernestine Großmann, Kenneth Langedal, and Christian Schulz. Accelerating reductions using graph neural networks and a new concurrent local search for the maximum weight independent set problem. *arXiv preprint arXiv:2412.14198*, 2024. doi:10.48550/arXiv.2412.14198.

13 Ernestine Großmann, Kenneth Langedal, and Christian Schulz. A comprehensive survey of data reduction rules for the maximum weighted independent set problem. *arXiv preprint arXiv:2412.09303*, 2024. doi:10.48550/arXiv.2412.09303.

14 Ernestine Großmann, Sebastian Lamm, Christian Schulz, and Darren Strash. Finding near-optimal weight independent sets at scale. *Journal of Graph Algorithms and Applications*, 28(1):439–473, 2024. doi:10.7155/JGAA.V28I1.2997.

15 Jiewei Gu, Weiguo Zheng, Yuzheng Cai, and Peng Peng. Towards computing a near-maximum weighted independent set on massive graphs. In Proceedings of the *27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 467–477, 2021. doi:10.1145/3447548.3467232.

16 Stefan Haller and Bogdan Savchynskyy. A Bregman-Sinkhorn algorithm for the maximum weight independent set problem. *arXiv preprint arXiv:2408.02086*, 2024.

17 Stefan Haller and Bogdan Savchynskyy. Personal communication, 2024.

18 Richard M. Karp. *Reducibility among combinatorial problems*. Springer, 2010. doi:10.1007/978-3-540-68279-0_8.

19 Leon Kellerhals, Tomohiro Koana, André Nichterlein, and Philipp Zschoche. The PACE 2021 parameterized algorithms and computational experiments challenge: Cluster editing. In Proceedings of the *16th International Symposium on Parameterized and Exact Computation (IPEC)*, pages 26:1–26:18. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2021. doi:10.4230/LIPICS.IPEC.2021.26.

20 Danai Koutra, Henning Meyerhenke, Ilya Safro, and Fabian Brandt-Tumescheit. Scalable graph mining and learning (dagstuhl seminar 23491). *Dagstuhl Reports*, 13(12):1–23, 2024. doi:10.4230/DAGREP.13.12.1.

21 Kenneth Langedal, Ernestine Großmann, and Christian Schulz. KennethLangedal/CHILS: v1.0 - SEA2025, April 2025. doi:10.5281/zenodo.15173364.

22 Kenneth Langedal, Johannes Langguth, Fredrik Manne, and Daniel Thilo Schroeder. Efficient minimum weight vertex cover heuristics using graph neural networks. In Proceedings of the *20th International Symposium on Experimental Algorithms (SEA)*, pages 12:1–12:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.SEA.2022.12.

23 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. URL http://snap.stanford.edu/data, June 2014.

24 Ruizhi Li, Shuli Hu, Shaowei Cai, Jian Gao, Yiyuan Wang, and Minghao Yin. NuMWVC: A novel local search for minimum weighted vertex cover problem. *Journal of the Operational Research Society*, 71(9):1498–1509, 2020. doi:10.1080/01605682.2019.1621218.

25 Yuanjie Li, Shaowei Cai, and Wenying Hou. An efficient local search algorithm for minimum weighted vertex cover on massive graphs. In Proceedings of the *11th International Conference on Simulated Evolution and Learning (SEAL)*, pages 145–157. Springer, 2017. doi:10.1007/978-3-319-68759-9_13.

26 Bruno Nogueira, Rian G. S. Pinheiro, and Anand 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.

27 OpenStreetMap. URL https://www.openstreetmap.org.

28 Chris Walshaw. Graph partitioning archive. URL https://chriswalshaw.co.uk/partition/, 2000.

A Instances

Table 3 Detailed graph properties and parallel results for the set of vehicle routing instances and the six additional instances used for parameter tuning, where n is the number of vertices and m is the number of edges. Furthermore, the size is given in MiB and the sequential and parallel times are in seconds. For these results, we used $P = 128$, 1 000 local search iterations, 10 CHILS iterations, $m_q = 32$, and the larger AMD EPYC 9754 128-core processor. The instances marked with a $\star$ were used for parameter tuning.

