Reoptimization of NP-Hard Problems

Anna Zych-Pawlewicz(✉)

University of Warsaw, Warsaw, Poland
anka@mimuw.edu.pl

Abstract. A reoptimization problem, given two similar instances of an optimization problem and a good solution to the first instance, asks for a solution to the second. In this paper we propose general approximation algorithms applicable to a wide class of reoptimization problems.

1 Introduction

In this paper we propose general approximation algorithms applicable to a wide class of reoptimization problems. We can build a reoptimization problem on top of any optimization problem. An input instance to the reoptimization problem is an instance of the underlying optimization problem, a good solution to it, and a local modification to that instance. We ask for the solution to the modified instance.

As an example imagine a train station where the train traffic is regulated via a certain time schedule. The schedule is computed when the station is ready to operate and, provided that no unexpected event occurs, there is no need to compute it again. Unfortunately an unexpected event, such as a delayed train, is in a long run inevitable. Reoptimization addresses this scenario, asking whether knowing a solution for a certain problem instance is beneficial when computing a solution for a similar instance. When dealing with relatively stable environments, i.e., where changing conditions alter the environment only for a short period of time, it seems reasonable to spend even a tremendous amount of time on computing a good solution for the undisturbed instance, and profit from it when confronted with a temporal change.

Due to its practical impact, a lot of work has been dedicated to reoptimization. The term reoptimization was mentioned for the first time in [16]. The authors studied the problem of scheduling with forbidden sets in the scenarios of adding or removing a forbidden set. Subsequently various other NP-hard problems in reoptimization settings have been successfully approached: the knapsack problem [2], the weighted minimum set cover problem [15], various covering problems [8], the shortest common superstring problem [7], the Steiner tree problem [6,11,13,19,20] and different variations of the traveling salesman problem [1,3,5,9,12,14]. We refer to [10,14,18,21] for the surveys, where [21] is the most

This work was partially conducted as part of a PhD programme at ETH Zürich and later supported by NCN grant UMO-2014/13/B/ST6/01811.

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 477–494, 2018.
https://doi.org/10.1007/978-3-319-98355-4_28

up to date one. Most of the proposed reoptimization algorithms are based on certain general problem patterns. Hence, an attempt to abstract the patterns from the specific problems and state them on a general level appears desirable. One such generalization was presented in [14], where the class of hereditary reoptimization problems under the modification of vertex insertion was observed to be a subject to a general pattern. This was an inspiration to writing this paper.

Our aim is to abstract the general patterns and properties of the problems that lead to good reoptimization algorithms. For the characterized reoptimization problems we propose general algorithms with provable approximation ratios. Our characteristics are very general and apply to a variety of seemingly unrelated problems. Our algorithms achieve the approximation ratios in [2,8,15] obtained for each problem separately. Moreover, as discussed further in the paper, the recent advances in reoptimization of the minimum Steiner tree problem are heavily based on our general methods, see [18]. In [8], reoptimization variants of some covering problems were considered for which the general approximation ratios proposed in this paper were proven tight. This indicates that our methods are unlikely to be improved on a level as general as presented here.

2 The Basics

In this section we model the concept of reoptimization formally and provide some preliminary results. We observe that, in principle, reoptimization of NP-hard problems is NP-hard. Most of the times approximation becomes easier, i.e., a reoptimization problem admits a better approximation ratio than its optimization counterpart. In principle, if the modification changes the optimal cost only by a constant, the reoptimization problem admits a PTAS. The more interesting situation when the optimal cost can change arbitrarily is addressed in subsequent sections.

We now introduce the notation and definitions used further in the paper.

Definition 1 (Vazirani [17]). *An NP* optimization *(NPO) problem* $\Pi = (\mathcal{D}, \mathcal{R}, cost, goal)$ *consists of*

1. *A set of valid instances* $\mathcal{D}_\Pi$ *recognizable in polynomial time. We will assume that all numbers specified in an input are rationals. The size of an instance* $I \in \mathcal{D}_\Pi$*, denoted by* $|I|$*, is defined as the number of bits needed to write* I *under the assumption that all numbers occurring in the instance are written in binary.*
2. *Each instance* $I \in \mathcal{D}_\Pi$ *has a set of feasible solutions,* $\mathcal{R}_\Pi(I)$*. We require that* $\mathcal{R}_\Pi(I) \neq \emptyset$*, and that every solution* $\textsc{Sol} \in \mathcal{R}_\Pi(I)$ *is of length polynomially bounded in* $|I|$*. Furthermore, there is a polynomial-time algorithm that, given a pair* $(I, \textsc{Sol})$*, decides whether* $\textsc{Sol} \in \mathcal{R}_\Pi(I)$*.*
3. *There is a polynomial-time computable objective function,* $cost_\Pi$*, that assigns a nonnegative rational number to each pair* $(I, \textsc{Sol})$*, where* I *is an instance and* Sol *is a feasible solution to* I*.*

