SubModST: A Fast Generic Solver for Submodular Maximization with Size Constraints

Henning Martin Woydt
Heidelberg University, Germany

Christian Komusiewicz
Friedrich Schiller University Jena, Institute of Computer Science, Germany

Frank Sommer
Friedrich Schiller University Jena, Institute of Computer Science, Germany

Abstract

In the Cardinality-Constrained Maximization (Minimization) problem the input is a universe $\mathcal{U}$, a function $f : 2^{\mathcal{U}} \to \mathbb{R}$, and an integer k, and the task is to find a set $S \subseteq \mathcal{U}$ with $|S| \leq k$ that maximizes (minimizes) $f(S)$. Many well-studied problems such as Facility Location, Partial Dominating Set, Group Closeness Centrality and Euclidean k-Medoid Clustering are special cases of Cardinality-Constrained Maximization (Minimization). All the above-mentioned problems have the diminishing return property, that is, the improvement of adding an element $e \in \mathcal{U}$ to a set S is at least as large as adding e to any superset of S. This property is called submodularity for maximization problems and supermodularity for minimization problems.

In this work we develop a new exact branch-and-cut algorithm `SubModST` for the generic Submodular Cardinality-Constrained Maximization and Supermodular Cardinality-Constrained Minimization. We develop several speed-ups for `SubModST` and we show their effectiveness on six example problems. We show that `SubModST` outperforms the state-of-the-art solvers developed by Csókás and Vinkó [J. Glob. Optim. '24] and Uematsu et al. [J. Oper. Res. Soc. Japan '20] for Submodular Cardinality-Constrained Maximization by orders of magnitudes.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Branch-and-bound

Keywords and phrases Branch-and-Cut, Lazy Evaluations, Facility Location, Group Closeness Centrality, Partial Dominating Set

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.102

Supplementary Material *Software (Source Code)*: `https://github.com/HenningWoydt/SubModST`
archived at `swh:1:dir:dee9bf07a182bd8d04257ddb4d9d94b2342f46e4`
Dataset (Experimental data): `https://zenodo.org/doi/10.5281/zenodo.12579218`

Funding *Henning Martin Woydt*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – DFG SCHU 2567/6-1.

Acknowledgements The results of this work are based on the first author's Masters thesis [33].

1 Introduction

A vast number of computational problems can be formulated as the problem of selecting a small set of elements from a universe that optimizes a given objective function.

In operations research, the Facility Location problem receives as input a set of facility locations and a set of demand locations and asks for optimally placing a set of at most k facilities at the facility locations so that the sum of costs of the demand locations for reaching any facility is minimized [8, 9]. Essentially the same problem occurs in geometric clustering approaches, where it is called k-Medoid Clustering. Here the input is a set of data points, for example in a Euclidean space, and the task is to select k cluster centers, so that the sum of the distances of data points to their cluster centers is minimized [15, 30]. In social network analysis, Group Closeness Centrality is again the same problem, here motivated by identifying a group of central actors: the input is a graph and the task is to select k vertices so that the unselected vertices have a small distance to the set of selected vertices [5, 11].

Other important examples of these problems are covering problems. Here, a central problem is Maximum Coverage [1] where the task is to select k sets from a set family $\{S_1, S_2, \ldots, S_n\}$ over a ground set X so that the union of the k sets is maximized. Maximum Coverage is a generalization of other well-studied problems for example Dominating Set and Partial Dominating Set. Moreover, Set Cover and, similarly, Hitting Set, correspond to the special case of the decision version of Maximum Coverage where we ask whether there is a set of k sets whose union is X, the full ground set. In the even more general Weighted Coverage [17, 18] problem each element is associated with a weight and the task is to select a coverage such that the sum of the covered weights is maximized. Maximum Coverage and its special cases find applications in machine learning [3], in routing [34, 29], document summarization [27, 35] and sensor networks [21].

All of the problems described above are subsumed by the following generic problem.

Cardinality-Constrained Maximization (Minimization)
Input: A set function $f : 2^{\mathcal{U}} \to \mathbb{R}$ and an integer $k \in \mathbb{N}$.
Task: Find a set $S \subseteq \mathcal{U}$ with $|S| \leq k$ that maximizes (minimizes) $f(S)$.

The fact that Hitting Set and Maximum Coverage are special cases entails strong hardness results for Cardinality-Constrained Maximization: The problem is NP-hard [14], APX-hard [2], and assuming the Exponential Time Hypothesis (ETH) [13] is true, it cannot be solved in $n^{o(k)}$ time, where n denotes the total input size [6]. Note that Cardinality-Constrained Maximization is trivially solvable in $n^k \cdot n^{\mathcal{O}(1)}$ time assuming that f can be evaluated in polynomial time.

A striking common feature of the examples discussed above is, that they exhibit what is called the *diminishing returns* property. Informally, this property, called submodularity for maximization problems, implies that the improvement of the objective value obtained by adding an element e to a set S (this is called the marginal gain of e) is at least as large as when adding e to any superset of S. More formally, a set function f is submodular if $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$ for $A \subseteq B \subseteq \mathcal{U}$ and $e \in \mathcal{U}$. For minimization problems, the diminishing returns property is called supermodularity. Interestingly, by negating the objective function f, we not only transform minimization problems into maximization problems but we also transform supermodular problems into submodular ones. Hence, all of the problems mentioned above are special cases of Submodular Cardinality-Constrained Maximization where the objective function f is submodular. This makes the development of generic solvers for this problem both desirable from an application point of view and challenging from an algorithmic point of view. We present a new generic exact solver `SubModST` for Submodular Cardinality-Constrained Maximization, develop and evaluate several speed-ups for `SubModST` and compare it to other state-of-the-art solvers.

Related Work. Submodularity can be used to show that the natural greedy algorithm which always selects an element with the maximum marginal gain gives an approximation which is a factor of $(1 - 1/e)$ away from the optimum [22]. Minoux [20] improved the speed of the greedy algorithm by using lazy evaluations. These can be applied because, due to submodularity, not all elements have to be reevaluated in each iteration. Minoux's algorithm [20] gives the same set as the standard greedy algorithm but can be magnitudes faster [18].

A first exact algorithm was described by Nemhauser and Wolsey in 1981 [23]. In its most basic form, it solves the problem via a 0/1-ILP with exponentially many constraints. To mitigate the potentially prohibitive constraint number, new constraints are added iteratively. A further solver, also based on 0/1-ILPs with an iterative constraint generation method, was described in 2009 [16]. Chen at al. [7] developed a framework in 2015 in which the user can tune the quality of the returned set and the time needed to compute it. This tradeoff can be made via a hyperparameter $\alpha \in [0,1]$, with $\alpha = 0$ being a fast but inaccurate greedy approximation and $\alpha = 1$ being a slow but optimal solution. They utilize a A* search algorithm on the search space, that is, on the space of all subsets of size at most k of $\mathcal{U}$.

Subsequently, Uematsu et al. [31] revised the idea of Nemhauser et al. [23]; they developed an improved solver based on the constraint generation algorithm, called ICG, by heuristically generating a set of feasible solutions that generate new constraints. Moreover, to quickly improve the best-found lower bound the solver makes use of a local search heuristic. The comparison of these solvers showed that ICG substantially outperforms the previous approaches. Very recently, Csókás and Vinkó [10] further refined ICG by considering other heuristics for the constraint generation and by exploiting some structure of the input graphs for specific problems. The proposed solvers outperform ICG on most but not all of the considered benchmark instances [10]. Summarizing, the current state-of-the-art generic solvers for SUBMODULAR CARDINALITY-CONSTRAINED MAXIMIZATION are ICG [31] and its refined variants [10].

Some algorithms have been developed for special cases. The most similar to `SubModST` is a solver for GROUP CLOSENESS CENTRALITY [28] which is based on the enumeration of candidate solutions in a search tree. The state-of-the-art (SOTA) solver for GROUP CLOSENESS CENTRALITY is based on an ILP formulation which computes exact solutions for small values of k on graphs with up to 30 000 vertices [28].

Our Results. We develop a generic solver `SubModST` for SUBMODULAR CARDINALITY-CONSTRAINED MAXIMIZATION. `SubModST` accesses the function f as a black-box oracle that supplies the value $f(S)$ for any set $S \subseteq \mathcal{U}$. `SubModST` is based on an exact solver for GROUP CLOSENESS CENTRALITY [28]. Basically, the solver of Staus et al. [28] is a search-tree algorithm maintaining the current subset of the solution. The search-tree is efficiently pruned by exploiting the diminishing return of single elements. First, in Section 2 we generalize the ideas of this solver to SUBMODULAR CARDINALITY-CONSTRAINED MAXIMIZATION. Second, in Section 3, inspired by Minoux [20], we introduce Lazy Evaluations to avoid having to recompute the marginal gains of all candidates in each search-tree node. Finally, in Section 4 we present new heuristics using the diminishing return of pairs of candidates.