Instance	n	m	Size [MiB]	Seq. [s]	Par. [s]	Speedup
⋆CR-S-L-1	863 368	331 203 970	10 570.36	1 021.35	25.14	40.62
CR-S-L-2	880 974	342 158 741	10 850.02	1 043.11	25.85	40.35
CR-S-L-4	881 910	344 057 350	10 884.97	1 040.26	25.60	40.63
CR-S-L-6	578 244	219 717 582	7 047.38	853.77	21.19	40.29
CR-S-L-7	270 067	94 109 215	3 161.62	653.13	13.56	48.18
CR-T-C-1	602 472	194 753 152	6 821.26	740.48	16.45	45.02
CR-T-C-2	652 497	215 694 927	7 460.45	764.48	16.97	45.05
CR-T-D-4	651 861	220 480 534	7 529.41	771.02	17.25	44.70
CR-T-D-6	381 380	115 082 762	4 192.90	522.43	11.60	45.03
CR-T-D-7	163 809	43 028 583	1 703.24	337.21	5.79	58.24
CW-S-L-1	411 950	283 860 106	6 963.56	1 281.68	45.43	28.21
CW-S-L-2	443 404	315 569 883	7 648.40	1 418.39	50.64	28.01
⋆CW-S-L-4	430 379	303 042 962	7 374.03	1 350.81	48.13	28.06
CW-S-L-6	267 698	171 132 761	4 321.77	941.01	31.82	29.58
CW-S-L-7	127 871	78 459 291	2 014.24	804.69	15.37	52.35
CW-T-C-1	266 403	144 634 578	3 909.16	853.00	26.59	32.08
⋆CW-T-C-2	194 413	111 098 006	2 937.45	810.82	21.39	37.90
CW-T-D-4	83 091	37 881 529	1 108.95	645.07	7.59	84.98
CW-T-D-6	83 758	38 781 839	1 126.95	640.02	7.42	86.29
MR-D-03	21 499	130 508	139.36	58.50	0.76	77.24
MR-D-05	27 621	236 044	180.09	62.99	0.83	75.88
⋆MR-D-FN	30 467	296 369	199.19	65.91	0.87	76.06
MR-W-FN	15 639	126 800	101.86	29.40	0.38	77.00
MT-D-01	979	3 125	6.30	104.31	1.32	78.76
MT-D-200	10 880	505 359	77.23	351.85	3.86	91.27
MT-D-FN	10 880	604 041	78.74	389.46	4.06	95.89
MT-W-01	1 006	2 411	6.46	43.16	0.46	94.39
MT-W-200	12 320	515 871	86.59	450.71	4.54	99.27
MT-W-FN	12 320	553 895	87.17	462.98	4.56	101.59
MW-D-01	3 988	13 556	25.69	107.49	1.37	78.52
MW-D-20	20 054	606 318	137.39	73.21	0.88	83.57
⋆MW-D-40	33 563	1 879 303	243.13	110.25	1.22	90.74
MW-D-FN	47 504	4 017 196	364.83	148.78	1.55	95.70
MW-W-01	3 079	22 664	20.02	40.77	0.60	67.73
⋆MW-W-05	10 790	485 261	76.35	68.27	1.04	65.46
MW-W-10	18 023	1 451 813	137.31	144.98	1.58	91.84
MW-W-FN	22 316	2 275 623	177.31	146.35	1.41	104.15
⋆fe-pwt	36 519	144 794	–	–	–	–
⋆fe-rotor	99 617	662 431	–	–	–	–
⋆osm-hawaii-3	28 006	49 444 921	–	–	–	–
⋆osm-vermont-3	3 436	1 136 164	–	–	–	–
⋆snap-as-skitter	1 696 415	11 095 298	–	–	–	–
⋆snap-soc-LiveJ.	4 847 571	42 851 237	–	–	–	–