4. *Finally, Π is specified to be either a minimization problem or a maximization problem: $goal_\Pi \in \{\min, \max\}$.*

An optimal solution to a problem Π is a feasible solution that minimizes or maximizes the cost, depending on $goal_\Pi$. We denote the set of optimal solutions to an instance I by $\textsc{Optima}_\Pi(I)$ and the optimal cost by $\text{Optcost}_\Pi(I)$. Typically, we write $\textsc{Opt} \in \textsc{Optima}_\Pi(I)$ to denote an optimal solution to an instance I of a problem Π. To denote any solution, we write $\textsc{Sol} \in \mathcal{R}_\Pi(I)$. The costs are referred to as Opt and Sol, respectively. We omit the index and/or the argument if it is clear from the context.

We view an algorithm *Alg* solving an *NPO* problem $\Pi = (\mathcal{D}, \mathcal{R}, cost, goal)$ as a mapping from the instance set to the solution set satisfying $Alg(I) \in \mathcal{R}(I)$. We denote the asymptotic running time of an algorithm *Alg* by $\text{Time}(Alg)$, whereas $\text{Poly}(n)$ stands for a function polynomial in n. For the sake of simplicity, we define an order $\succeq$ on the values of *cost* which favors better solutions: $\succeq$ equals $\leq$ if $goal = \min$, and $\geq$ otherwise. An algorithm *Alg* is exact if, for any instance $I \in \mathcal{D}$,

$$cost(I, Alg(I)) = \max_{\succeq}\{cost(I, \textsc{Sol}) \mid \textsc{Sol} \in \mathcal{R}(I)\}.$$

Alg is a σ-approximation, if for any instance $I \in \mathcal{D}$,

$$cost(I, Alg(I)) \succeq \sigma \max_{\succeq}\{cost(I, \textsc{Sol}) \mid \textsc{Sol} \in \mathcal{R}(I)\},$$

where $\sigma \leq 1$ if $goal = \max$, and $\sigma \geq 1$ otherwise.

We model the scenario where an instance and a corresponding solution are known, and one needs to find a solution after the instance is modified. Hence, we introduce a binary modification relation on the set of instances, which determines which modifications are allowed. Additionally, the parameter ρ measures the quality of the input solution. Formally, a reoptimization problem is defined as follows.

Definition 2. *Let $\Pi = (\mathcal{D}_\Pi, \mathcal{R}_\Pi, cost, goal)$ be an NPO problem and $\mathcal{M} \subseteq \mathcal{D}_\Pi \times \mathcal{D}_\Pi$ be a binary relation (the modification). The corresponding reoptimization problem $\mathfrak{Re}^\rho_{\mathcal{M}}(\Pi) = (\mathcal{D}_{\mathfrak{Re}^\rho_{\mathcal{M}}(\Pi)}, \mathcal{R}_{\mathfrak{Re}^\rho_{\mathcal{M}}(\Pi)}, cost, goal)$ consists of:*

- *a set of feasible instances: $\mathcal{D}_{\mathfrak{Re}^\rho_{\mathcal{M}}(\Pi)} = \{(I, I', \textsc{Sol}) \mid (I, I') \in \mathcal{M}, \textsc{Sol} \in \mathcal{R}_\Pi(I)$ and $\text{Sol} \succeq \rho\text{Optcost}(I)\}$; we refer to I as the original instance and to I' as the modified instance*
- *a feasibility relation: $\mathcal{R}_{\mathfrak{Re}^\rho_{\mathcal{M}}(\Pi)}((I, I', \textsc{Sol})) = \mathcal{R}_\Pi(I')$.*

For the sake of simplicity, we denote $\mathfrak{Re}^1_{\mathcal{M}}(\Pi)$ by $\mathfrak{Re}_{\mathcal{M}}(\Pi)$.

We illustrate this concept on the following example.