In Section 5 we present an extensive experimental evaluation of our approach. More precisely, we analyze `SubModST` on six example problems. First, we show the effectiveness of heuristics. Second, we compare `SubModST` to the SOTA solvers of Uematsu et al. [31] and Csókás and Vinkó [10] for SUBMODULAR CARDINALITY-CONSTRAINED MAXIMIZATION. Our approach is orders of magnitudes faster than their algorithms. For example, instances of FACILITY LOCATION solved by Uematsu et al. [31] in 500 seconds are solved by our solver in less than 1 second. We also compare `SubModST` to a special-purpose solver for GROUP CLOSENESS CENTRALITY [28]. While `SubModST` is much slower on the instances solved by both solvers, `SubModST` is able to find solutions for some graphs which cannot be solved by the SOTA [28].

Due to lack of space, correctness proofs for some speed-up techniques (marked by ($\star$)) and several details of the experimental analysis are deferred to a full version.

Preliminaries. A function $f : 2^{\mathcal{U}} \to \mathbb{R}$ for a *universe* $\mathcal{U} = [n] = \{0, 1, \dots, n-1\}$ is called a *set function*. Let $S \subseteq \mathcal{U}$ and $e \in \mathcal{U}$. The *marginal gain* (also called the *discrete derivative*) of f at S with respect to e is $\Delta_f(e \mid S) \coloneqq f(S \cup \{e\}) - f(S)$. Analogously, the *marginal gain* for set $A \subseteq \mathcal{U}$ of f at S with respect to A is $\Delta_f(A \mid S) \coloneqq f(S \cup A) - f(S)$ [18]. When f is evident from the context, we drop the subscript and write $\Delta(e \mid S)$. A function $f : 2^{\mathcal{U}} \to \mathbb{R}$ is *submodular* if for every $A \subseteq B \subseteq \mathcal{U}$ and $e \in \mathcal{U} \setminus B$ it holds that $\Delta(e \mid A) \geq \Delta(e \mid B)$. Equivalently, f is *submodular* if for every $A, B \subseteq \mathcal{U}$, $f(A \cap B) + f(A \cup B) \leq f(A) + f(B)$.

A set function f is *monotone* if for every $A \subseteq \mathcal{U}$ and every $e \in \mathcal{U}$, $\Delta(e \mid A) \geq 0$. Throughout the rest of this work, we consider only submodular and monotone set functions f (calling them score function) and assume that f is well-defined and computable on all inputs. Monotonicity is a mild technical condition which was also assumed in the other generic solvers [31, 10] for Submodular Cardinality-Constrained Maximization. It enables us to only focus on sets of size k. Since the marginal gain of any element e gradually declines as elements are added to the set A, we can compute an upper bound on the marginal gain for every subset $B \subseteq \mathcal{U}$ at $A \subseteq \mathcal{U}$ if we have the marginal gain of each element $e \in \mathcal{U}$. Lemma 1.1 is used extensively later in Sections 2 and 4 to prove the correctness of our numerous heuristics.

▶ **Lemma 1.1** (Folklore). *Let $A, B \subseteq \mathcal{U}$, and let $f : 2^{\mathcal{U}} \to \mathbb{R}$ be a submodular set function. Then $\sum_{e \in B} \Delta(e \mid A) \geq \Delta(B \mid A)$.*

2 The Basic Search Algorithm

SubModST is based on a branching algorithm from Staus et al. [28] for Group Closeness Centrality with worst-case running time $n^k \cdot n^{\mathcal{O}(1)}$. The ideas of their algorithm are not limited to Group Closeness Centrality, instead it can be used for the much more general Cardinality-Constrained Maximization. Next, we present their ideas and lift them to Cardinality-Constrained Maximization. Also, we present one further technique (Fast Pruning) which was not part of the original algorithm.

Staus et al. [28] use a Set-Enumeration Tree (SE Tree) [26] as a data structure to efficiently iterate over $[n]$. More precisely, each *node* $T = (S_T, C_T = \{c_1, \dots, c_m\})$ has a *working set* $S_T \subseteq [n]$ and a *candidate set* $C_T \subseteq [n]$ with $S_T \cap C_T = \emptyset$. The working set S_T is the current solution and the data structure ensures that the working set of each node is unique. The candidate set C_T contains all elements which can be added to enlarge S_T. For each $1 \leq i \leq m$ the *child* T_i *of* T is $T_i = (S_T \cup \{c_i\}, C_T \setminus \{c_1, c_2, \dots, c_i\})$. The *children of* T are $\mathrm{Ch}(T) = \{T_i : 1 \leq i \leq m\}$. The *descendants* of T are $\mathrm{Des}(T) = \mathrm{Ch}(T) \cup \left(\bigcup_{T_i \in \mathrm{Ch}(T)} \mathrm{Des}(T_i)\right)$. A *depth-$k$-limited Set-Enumeration Tree* (*k-SE Tree*) for $k \in \mathbb{N}$ consists of a root node R with $S_R = \emptyset$ and $C_R = [n]$, and all descendants of R having a working set of size at most k. By s_{best} we denote the score of the best set the algorithm has found so far. The *best remaining subset* (with respect to node T) is $S^*_{C_T} = \arg\max_{S' \subseteq C_T, |S'| = k - |S_T|} f(S_T \cup S')$. By $\mathbb{T} = \mathrm{Des}(R) \cup \{R\}$ we denote the set of all nodes of the SE Tree. Basic simply explores the k-SE Tree starting in root R. Clearly, Basic has running time $\mathcal{O}(n^k \cdot T_f(n))$, where T_f denotes the time needed to evaluate f. Note that Basic exploits monotonicity since it only evaluates f at working sets of size k. The Basic search algorithm can be seen in Algorithm 1, it skips lines 3, 5 and 6.

Algorithm 1 Pseudocode of the `SESearch` algorithm. The initial call is `SESearch`$((\emptyset, [n]), k, -\infty)$.

```
1 Function SEsearch (T, k, s_best)
      Input: A node $T$, the budget $k$, and the so far best achieved score $s_{\text{best}}$.
      Output: The maximum score.
2     if $|S_T| = k$ then return $\max(s_{\text{best}}, f(S_T))$          // check for solution
3     Apply Dynamic Candidate Ordering with Fast Pruning and Lazy Evaluation
4     while $|C_T| \geq k$ do
5         Apply Heuristics (such as SUB or PW) to prune $T$
6         Apply Candidate Reduction to $C_T$
7         $c :=$ pop first element of $C_T$
8         $s_{\text{best}} =$ SEsearch $((S_T \cup \{c\}, C_T), k, s_{\text{best}})$
9     return $s_{\text{best}}$
```

Search Tree Pruning. To avoid traversing the whole search space, we use pruning rules to remove candidates or whole subtrees. Formally, *pruning a set of candidates* C at node T in a k-SE Tree is the action of setting $C_T = C_T \setminus C$. If $C = C_T$, we say that we *prune node* T and if $C = \{c\}$, we say that we *prune candidate* c. Pruning is safe when it does not remove any nodes with a score that is better than the current best. This is formalized as follows.

▶ **Lemma 2.1.** *A node T of a k-SE Tree can be pruned if $f(S_T) + \Delta(S^*_{C_T} \mid S_T) \leq s_{best}$.*

In this form, the lemma is not helpful since it depends on $S^*_{C_T}$, which is not known in advance. We need ways to determine an upper bound on the marginal gain of $S^*_{C_T}$.

▶ **Definition 2.2.** *Let T be a node in a k-SE Tree. We call a function $h : \mathbb{T} \times 2^{[n]} \times \mathbb{N} \to \mathbb{R}$ a* valid heuristic *if $\max_{S \subseteq C_T, |S| = k - |S_T|} f(S_T \cup S) \leq h(T, C_T, k)$. We call the value $h(T, C_T, k)$ a* valid upper bound.

Next, we introduce a method to quickly find good sets with a large score, a heuristic for pruning a node T, and a function for pruning a set of candidates C. All three methods were described by Staus et al. [28] for Group Closeness Centrality. Afterwards, we introduce a new method that identifies as early as possible if a node can be pruned.

Dynamic Candidate Ordering. The children of a node T are explored based on the ordering in $C_T = \{c_1, \dots, c_m\}$. We choose the following candidate ordering to quickly obtain large values for s_{best}. *Dynamic Candidate Ordering* reindexes the candidates such that $\Delta(c_1 \mid S_T) \geq \Delta(c_2 \mid S_T) \geq \ldots \geq \Delta(c_m \mid S_T)$. Applying the ordering costs $\mathcal{O}(n \cdot T_f)$ time to evaluate the score function and additionally $\mathcal{O}(n \log n)$ time to sort the candidates. This would result in a slower algorithm without further improvements. Observe that now the first leaf the algorithm finds, corresponds to the solution of the greedy algorithm and hence, has a value of at least $(1 - 1/e) \cdot f(S_{\max})$ where $S_{\max}$ is a maximizing set [22].

Simple Upper Bound. The *Simple Upper Bound (SUB)* heuristic is $\text{SUB}(T, C_T, k) := f(S_T) + \max_{S' \subseteq C_T, |S'| = k - |S_T|} \sum_{e \in S'} \Delta(e \mid S_T)$. SUB is valid, due to the submodularity of f. When using Dynamic Candidate Ordering, the upper bound can be computed in $\mathcal{O}(k)$ time. After returning from a child with candidate c, we can update the upper bound in $\mathcal{O}(1)$ time by removing the marginal gain of c and adding the next best marginal gain.

Candidate Reduction. The idea of *Candidate Reduction (CR)* is to remove candidates whose marginal gain is too low to be part of a maximizing set. Let $C \subseteq C_T$ be the set of candidates c_i with $\mathrm{SUB}(T, C_T \setminus \{c_i\}, k-1) + \Delta(c_i \mid S_T) \leq s_{\text{best}}$. The idea is that candidates in C need not be considered by the algorithm as shown in the following lemma.

▶ **Lemma 2.3** ($\star$)**.** *If $c \in C$, then c is not part of a set with a larger score than s_{best}.*

The Candidate Reduction function is to set $C_T := C_T \setminus C$. By using Dynamic Candidate Ordering, we can start the candidate reduction with the candidate with lowest marginal gain and stop as soon as we identify the first candidate that violates the above inequality.

Fast-Pruning. The aim of *Fast-Pruning*, which is not part of the branching algorithm of Staus et al. [28], is to prune a node during Dynamic Candidate Ordering. To prune after the update of the ith marginal gain two conditions must be fulfilled: (i) The sum of the $k - |S_T|$ largest *updated* marginal gains is at most $s_{\text{rem}} = s_{\text{best}} - f(S_T)$ and (ii) the largest *not yet updated* marginal gain is upper-bounded by the smallest marginal gain of the $k - |S_T|$ chosen marginal gains. The first condition ensures that the sum of the updated marginal gains does not surpass s_{rem}, while the second condition ensures that all not-updated marginal gains can not increase this sum. Fast Pruning can be implemented efficiently with a Min-Heap which stores the $k - |S_T|$ largest updated elements. In the worst-case, the pruning condition is never fulfilled, and the Fast-Pruning technique has a total running time of $\mathcal{O}(n \log k)$ for a search tree node. This is dominated by the time needed for Dynamic Candidate Ordering.

3 Lazy Evaluations

The bottleneck of the search algorithm is the computation of the Dynamic Candidate Ordering in each of the $\Theta(n^k)$ nodes which consumes $\mathcal{O}(n \cdot T_f)$ time per node. However, we observed two interesting behaviors. First, the algorithm only expands a fraction of its children due to SUB and CR. Second, candidates with a small marginal gain are unlikely to be part of a maximizing set. Thus, calculating marginal gains for these candidates seems to be wasted time. We address this observation with Lazy Evaluations, also used by Minoux [20] for the greedy algorithm. Generally speaking, we do not update the marginal gains of some candidates, and instead we reuse their marginal gain from the parent. Observe that this approach might enlarge the search space and change the order in which nodes are traversed.

▶ **Definition 3.1.** *Let T be a node in an k-SE Tree, and let T^* be its parent. The function $U_T : \mathcal{U} \to \mathbb{R}$ subject to some predicate π is defined via*

$$U_T(c) := \begin{cases} \Delta(c \mid S_T) & \text{if } \pi \text{ is true,} \\ U_{T^*}(c) & \text{otherwise} \end{cases}$$

is called an update scheme *with respect to node T. $U_T(c)$ is the* lazy marginal gain.

Note that the existence of a parent is necessary. Thus, we assign the *default update scheme* $\mathrm{DU} : \mathcal{U} \to \mathbb{R}$ with $\mathrm{DU}(c) := \Delta(c \mid S_T)$ to the root. To apply Dynamic Candidate Ordering on non-root node T with update scheme U_T, the candidates are reindexed such that $U_T(c_1) \geq U_T(c_2) \geq \ldots \geq U_T(c_m)$. A necessary condition for any useful update scheme is that the search algorithm does not lose its exactness guarantee. Hence, an update scheme U_T is *valid* if $U_T(c) \geq \Delta(c \mid S_T)$ for each candidate $c \in C_T$.

▶ **Lemma 3.2** ($\star$)**.** *Let T be a node in a k-SE Tree, and let U_T be a valid update scheme. Each valid heuristic h that uses $U_T(c)$ instead of $\Delta(c \mid S_T)$ stays valid.*

Notice that by using update schemes the heuristic becomes less sharp and hence its less likely that nodes are pruned. Now, it is essential to skip some evaluations while not enlarging the search space too much. For this, we present two specific update schemes.

Score Update Scheme. The *score update scheme* $U_T^{\mathrm{avg}}(c)$ only calculates the marginal gain of a candidate c at node T if the lazy marginal gain of the candidate at parent node T^* was greater than the average required marginal gain, that is, if $U_{T^*}(c) \geq r_{\mathrm{avg}}$ where $r_{\mathrm{avg}} = (s_{\mathrm{best}} - f(S_T))/(k - |S_T|)$. The hope is that candidates c with $U_{T^*}(c) \geq r_{\mathrm{avg}}$ receive a marginal gain that is below the average marginal gain required. If this aim is achieved for every candidate $c \in C_T$, then SUB prunes the node T, and we have saved the evaluations for all candidates with $U_{T^*}(c) < r_{\mathrm{avg}}$. The threshold of r_{avg} is specifically set for SUB, but any threshold is applicable, and there is no guarantee that one threshold is better than the other. One may use a hyperparameter to scale r_{avg}, that is, the update scheme compares $U_{T^*}(c) \geq y \cdot r_{\mathrm{avg}}$. Observe that if $y = 0$, then the score update scheme behaves like the default update scheme, and if $y = \infty$ no candidate is ever updated.

Rank Update Scheme. The *rank update scheme* $U_T^{\ell}(c)$ only updates the marginal gain of candidate c if c has a sufficiently high rank in the candidate set of the parent, that is, if $\mathrm{rank}(c, C_{T^*}) \leq \ell(T)$, where $\mathrm{rank}(c, C_{T^*})$ returns the rank of c in the set C_{T^*} (the candidate with the greatest marginal gain has rank 1). If Dynamic Candidate Ordering was applied to the parent node T^*, then determining each candidate's rank is straightforward. This scheme is only as powerful as the ranking function $\ell(T)$. In our experiments we use $\ell(T) = 3(k - |S_T|)$. Preliminary experiments showed that choosing $\ell(T) = |C_T| \cdot y$ or $\ell(T) = n \cdot y$ for some $y \in [0, 1]$ does not yield smaller running times. Note that a function with $\ell(T) \geq |C_T|$ results in the default update scheme, and $\ell(T) = 0$ never updates any candidate.

Combination of Update Schemes. We combine score and rank schemes as follows: We define the *score or rank update scheme* $U_T^{\vee}(c)$ where $U_T^{\vee}(c) = \Delta(c \mid S_T)$ if $\pi = U_{T^*}(c) \geq r_{\mathrm{avg}} \vee \mathrm{rank}(c, C_{T^*}) \leq l(T)$ is true. We also use the *score and rank update scheme* which is defined similarly with an $\wedge$ instead of an $\vee$.

4 Advanced Heuristics

Until now, we only considered marginal gains of single candidates. Now, we extend this to candidate sets of size 2, called *pairs*, that is, we give a valid upper bounds for the best remaining set $S^*_{C_T}$ in T with size $k' = k - |S_T|$ where we choose $k'/2$ pairs if k' is even and $(k'-1)/2$ pairs and one single marginal gain if k' is odd. The hope is that $\Delta(\{u, w\} \mid S_T)$ is much smaller than $\Delta(u \mid S_T) + \Delta(w \mid S_T)$ and thus we can obtain sharper upper bounds. Ideally, we want that no element is used in at least two pairs; this property is referred to as *pair disjointness*. Note that the best remaining set $S^*_{C_T}$ can always be partitioned into disjoint pairs. Clearly, testing all different pair disjoint sets consisting of k' elements is not feasible. Hence, we use maximum-weight matching which is much faster. Second, we describe a greedy approach which does not fulfill the pair disjointness property but which is much faster than the maximum matching approach. Here, we assume that k' is even. The proof for the case that k' is odd is deferred to a full version of this article. For both approaches the number of candidate pairs is still too large. Thus, we then describe a hybrid approach where C_T is partitioned into two sets P_1 and P_2 and only for P_1 we use a pairwise heuristic.

4.1 Approaches for Candidate Sets of Size 2

Maximum-Weight Matching and Matching Graph. Recall that a *matching* in a graph G is a set $E' \subseteq E(G)$ such that no two edges in E' are incident, and E' is a *maximum-weight matching for G with weight function ω of size ℓ* if there is no size ℓ-matching E^* with larger weight. We interpret the candidate set C_T as vertices and the pairs as edges. The weight of an edge e is the marginal gain of the corresponding pair of candidates. Note that the resulting graph G_T is a clique. A maximum-weight matching of size $k'/2$ would yield a valid upper-bound with pair disjointness. Additionally the bound would be smaller than SUB. In order to find a maximum-weight matching of size $k'/2$, we will build a new graph H_T on which we can compute a maximum-weight matching and extract the relevant edges.