Example 1. Consider the weighted maximum satisfiability problem (*wMaxSAT*), where clauses are assigned weights and one seeks for assignment that satisfies a set of clauses of maximum weight. In the reoptimization scenario, a ρ-approximate assignment Sol to formula Φ is given. The given formula Φ is

altered for instance by adding one of its literals to one of its clauses. This is captured by a modification relation: the original instance I and the modified instance I' are a valid part of the input if and only if $(I, I') \in \mathcal{M}$. The modification is defined as follows: $(I, I') \in \mathcal{M}$ if and only if all the clauses but one are the same in Φ and Φ' and the only different clause in Φ' contains one additional literal. The reoptimization problem $\mathfrak{Re}^{\rho}_{\mathcal{M}}(wMaxSAT)$, given $(\Phi, \Phi', \textsc{Sol})$ such that $(\Phi, \Phi') \in \mathcal{M}$, asks for an assignment $\textsc{Sol}'$ for Φ' maximizing the total weight of satisfied clauses. Let us denote the set of satisfied clauses by $Sats(\Phi', \textsc{Sol}')$.

The two consecutive lemmas show both the limits and the power of reoptimization.

Lemma 1 (Böckenhauer et al. [10]). *Reoptimization problems of NP-hard problems, where applying the modification a polynomial number of times can arbitrarily change an input instance, are NP-hard, even if all input instances contain an optimal solution.*

Lemma 2 (Bilò et al. [8]). *If, for every instance $(I, I', \textsc{Sol})$ of $\mathfrak{Re}^{\rho}_{\mathcal{M}}(\Pi)$ and any constants b and b', we can compute in polynomial time a feasible solution $\textsc{Sol}'$ to I', such that $|\mathrm{Sol}' - \mathrm{Optcost}_{\Pi}(I')| \leq b$, and a best feasible solution among the solutions to I' whose cost is bounded by b', then $\mathfrak{Re}^{\rho}_{\mathcal{M}}(\Pi)$ admits a* PTAS.

Proof. Let Π be a maximization problem (for minimization problems the proof is analogous). For a given $\varepsilon > 0$, we compute two solutions to I': $\textsc{Sol}'$ as in the claim of the lemma, and $\textsc{Sol}''$, a best among the feasible solutions with cost bounded by $\frac{b}{\varepsilon}$. If $\mathrm{Optcost}_{\Pi}(I') \leq \frac{b}{\varepsilon}$, then $\textsc{Sol}''$ is optimal. Otherwise $\mathrm{Sol}' \geq \mathrm{Optcost}_{\Pi}(I') - b \geq (1 - \varepsilon)\mathrm{Optcost}_{\Pi}(I')$. □

Example 2. We illustrate Lemma 2 on the example of the maximum satisfiability problem under clause addition: $\mathfrak{Re}_{\mathcal{M}}(MaxSAT)$, where $(\Phi, \Phi') \in \mathcal{M}$ if and only if $\Phi' = \Phi \wedge C_{\text{new}}$ for $C_{\text{new}} \notin \Phi$. Due to Lemma 2, $\mathfrak{Re}_{\mathcal{M}}(MaxSAT)$ admits a PTAS. Clearly, if C_{new} contains new variables, the reoptimization is trivial, so we focus on the case when C_{new} contains only variables from $Var(\Phi)$. Optimal assignments: $\textsc{Opt}$ to Φ and $\textsc{Opt}'$ to Φ' differ at most by one (the contribution of C_{new}). Hence, $\textsc{Opt}$ as a solution to Φ' has a cost greater or equal to $\mathrm{Opt}' - 1$. To satisfy the second condition of the lemma, we need to compute a best among the solutions with cost bounded by a constant b in polynomial time. To that end, we exhaustively search for at most b to be satisfied clauses. For each choice of $b' \leq b$ clauses we verify if the selected set of clauses is satisfiable. Note that b' variables suffices to satisfy b' clauses, so the satisfying assignment, if it exists, can be found in polynomial time via exhaustive search. The PTAS for $\mathfrak{Re}_{\mathcal{M}}(MaxSAT)$ follows by Lemma 2.

Lemma 2 typically applies to unweighted reoptimization problems. It implies a PTAS for *MaxSAT* (maximum satisfiability) if the modification alters only a constant number of clauses, a PTAS for maximum independent set (*MaxIndSet*), maximum clique (*MaxCli*), minimum vertex cover (*MinVerCov*) and minimum dominating set (*MinDomSet*) if the modification alters a constant number of

vertices/edges in the graph [8], a PTAS for minimum set cover (*MinSetCov*) if the modification alters a constant number of sets [8], and a PTAS for β-minimum Steiner tree (β-*SMT*) under changing the weight of an edge and under changing the status of a vertex [13].

3 Self-Reducibility

In the previous section we addressed the reoptimization problems where the modification alters the optimal cost only by a constant. Should that not be the case, other measures need to be developed. As we show in the next section, typically an adjustment of the solution given in the input provides a greedy reoptimization algorithm. To go beyond the approximation ratio of the greedy algorithm, we compute a solution alternative to the greedy one and return a better among them. We propose the self-reduction method for computing the alternative solution and show how it applies to self-reducible problems. We explain the concept of self-reducibility before moving on to the method.