Starting from G_T, we build a new graph, which we call the *matching graph for k' vertices H_T* which is a complete split graph in which the clique corresponds to $V(G_T)$ and the independent set consists of $n - k'$ newly added vertices. Formally, $V(H_T) = V(G_T) \cup V_I$ where $V_I = \{u_i : 1 \leq i \leq n - k'\}$ is the independent set. The weight of each edge e with both endpoints in $V(G_T)$ is equal to the weight of e in G_T, and the weight of each edge with exactly one endpoint in V_I is 0. Next, we verify that a maximum-weight perfect matching on H_T contains a maximum-weight matching of size $k'/2$ for G_T.

▶ **Lemma 4.1** (⋆). *Let H_T be a matching graph for k' vertices. Let E' be a maximum-weight perfect matching of H_T and let $E'_{G_T} \subseteq E'$ be the restriction of H_T to G_T. Then, E'_{G_T} has size $k'/2$ and for each other matching E^* of size $k'/2$ in G_T we have $\omega(E^*) \leq \omega(E'_{G_T})$.*

Now, we show that using the marginal gains as edge weights in G_T yields a valid heuristic for Cardinality-Constrained Maximization.

▶ **Definition 4.2.** *Let T be a node in a k-SE Tree, and $k' = k - |S_T|$ even. Let G_T be the clique graph formed by the candidates of C_T, and let $E' \subseteq E(G_T)$ be a maximum-weight matching of size $k'/2$ for G_T. The* Pairwise Matching *heuristic is*

$$PW_M(T, C_T, k) := f(S_T) + \sum_{c \in E'} \Delta(c \mid S_T).$$

▶ **Lemma 4.3** (⋆). *The Pairwise Matching heuristic is valid and has running time $\mathcal{O}(n^5)$.*

Greedy Approach. The greedy approach simply chooses the $k'/2$ pairs with the greatest marginal gain regardless of pair disjointness. Note that the greedy approach never has a smaller upper bound than the matching approach and it can even have a worse bound than SUB. However, if the variance of the marginal gains of the $\mathcal{O}(n^2)$ pairs is small, then the greedy approach provides a relatively sharp upper bound significantly faster than the matching approach. Observe that all heuristics require at least $\mathcal{O}(n^2 T_f)$ time, since the marginal gains of all pairs need to be computed.

▶ **Definition 4.4.** *Let T be a node in a k-SE Tree, $k' = k - |S_T|$ even, and let $P(C_T)$ be the set of all pairs of C_T. The* Pairwise Greedy *heuristic is*

$$PW_G(T, C_T, k) := f(S_T) + \max_{P \subseteq P(C_T), |P| = k'/2} \sum_{p \in P} \Delta(p \mid S_T).$$

▶ **Lemma 4.5** (⋆). *The Pairwise Greedy heuristic is valid and has running time $\mathcal{O}(n^2 \log k')$.*

4.2 Avoiding Practical Limitations

Until now, we have assumed that we use all available $\mathcal{O}(n^2)$ pairs, but such an approach requires $\Omega(n^2 \cdot T_f)$ running time to calculate all marginal gains. Additionally, a running time for computing the upper bound of $\mathcal{O}(n^5)$ (matching) or $\mathcal{O}(n^2 \log k')$ (greedy) is necessary. Since this is quite time-consuming, we partition the candidate set C_T into two parts P_1 and P_2. Intuitively, P_1 contains all elements with high marginal gains and P_2 contains all elements with low marginal gains. Now, we only use the pairwise heuristics for P_1, and the much faster SUB heuristic for P_2. However, this partitioning does come with additional costs: previously we knew that $S^*_{C_T} \subseteq C_T$ with $|S^*_{C_T}| = k'$. Now, we do not know how $S^*_{C_T}$ is distributed across P_1 and P_2, and theoretically all distributions are possible.

▶ **Definition 4.6.** *Let T be a node in a k-SE Tree, and $k' = k - |S_T|$. Let P_1, P_2 be a partition of C_T with $|P_1| = \ell$ and $\forall a \in P_1, \forall b \in P_2 : U_T(a \mid S_T) \geq U_T(b \mid S_T)$. Let h be a valid heuristic. The* Partitioning *heuristic is*

$$PT_h(T, C_T, k) := \max_{0 \leq i \leq k'} \left(h(T, P_1, k - k' + i) + SUB(T, P_2, k - i) - f(S_T) \right).$$

▶ **Lemma 4.7** ($\star$). *The Partitioning heuristic is valid.*

For the running time, observe that if Dynamic Candidate Ordering is applied to a node, then the Simple Upper Bound heuristic can be pre-computed and then performed in $\mathcal{O}(1)$ and does not increase any complexity. Hence, the parameter n in the running time of h is replaced by the new hyperparameter ℓ, but we get an additional factor of k'.

Clearly, there is a trade-off for the hyperparameter ℓ: a small value of ℓ speeds up the computation but the result is less sharp. In our experiments we consider two choices for ℓ: One intuitive choice is $\ell = \sqrt{n}$. Then, the time complexity of the $\mathcal{O}(\ell^2)$ score function evaluations is $\mathcal{O}(n \cdot T_f)$, and the space complexity is $\mathcal{O}(n)$; matching the time needed for Dynamic Candidate Ordering. We also use $\ell = k'$ in our experiments.

5 Experiments

To evaluate our solver, we perform experiments on the following six problems:

(a) Group Closeness Centrality. Here, we aim to select a vertex set in a graph with a small distance to the vertices of graph. More formally, we aim to minimize the following measure.

▶ **Definition 5.1** (Group Farness). *Let $G = (V, E)$ be a connected undirected graph. Let* $\mathrm{dist}(u, v)$ *be the distance of a shortest path between the vertices u and v. Let $S \subseteq V$ be a subset of vertices. The* group farness *of S is*

$$f_{\mathrm{GF}}(S) := \sum_{v \in V \setminus S} \min_{u \in S} \mathrm{dist}(u, v).$$

In other words, the group farness of a set S is the sum of the minimum distances of each vertex not in S to any vertex in S. Note that the function value of the empty set is not properly defined, so we define it in this work as $f_{\mathrm{GF}}(\emptyset) := |V|^2$. Note that f_{GF} is supermodular and monotone (decreasing). Based on group farness, we can now properly define the problem.

Group Closeness Centrality
Input: A connected undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.
Task: Find a set $S \subseteq V$ with $|S| = k$ that has the lowest group farness $f_{\mathrm{GF}}(S)$.

Table 1 The ten graphs used for **(a)** Group Closeness Centrality and **(b)** Partial Dominating Set (left) and the ten cluster datasets used for **(f)** Euclidean k-Medoid Clustering (right).

Name	# Vertices	# Edges	diameter	density	Name	# Points	# Dim	# Clusters
ca-netscience	379	913	17	0.013	skewed	1 000	2	6
soc-wiki-Vote	889	2 914	13	0.007	asymmetric	1 000	2	5
bio-yeast	1 458	1 948	19	0.002	overlap	1 000	2	6
econ-orani678	2 529	86 768	5	0.027	dim032	1 024	32	16
soc-advogato	5 054	39 374	9	0.003	a1	3 000	2	20
bio-dmela	7 393	25 569	11	0.001	s1	5 000	2	15
ia-escorts-dynamic	10 106	39 016	10	0.001	s2	5 000	2	15
soc-anybeat	12 645	49 132	10	0.001	a2	5 250	2	35
ca-AstroPh	17 903	196 972	14	0.001	unbalance2	6 500	2	8
fb-pages-media	27 917	205 964	15	0.001	a3	7 500	2	50

(b) Partial Dominating Set. In this problem, we aim to select a vertex set with many neighbors in a graph $G = (V, E)$. Formally, the *open neighborhood* $N(u)$ of $u \in V$ is the set $\{w : \{u, w\} \in E\}$ and the *closed neighborhood* of u is $N[u] := N(u) \cup \{u\}$. For a vertex set $S \subseteq V$, the *vertex domination number* is $f_{\mathrm{D}} := |\bigcup_{u \in S} N[u]|$. For $S = \emptyset$, we set $f_{\mathrm{D}}(S) = 0$.

Partial Dominating Set
Input: An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.
Task: Find a $S \subseteq V$ with $|S| = k$ that has maximum $f_{\mathrm{D}}(S)$.

(c) Facility Location. This variant of the facility location problem was described by Uematsu et al. [31]. Let $N = \{1, \dots, n\}$ be a set of locations and $M = \{1, \dots, m\}$ a set of customers. The value $w_{ij} > 0$ is the profit for customer $i \in M$ if served by facility $j \in N$. Let $G = (N \cup M, E)$ be a fully connected bipartite graph with edge weights w_{ij}. For a subset $S \subseteq N$ of selected facilities, each customer is served by a facility with the greatest profit.

Facility Location
Input: A graph $G = (N \cup M, E)$ and an integer $k \in \mathbb{N}$.
Task: Find a $S \subseteq N$ with $|S| = k$ that maximizes

$$f(S) = \sum_{i \in M} \max_{j \in S} w_{ij}.$$

(d) Weighted Coverage. Here, we are given a collection of item sets and want to select k of them such that their union has a maximum total weight. Let $\bigcup S := \bigcup_{s_i \in S} s_i$.

Weighted Coverage
Input: A collection $N = \{s_1, \dots, s_n\}$ of subsets of an item set $M = \{1, \dots, m\}$, a weight function $\omega : M \to \mathbb{R}$ and an integer $k \in \mathbb{N}$.
Task: Find a $S \subseteq N$ with $|S| = k$ that maximizes $f(S) = \sum_{i \in \bigcup S} \omega(i)$.

(e) Bipartite Influence. Let $N = \{1, \dots, n\}$ be a set of sources and $M = \{1, \dots, m\}$ a set of targets. Let $G = (N \cup M, E)$ be a bipartite directed graph with $E \subseteq N \times M$. Let $0 \leq p_{ij} \leq 1$ be the activation probability of edge (j, i) for target $i \in M$ and source $j \in N$. If the edge does not exist in G then $p_{ij} = 0$. A target $i \in M$ is activated by a set $S \subset N$ of sources with probability $1 - \prod_{j \in S}(1 - p_{ij})$.

BIPARTITE INFLUENCE
Input: A Graph $G = (N \cup M, E)$, probabilities p_{ij} and an integer $k \in \mathbb{N}$.
Task: Find a $S \subseteq N$ with $|S| = k$ that maximizes

$$f(S) = \sum_{i \in M} \left(1 - \prod_{j \in S} (1 - p_{ij})\right).$$

(f) Euclidean k-Medoid Clustering. This problem is a geometric clustering problem with the following objective function.

▶ **Definition 5.2** (Clustering Cost). *Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of n data points in $\mathbb{R}^n$. Let $d(x_i, x_j)$ be a distance function, and $S \subseteq X$ be a subset of the data points. The* clustering cost *of S with distance function d is*

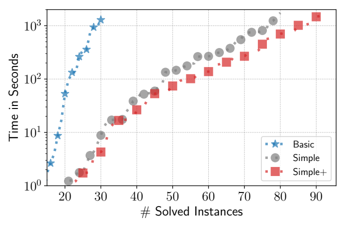
$$f_{\mathrm{CC}}(S) := \sum_{i=1}^{n} \min_{x_j \in S} d(x_i, x_j).$$

Since the clustering cost is not defined for the empty set, we define $f_{\mathrm{CC}}(\emptyset)$ as the sum of all distances between all data points. Like group farness f_{GF}, the clustering cost function d is supermodular and monotone (decreasing).

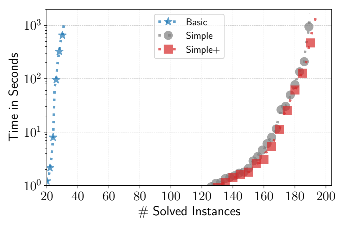
EUCLIDEAN k-MEDOID CLUSTERING
Input: A set X consisting of n datapoints, the Euclidean distance function $d(x_i, x_j)$, and an integer $k \in \mathbb{N}$.
Task: Find a set $S \subseteq X$ with $|S| = k$ that has the lowest clustering cost $f_{\mathrm{CC}}(S)$.

Data and Experimental Setup. For problems **(a)** and **(b)** we use 10 graphs from the Network Repository [25], where we always only use the largest connected component of each graph (see Table 1). These graphs are (mostly) a subset of those used by Staus et al. [28]. For problems **(c)** and **(d)** we used the benchmark data provided by Uematsu et al. [31]. In this data set, the size ranges from $|N| = 20, \ldots, 100$ and $|M| = |N| + 1$. For problem **(e)** we use the benchmark data provided by Csókás et al. [10]. The problem sizes are identical to the ones of the benchmark set of Uematsu et al. [31]. We refer the reader to Uematsu et al. [31] and Csókás et al. [10] for more information on the data for problems **(c)**, **(d)** and **(e)**. We used the same data as them to have a fair comparison. For **(f)** we use six synthetic datasets proposed by Fränti et al. [12] and four proposed by Rezaei et al. [24]. Table 1 gives a description.

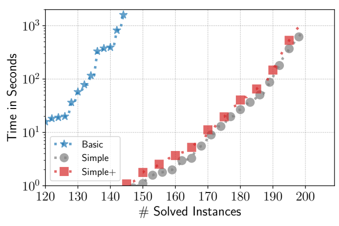
Our algorithm is implemented in C++ using the C++ 17 Standard. We use `-O3` as a compilation flag to generate the most efficient machine code. The used machine has an Intel® Xeon® Gold 6230 Processor with 2.1 GHz. and 96GB RAM running Red Hat Enterprise Linux (RHEL) 8.4. The code is compiled using GCC compiler version 13.2.0 and CMake version 3.23.3 as build tool. We limit the experiments by only allowing $k \in \{1, \ldots, 20\}$ and set the time limit to 30 minutes. Should an algorithm not solve an instance for k in the time limit, we assume that all runs with greater k will also time out, and therefore they will not be started. Any preprocessing is not included in the time measurements.



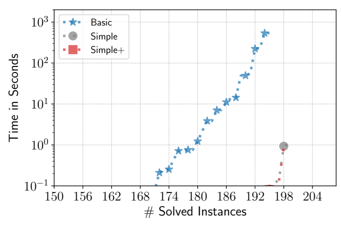
(a) Group Closeness Centr.



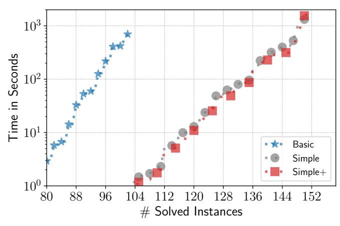
(b) Partial Dominating Set



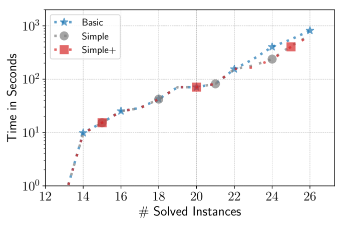
(c) Facility Location



(d) Weighted Coverage



(e) Bipartite Influence



(f) Euclidean Clustering

Figure 1 Number of solved instances of the three basic algorithms for each of the six problems. In total 200 instances can be solved per problem.

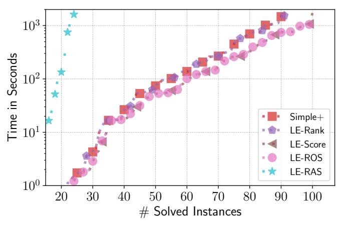
5.1 Comparison of Different Versions of our Solver

Simple Search Algorithm. We first evaluate the ideas of Section 2. Recall that `Basic` employs complete search without pruning, `Simple` is the generalized version of the solver of Staus et al. [28], and `Simple+` additionally uses Fast-Pruning. Figure 1 shows the number of solved instances for each of the six problems. Table 2 gives further results for each instance. Except for Euclidean Clustering (which is very hard for all versions of the algorithm), `Basic` is by far the slowest version. `Simple+` performs better than `Simple` for all problems except Facility Location. Additional profiling has shown that the heap management takes a significant amount of runtime of `Simple+` for instances of Facility Location. This however can still be improved by a faster and more specialized heap.

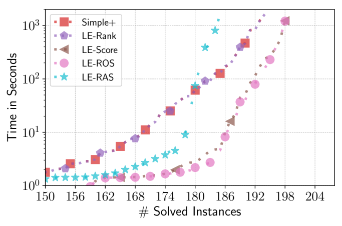
Lazy Evaluation. Next, we compare `Simple+`, the best version so far, against versions using Lazy Evaluations. We use the following four configurations: `LE-Score` is the score update scheme with $y = 1$, `LE-Rank` is the rank update scheme with $l(T) = 3(k - |S_T|)$, `LE-ROS` is the $\vee$ (or) combination of the previous two schemes and `LE-RAS` is the $\wedge$ (and) combination of the first two schemes. Figure 2 shows the number of solved instances for each of the six problems. Table 2 gives further results for each instance.

`LE-Rank` has almost the same running time as `Simple+`, while `LE-RAS` is much slower. `LE-Score` and `LE-ROS` are faster than `Simple+` with `LE-Score` being slightly faster. `LE-Score` is very beneficial for problems **(a)** and **(b)** and has almost no effect for **(c)** and **(e)**. For **(d)** and **(f)** no statement is possible, since the instances are too easy (**(d)**) or too hard (**(f)**). For some instances `LE-Score` was able to solve three greater k as `Simple+` (econ-orani678 for **(b)**) and for other instances it halved the running time (inf_60_5).