The self-reducibility explains why, for many *NPO* problems, we can construct a recursive algorithm, which reduces an input instance to a few smaller instances and calls itself on each of them. Perhaps the best problem to explain what we have in mind is *MaxSAT*. For a given a formula Φ in n variables, a feasible solution is a partial assignment that assigns to each variable $x_i \in Var(\Phi)$ either 0 or 1 or nothing. The reason why it is convenient for us to allow partial assignments will become clear in Sect. 4. For a partial assignment $\textsc{Sol}$ to Φ, we set $cost(\Phi, \textsc{Sol})$ to be the total number of clauses satisfied by $\textsc{Sol}$. The recursive algorithm takes the first unassigned variable $x_1 \in Var(\Phi)$ and sets $x_1 = 1$. It then reduces Φ to a smaller formula $\Phi_{x_1=1}$, the solutions for which are the partial assignments on the remaining $n-1$ variables. The reduced formula $\Phi_{x_1=1}$ is obtained by removing all the literals x_1 and $\overline{x_1}$ and all the clauses containing x_1 from Φ. The algorithm calls itself recursively on $\Phi_{x_1=1}$. The recursive call returns a partial assignment $\textsc{Sol}_{x_1=1}$ to $\Phi_{x_1=1}$. Then, a solution $\textsc{Sol}$ to Φ is computed by assigning $x_1 = 1$ and the values of $\textsc{Sol}_{x_1=1}$ to the remaining variables. The algorithm analogously computes solution $\textsc{Sol}'$ where $x_1 = 0$. Finally, it returns a better solution among $\textsc{Sol}$ and $\textsc{Sol}'$.

This algorithm finds an optimum for the following reasons. Firstly, it stops after at most $2^{|\Phi|}$ recursive calls, as $|\Phi_{x_1=j}| < |\Phi|$. Secondly, there is a one-to-one correspondence between feasible assignments to Φ setting x_1 to 1 and the feasible assignments to $\Phi_{x_1=1}$. Observe that $cost(\Phi, \textsc{Sol})$ is equal to the number of removed clauses plus $cost(\Phi_{x_1=1}, \textsc{Sol}_{x_1=1})$, so the aforementioned correspondence preserves the order on the assignments induced by their cost. We need that property to find the optimum. In what follows we formalize the above and abstract it from *MaxSAT*.

We assume that the solutions to instances of *NPO* problems have a certain granularity, i.e., they are composed of smaller pieces called atoms. In the above example with *MaxSAT*, an atom is an assignment on a single variable, for example $x_1 = 1$ or $x_1 = 0$. We denote the set of atoms of solutions to I by $Atoms(I)$.

We assume that the size of $Atoms(I)$ is polynomially bounded in the size of I, i.e., $|Atoms(I)| \leq \text{Poly}(|I|)$. The set of atoms of solutions to I is formally defined by $Atoms(I) = \bigcup_{\text{Sol} \in \mathcal{R}(I)} \text{Sol}$. Note that $Atoms(I)$ is entirely determined by the set of feasible solutions for I. For a problem $\Pi = (\mathcal{D}, \mathcal{R}, cost, goal)$, the set of all atoms in all instances of Π is denoted by $Atoms_\Pi = \bigcup_{I \in \mathcal{D}} Atoms(I)$ and we omit the index Π if it is clear from the context. We assume that $cost(I, A)$ is defined for any set of atoms $A \subseteq Atoms(I)$. We extend the function $cost(I, \cdot)$ to any set of atoms in $Atoms_\Pi$ so that the atoms do not contribute unless they are valid atoms to I. More formally, we denote $A \cap Atoms(I)$ by $[A]_I$ and set $cost(I, A) = cost(I, [A]_I)$. We assume that we can extract $[A]_I$ from A in polynomial time, i.e., we can recognize which atoms are the atoms of I for any problem instance I. We impose a special form of the correspondence function, i.e., a solution to the reduced instance corresponds to the union of itself and the set containing the atom used for the reduction. Clearly, the cost should carry over accordingly.