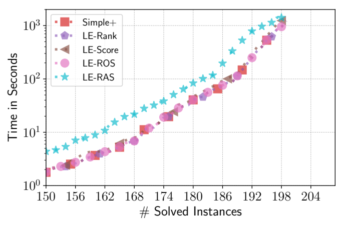
Pairwise Heuristics. Next, we compare `LE-Score` against versions using pairwise heuristics. We test four configurations: the Pairwise Greedy heuristic with $\ell(T) = k - S_T$ (denoted as `G-`k') and with $\ell(T) = \sqrt{|C_T|}$ (denoted as `G-`$\sqrt{n'}$), and the Pairwise Matching heuristic with



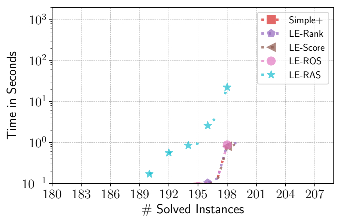
(a) Group Closeness Centr.



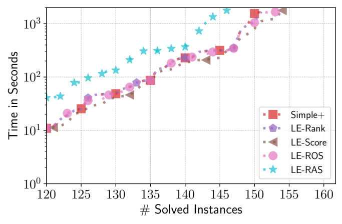
(b) Partial Dominating Set



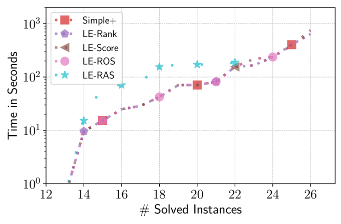
(c) Facility Location



(d) Weighted Coverage

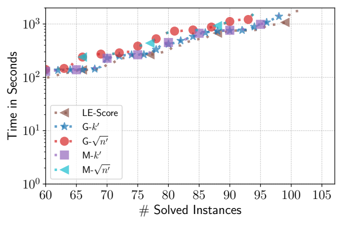


(e) Bipartite Influence

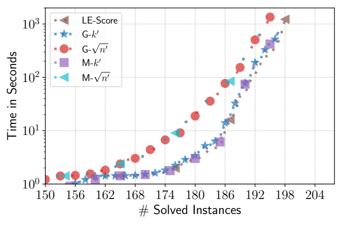


(f) Euclidean Clustering

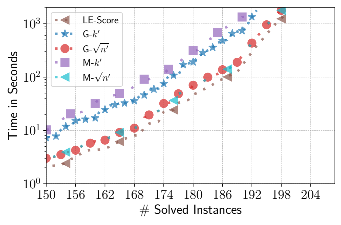
Figure 2 Number of solved instances of the four Lazy Evaluation algorithms for each of the six problems. In total 200 instances can be solved per problem.



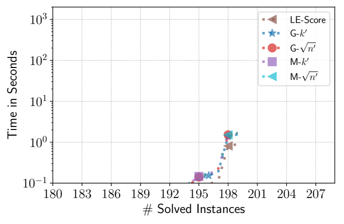
(a) Group Closeness Centr.



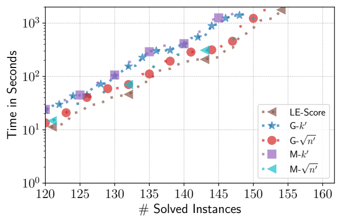
(b) Partial Dominating Set



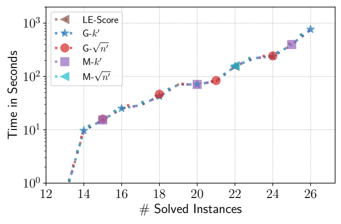
(c) Facility Location



(d) Weighted Coverage



(e) Bipartite Influence



(f) Euclidean Clustering

Figure 3 Number of solved instances of the four Pairwise algorithms for each of the six problems. In total 200 instances can be solved per problem. `LE-Score` still has the overall fastest running time.

$\ell(T) = k - S_T$ (denoted as M-k') and with $\ell(T) = \sqrt{|C_T|}$ (denoted as M-$\sqrt{n'}$). Figure 3 shows the numbers of solved instances. Sadly no configuration surpasses `LE-Score`. There is also no clear rule on which configuration should be preferred. While G-k' and M-k' are faster than G-$\sqrt{n'}$ and M-$\sqrt{n'}$ for Partial Dominating Set and Group Closeness Centrality, it is reversed for Facility Location and Bipartite Influence.

Using Oracles. Figure 3 shows that using a pairwise heuristic does not improve the running time. Next, we analyze whether there can be a faster implementation using candidate pairs. To this end, we introduce *oracles* to outline the potential of using candidate pairs. An oracle returns the solution of an heuristic for free. Thus, an oracle provides a lower bound on the best achievable time of any implementation.

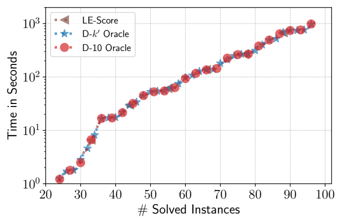
Table 2 Comparison of the Simple and Lazy Evaluation algorithms. The Lazy Evaluation algorithms use `Simple+` as basis. For each instance and algorithm the largest solved k and the required time to solve it in seconds are shown. Bold numbers indicate that an algorithm solved the largest k out of any algorithm. `Simple+` is the fastest Simple algorithm and `LE-score` successfully improves the running time. For Weighted Coverage every algorithm solved each instance for $k \in [20]$ in under 1 second.

	Instance	`Basic`		`Simple`		`Simple+`		`LE-Score`		`LE-Rank`		`LE-ROS`		`LE-RAS`	
		k	seconds	k	seconds	k	seconds	k	seconds	k	seconds	k	seconds	k	seconds
Partial Dom. Set	ca-netscience	5	324	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1
	soc-wiki-Vote	4	95	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1
	bio-yeast	4	652	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1
	econ-orani678	3	7	14	1 649	17	1 263	**19**	1 228	17	1 265	**19**	1 214	6	429
	soc-advogato	3	70	**20**	7	**20**	4	**20**	< 1	**20**	4	**20**	< 1	**20**	3
	bio-dmela	3	217	**20**	11	**20**	11	**20**	< 1	**20**	10	**20**	< 1	**20**	< 1
	ia-escorts-dynamic	3	544	**20**	7	**20**	7	**20**	< 1	**20**	7	**20**	< 1	**20**	< 1
	soc-anybeat	3	1 072	**20**	2	**20**	1	**20**	< 1	**20**	1	**20**	< 1	**20**	< 1
	ca-AstroPh	2	< 1	17	941	18	1 656	**20**	266	18	1 649	**20**	162	**20**	1 606
	fb-pages-media	2	2	**20**	185	**20**	168	**20**	4	**20**	173	**20**	2	**20**	4
Group Close. Cen.	ca-netscience	5	358	**14**	1 743	**14**	1 104	**14**	968	**14**	1 105	**14**	979	5	1 619
	soc-wiki-Vote	4	132	12	1 489	**13**	1 521	**13**	752	**13**	1 536	**13**	796	3	14
	bio-yeast	4	956	10	725	**11**	1 080	**11**	458	**11**	1 085	**11**	482	3	112
	econ-orani678	3	8	11	532	14	1 048	**17**	1 724	14	1 061	16	975	3	1 264
	soc-advogato	3	95	9	1 142	12	1 470	**14**	1 750	12	1 438	13	809	2	16
	bio-dmela	3	351	7	1 237	8	1 036	**9**	700	8	1 041	**9**	721	2	51
	ia-escorts-dynamic	3	936	6	347	7	698	**8**	675	7	700	**8**	687	2	134
	soc-anybeat	3	1 288	6	762	7	1 018	**10**	1 769	7	1 038	9	1 053	2	265
	ca-AstroPh	2	734	5	859	5	816	**6**	994	5	809	**6**	980	2	749
	fb-pages-media	**1**	1	**1**	1	**1**	1	**1**	1	**1**	1	**1**	1	**1**	1
Clustering	skewed	**3**	28	**3**	28	**3**	28	**3**	28	**3**	28	**3**	28	**3**	171
	asymmetric	**4**	557	**4**	421	**4**	404	**4**	390	**4**	404	**4**	389	3	171
	overlap	**4**	815	**4**	662	**4**	649	**4**	756	**4**	643	**4**	756	3	171
	dim032	**4**	403	**4**	179	**4**	166	**4**	218	**4**	166	**4**	218	3	188
	a1	**2**	15	**2**	15	**2**	15	**2**	15	**2**	15	**2**	15	**2**	15
	s1	**2**	70	**2**	70	**2**	70	**2**	70	**2**	70	**2**	70	**2**	70
	s2	**2**	70	**2**	70	**2**	70	**2**	70	**2**	70	**2**	71	**2**	70
	a2	**2**	81	**2**	81	**2**	81	**2**	81	**2**	81	**2**	81	**2**	81
	unbalance2	**2**	154	**2**	154	**2**	155	**2**	155	**2**	155	**2**	155	**2**	154
	a3	**2**	237	**2**	237	**2**	237	**2**	237	**2**	237	**2**	237	**2**	237
Fac. Location	L.20.5.1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1
	L.20.8.1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1
	L.30.5.1	**20**	5	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1
	L.30.8.1	**20**	5	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1
	L.40.5.1	13	828	**20**	1	**20**	3	**20**	4	**20**	3	**20**	3	**20**	5
	L.40.8.1	13	821	**20**	3	**20**	6	**20**	8	**20**	6	**20**	6	**20**	8
	L.50.5.1	11	1 601	**20**	52	**20**	87	**20**	115	**20**	87	**20**	88	**20**	115
	L.50.8.1	11	1 619	**20**	45	**20**	75	**20**	100	**20**	75	**20**	76	**20**	101
	L.60.5.1	9	393	**20**	799	**20**	1 225	**20**	1 608	**20**	1 227	**20**	1 242	**20**	1 782
	L.60.8.1	9	393	**20**	617	**20**	942	**20**	1 239	**20**	945	**20**	953	**20**	1 408
Bip. Influence	inf_20_5_1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1
	inf_20_8_1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1	**20**	< 1
	inf_40_5_1	11	698	**20**	527	**20**	343	**20**	341	**20**	348	**20**	346	**20**	341
	inf_40_8_1	11	696	**20**	319	**20**	315	**20**	311	**20**	314	**20**	314	**20**	311
	inf_60_5_1	8	411	**14**	1 319	**14**	1 551	**14**	808	**14**	1 100	**14**	1 049	**14**	1 052
	inf_60_8_1	8	527	14	915	14	751	**15**	1 776	14	750	14	736	14	731
	inf_80_5_1	7	420	11	408	**12**	1 687	**12**	1 115	**12**	1 677	**12**	1 622	10	576
	inf_80_8_1	7	408	**12**	1 259	**12**	966	**12**	641	**12**	966	**12**	938	11	1 345
	inf_100_5_1	6	126	10	399	10	297	**11**	1 204	10	296	**11**	1 662	9	1 776
	inf_100_8_1	6	130	10	415	10	316	**11**	1 221	10	315	**11**	1 637	9	1 521

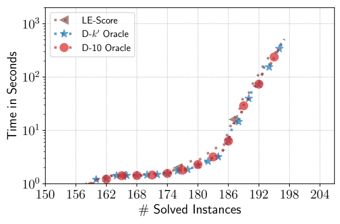
To outline the potential of the pairwise heuristics, we present a new valid pairwise heuristic which returns a sharper bound than the matching heuristic but which is infeasible in practice. Recall that the matching heuristic for a node T outputs the largest sum of the marginal gains of $k'/2$ pairs out of a set A of candidates. Hence, $\max_{A \subseteq C_T, |A|=k'} \max_{P \subseteq P(A), |P|=k'/2} \sum_{p \in P} \Delta(p \mid S_T)$ where $P(A)$ is the set of all pairs of A, is computed. To obtain a sharper bound for a set A we do not want to compute a maximum matching, instead we want to compute $D(T) := \max_{A \subseteq C_T, |A|=k'} \min_{P \subseteq P(A), |P|=k'/2} \sum_{p \in P} \Delta(p \mid S_T)$. Clearly, $D(T)$ is lower than the bound of the matching, however efficiently computing the bound is not possible. In practice, a Dynamic Programming approach takes $\mathcal{O}(n^{k'} {k'}^2)$ time. Recall that we only use this heuristic to outline the potential of the pairwise heuristic, its running time is of no importance here.

▶ **Lemma 5.3** ($\star$). *The upper bound $f(S_T) + D(T)$ is valid.*

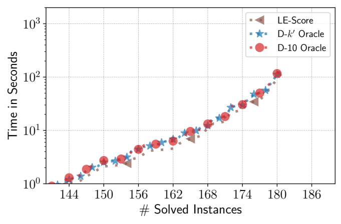
Figure 4 shows the result for $k' = 10$ (`D-10`) and for k' being the remaining budget (`D-`k'). The plots only use instances that were solved by all three algorithms. `D-`k' and `D-10` have almost the same running time as `LE-Score`, showing that there is no hope that any implementation of the pairwise heuristic can improve `LE-Score`. Thus, `SubModST` is `LE-Score`.



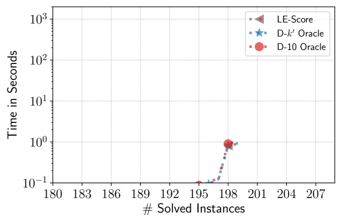
(a) Group Closeness Centr.



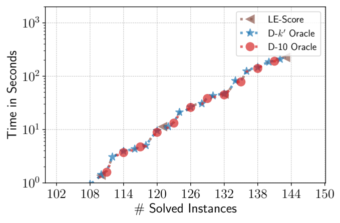
(b) Partial Dominating Set



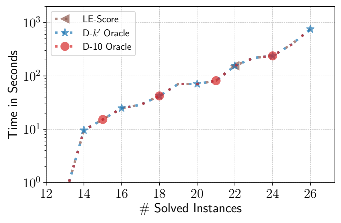
(c) Facility Location



(d) Weighted Coverage



(e) Bipartite Influence



(f) Euclidean Clustering

Figure 4 Number of solved instances of the `LE-Score` algorithm and the two oracle configurations.

Table 3 Running times in seconds of `Simple+` and `LE-Score` against the SOTA algorithms `ICG` [31], and `ICG`$(k-1)$, `GCG` and `ECG` [10] for Facility Location instances with $k = 8$. ⏱ denotes a timeout after 7200 seconds. The smallest running times are marked bold.

Instance	`ICG`	`ICG`$(k-1)$	`GCG`	`ECG`	`Simple+`	`LE-Score`
L.20.8.1	0.34	0.31	0.26	0.78	**0.00**	**0.00**
L.30.8.1	35.93	14.65	10.04	17.15	**0.01**	**0.01**
L.40.8.1	1 502.32	563.23	323.94	568.86	**0.04**	**0.04**
L.50.8.1	⏱	⏱	6 357.27	6 186.57	0.12	**0.11**
L.60.8.1	⏱	⏱	⏱	⏱	0.31	**0.26**

5.2 Comparison to SOTA Algorithms

We compare `SubModST` against the state-of-the-art solver for the generic Submodular Cardinality-Constrained Maximization [10] and against the SOTA solver [28] for Group Closeness Centrality.

Csókás et al. [10] introduced three solvers which are all based on the Improved Constraint Generation (ICG) solver [31]. Unfortunately, we did not get their solver in time. Thus, we here report the results provided in their work [10]. The software is written in AMPL and they used CPLEX 20.1.0.0. They evaluated their experiments on a system with an Intel® Core™ i5-6500 at 3.20 Ghz and 64GB RAM which was running Ubuntu Linux 22.04. We are therefore at a disadvantage in terms of clock speed. The results for Facility Location are shown in Table 3. Our algorithm `SubModST` solves all instances in less than a second and thus `SubModST` is orders of magnitudes faster than the SOTA solvers. For Weighted Coverage and Bipartite Influence the results are roughly similar and can be found in the supplementary material.

In Table 4 we compare `SubModST` against exact solvers of Staus et al. [28] for Group Closeness Centrality. `CI` is also a branch-and-bound algorithm that uses SUB and Candidate Reduction, making it comparable to `Simple+` and `LE-Score`. `ILPnew` is an

Table 4 Comparison of our solvers `Simple+` and `LE-Score` algorithm against `CI` and `ILPnew` [28] for Group Closeness Centrality. For each instance and algorithm the largest solved k and the required time to solve this k in seconds is shown. Numbers marked in bold indicate that this algorithm was able to solve the largest k out of any algorithm.

Instance	`Simple+`		`LE-Score`		`CI`		`ILPnew`	
	k	seconds	k	seconds	k	seconds	k	seconds
ca-netscience	14	1 104	14	968	12	1 663	**20**	< 1
soc-wiki-Vote	13	1 521	13	752	11	1 778	**20**	2
bio-yeast	11	1 080	11	458	9	513	**20**	19
econ-orani678	14	1 048	17	1 724	11	1 362	**20**	13
soc-advogato	12	1 470	14	1 750	8	726	**20**	101
bio-dmela	8	1 036	**9**	700	6	419	1	⏱
ia-escorts-dynamic	7	698	**8**	675	6	742	1	⏱
soc-anybeat	7	1 018	10	1 769	6	1 618	**20**	1 314
ca-AstroPh	5	816	**6**	994	4	1 595	1	⏱
fb-pages-media	**1**	1	**1**	1	**1**	< 1	1	⏱

improved version of the ILP algorithm by Bergamini et al. [4]. Both `Simple+` and `LE-Score` are considerably faster than `CI`. In general, `ILPnew` is much faster than `LE-Score`, but several instances were solved by `LE-Score` and not by `ILPnew`.

6 Discussion

We presented a new solver `SubModST` for Submodular Cardinality-Constrained Maximization which is orders of magnitudes faster than the SOTA algorithms [10, 31]. While being overall slower than a solver for the special case of Group Closeness Centrality [28], `SubModST` solves some instances not solved by Staus et al. [28]. Our hyperparameters are not optimized. If one uses our implementation for a specific problem, one should use hyperparameter optimization tools as SMAC [19] to optimize them.

Recall that we assumed that f is monotone. This property can be scrapped if one evaluates f in each SE-Tree node and only removes candidates with negative marginal gain. Another direction is to consider additional properties of the submodular function f, for example their *total curvature*. This number reflects how much the marginal gains of elements can decrease. For example, for some curvature values, improved approximation algorithms can be given [32]. Can they be used profitably in `SubModST`? Here, we considered cardinality constraints but it seems worthwhile to extend `SubModST` to other constraints such as Knapsack constraints, where each element has a weight and there is a limit on the maximum weight.

References

1 Alexander A. Ageev and Maxim Sviridenko. Approximation Algorithms for Maximum Coverage and Max Cut with Given Sizes of Parts. In *Integer Programming and Combinatorial Optimization, 7th International IPCO Conference, Graz, Austria, June 9-11, 1999, Proceedings*, volume 1610 of *Lecture Notes in Computer Science*, pages 17–30. Springer, 1999. `doi:10.1007/3-540-48777-8_2`.

2 Paola Alimonti and Viggo Kann. Some APX-completeness results for cubic graphs. *Theoretical Computer Science*, 237(1-2):123–134, 2000. `doi:10.1016/S0304-3975(98)00158-3`.

3 Francis R. Bach. Learning with Submodular Functions: A Convex Optimization Perspective. *Foundations and Trends in Machine Learning*, 6(2-3):145–373, 2013. `doi:10.1561/2200000039`.

4 Elisabetta Bergamini, Tanya Gonser, and Henning Meyerhenke. Scaling up group closeness maximization. In Rasmus Pagh and Suresh Venkatasubramanian, editors, *Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments, ALENEX 2018, New Orleans, LA, USA, January 7-8, 2018*, pages 209–222. SIAM, 2018. `doi:10.1137/1.9781611975055.18`.

5 Chen Chen, Wei Wang, and Xiaoyang Wang. Efficient Maximum Closeness Centrality Group Identification. In *Databases Theory and Applications - 27th Australasian Database Conference, ADC 2016, Sydney, NSW, Australia, September 28-29, 2016, Proceedings*, volume 9877 of *Lecture Notes in Computer Science*, pages 43–55. Springer, 2016. `doi:10.1007/978-3-319-46922-5_4`.

6 Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David W. Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized NP-hard problems. *Information and Computation*, 201(2):216–231, 2005. `doi:10.1016/j.ic.2005.05.001`.

7 Wenlin Chen, Yixin Chen, and Kilian Q. Weinberger. Filtered Search for Submodular Maximization with Controllable Approximation Bounds. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*, volume 38 of *JMLR Workshop and Conference Proceedings*. JMLR.org, 2015. URL: `http://proceedings.mlr.press/v38/chen15c.html`.

8 Fabián A. Chudak and David B. Shmoys. Improved Approximation Algorithms for the Uncapacitated Facility Location Problem. *SIAM Journal on Computing*, 33(1):1–25, 2003. `doi:10.1137/S0097539703405754`.

9 Gérard Cornuéjols, George Nemhauser, and Laurence Wolsey. The Uncapicitated Facility Location Problem. Technical report, Cornell University Operations Research and Industrial Engineering, 1983.

10 Eszter Julianna Csókás and Tamás Vinkó. Constraint generation approaches for submodular function maximization leveraging graph properties. *Journal of Global Optimization*, 88(2):377–394, 2024. `doi:10.1007/s10898-023-01318-4`.

11 M. G. Everett and S. P. Borgatti. The Centrality of Groups and Classes. *The Journal of Mathematical Sociology*, 23(3):181–201, 1999.

12 Pasi Fränti and Sami Sieranoja. k-means properties on six clustering benchmark datasets. *Applied Intelligence*, 48(12):4743–4759, 2018. `doi:10.1007/s10489-018-1238-7`.

13 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. `doi:10.1006/jcss.2001.1774`.

14 Richard M. Karp. Reducibility among Combinatorial Problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. `doi:10.1007/978-1-4684-2001-2_9`.

15 Leonard Kaufmann and Peter Rousseeuw. Clustering by Means of Medoids. *Data Analysis based on the L1-Norm and Related Methods*, pages 405–416, 1987.

16 Yoshinobu Kawahara, Kiyohito Nagano, Koji Tsuda, and Jeff A. Bilmes. Submodularity Cuts and Applications. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada*, pages 916–924. Curran Associates, Inc., 2009. URL: `https://proceedings.neurips.cc/paper/2009/hash/9ad6aaed513b73148b7d49f70afcfb32-Abstract.html`.

17 Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999. `doi:10.1016/S0020-0190(99)00031-9`.

18 Andreas Krause and Daniel Golovin. Submodular Function Maximization. In *Tractability: Practical Approaches to Hard Problems*, pages 71–104. Cambridge University Press, 2014. `doi:10.1017/CBO9781139177801.004`.

19 Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *Journal of Machine Learning Research*, 23:54:1–54:9, 2022. URL: `http://jmlr.org/papers/v23/21-0888.html`.

20 Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques II*, pages 234–243. Springer Berlin, Heidelberg, 1977.

21 Thomas Moscibroda and Roger Wattenhofer. Maximizing the Lifetime of Dominating Sets. In *19th International Parallel and Distributed Processing Symposium (IPDPS 2005), CD-ROM / Abstracts Proceedings, 4-8 April 2005, Denver, CO, USA*. IEEE Computer Society, 2005. `doi:10.1109/IPDPS.2005.276`.

22 George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Mathematical Programming*, 14(1):265–294, 1978. `doi:10.1007/BF01588971`.

23 G.L. Nemhauser and L.A. Wolsey. Maximizing Submodular Set Functions: Formulations and Analysis of Algorithms. In *Annals of Discrete Mathematics (11)*, volume 59 of *North-Holland Mathematics Studies*, pages 279–301. North-Holland, 1981.

24 Mohammad Rezaei and Pasi Fränti. Can the Number of Clusters Be Determined by External Indices? *IEEE Access*, 8:89239–89257, 2020. `doi:10.1109/ACCESS.2020.2993295`.

25 Ryan A. Rossi and Nesreen K. Ahmed. The Network Data Repository with Interactive Graph Analytics and Visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 4292–4293. AAAI Press, 2015. `doi:10.1609/aaai.v29i1.9277`.

26 Ron Rymon. Search through Systematic Set Enumeration. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, USA, October 25-29, 1992*, pages 539–550. Morgan Kaufmann, 1992.

27 Chao Shen and Tao Li. Multi-Document Summarization via the Minimum Dominating Set. In *COLING 2010, 23rd International Conference on Computational Linguistics, Proceedings of the Conference, 23-27 August 2010, Beijing, China*, pages 984–992. Tsinghua University Press, 2010. URL: `https://aclanthology.org/C10-1111/`.

28 Luca Pascal Staus, Christian Komusiewicz, Nils Morawietz, and Frank Sommer. Exact Algorithms for Group Closeness Centrality. In *SIAM Conference on Applied and Computational Discrete Algorithms, ACDA 2023, Seattle, WA, USA, May 31 - June 2, 2023*, pages 1–12. SIAM, 2023. `doi:10.1137/1.9781611977714.1`.

29 Ivan Stojmenovic, Mahtab Seddigh, and Jovisa D. Zunic. Dominating Sets and Neighbor Elimination-Based Broadcasting Algorithms in Wireless Networks. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):14–25, 2002. `doi:10.1109/71.980024`.

30 Sergios Theodoridis and Konstantinos Koutroumbas. Pattern Recognition. *IEEE Transactions on Neural Networks*, 19(2):376, 2008. `doi:10.1109/TNN.2008.929642`.

31 Naoya Uematsu, Shunji Umetani, and Yoshinobu Kawahara. An efficient branch-and-cut algorithm for submodular function maximization. *Journal of the Operations Research Society of Japan*, 63(2):41–59, 2020. `doi:10.48550/arXiv.1904.12682`.

32 Jan Vondrak. Submodularity and curvature: the optimal algorithm. *RIMS Kôkyûroku Bessatsu*, pages 253–266, 2010.

33 Henning M. Woydt. Algorithm engineering for generic subset optimization problems. Master's thesis, Friedrich-Schiller-Universität Jena, 2023. `doi:10.22032/dbt.61730`.

34 Jie Wu and Hailan Li. A Dominating-Set-Based Routing Scheme in Ad Hoc Wireless Networks. *Telecommunication Systems*, 18(1-3):13–36, 2001. `doi:10.1023/A:1016783217662`.

35 Yi-Zhi Xu and Hai-Jun Zhou. Generalized minimum dominating set and application in automatic text summarization. *Computing Research Repository*, abs/1602.04930, 2016. `doi:10.48550/arXiv.1602.04930`.