Definition 3. *We will say that a problem Π is* self-reducible *if there is a polynomial-time algorithm, Δ, satisfying the following conditions.*

- *Given an instance I and an atom α of a solution to I, Δ outputs an instance I_α. We require that the size of I_α is smaller than the size of I, i.e., $|I_\alpha| < |I|$. Let $\mathcal{R}(I|\alpha)$ represent the set of feasible solutions to I containing atom α. We require that every solution Sol of I_α, i.e., $\text{Sol} \in \mathcal{R}(I_\alpha)$, has a corresponding solution $\text{Sol} \cup \{\alpha\} \in \mathcal{R}(I|\alpha)$ and that this correspondence is one-to-one.*
- *For any set $A \subseteq \text{Sol}_\alpha \in \mathcal{R}(I_\alpha)$ it holds that $cost(I, A \cup \{\alpha\}) = cost(I, \alpha) + cost(I_\alpha, A)$.*

Definition 3 is an adaptation of the corresponding definition in [17]. We illustrate it on a few examples.

Example 3 (Self-Reducibility of the Weighted Maximum Satisfiability). For a given instance $I = (\Phi, c)$, the set of feasible solutions $\mathcal{R}(I)$ contains truth assignments on the subsets of variables. A feasible truth assignment assigns either 0 or 1 but not both to each variable in some set $X \subseteq Var(\Phi)$. We represent such assignments as a set of pairs in $X \times \{0, 1\}$. For example, an assignment that assigns 1 to x_1 and 0 to x_2 is represented as $\{(x_1, 1), (x_2, 0)\}$. Such a representation determines the set of atoms for an instance (Φ, c) as $Atoms((\Phi, c)) = Var(\Phi) \times \{0, 1\}$. The cost of a subset of atoms $A \subseteq Atoms((\Phi, c))$ is $cost(\Phi, A) = \sum_{C \in Sats(\Phi, A)} c(C)$. The reduction algorithm Δ on atom (x, i), $i \in \{0, 1\}$, returns the instance $(\Phi_{(x,i)}, c)$ obtained by removing all literals x and $\bar{x}$ and all clauses containing x from Φ. Clearly, $Var(\Phi_{(x,i)}) \subseteq Var(\Phi) \setminus \{x\}$, so any assignment $\text{Sol}_{(x,i)}$ to $\Phi_{(x,i)}$ can be extended to an assignment Sol to Φ by setting $\text{Sol} = \text{Sol}_{(x,i)} \cup \{(x, i)\}$. Then, $cost((\Phi, c), \text{Sol}) = c(Sats(\Phi, \{(x, i)\})) + cost((\Phi_{(x,i)}, c), \text{Sol}_{(x,i)})$. It may happen that, if during the reduction step from I to $\Delta(I, (x, i))$ two variables x and y are lost, then the solutions to I assigning 1 or 0 to y cannot be reached from solutions to $\Delta(I, (x, i))$ by adding (x, i). They are, however, equivalent with solutions to I assigning nothing to y, and these

can be reached. There is a way to define Δ to meet the conditions in Definition 3 exactly, for the sake of clarity however we do not describe it here.

The next few examples are built on graph problems. For such we adopt the following notation. The neighborhood of a vertex $v \in V(G)$ is denoted by $\Gamma_G(v)$. Note that if G is a simple graph then $v \notin \Gamma_G(v)$. The degree of a vertex $v \in V(G)$ is defined as $deg_G(v) = |\Gamma_G(v)|$. For a set of vertices $V' \subseteq V(G)$, by $G - V'$ we denote the subgraph of G induced on $V(G) \setminus V'$.

Example 4 (Self-Reducibility of the Weighted Maximum Independent Set Problem). For a given input instance $G = (V, E, c)$ of *wMaxIndSet*, the set $\mathcal{R}(G)$ of feasible solutions contains all sets of vertices that are independent in G: $\text{SOL} \in \mathcal{R}(G) \iff (\text{SOL} \subseteq V(G)$ and SOL is independent in $G)$. The set of atoms for G, according to the definition, is the union of atoms over all the solutions in $\mathcal{R}(G)$. Since any vertex in $V(G)$ is in some independent set, we obtain $Atoms(G) = V(G)$. Given a set of atoms $A \subseteq Atoms(G)$ (not necessarily independent in G), we set $cost(G, A) = \sum_{v \in A} c(v)$. We argue that this representation of *wMaxIndSet* is self-reducible. The reduction algorithm Δ on G and $v \in Atoms(G)$ returns a graph G_v, obtained by removing v and its neighborhood from G: $\Delta(G, v) = G - (\Gamma_G(v) \cup \{v\})$. Clearly, any solution SOL_v to G_v maps to the solution to G given by $\text{SOL} = \text{SOL}_v \cup \{v\}$. Clearly, $cost(G, \text{SOL}) = cost(G_v, \text{SOL}_v) + cost(G, v)$. Hence the conditions of Definition 3 are satisfied.

Example 5 (Self-Reducibility of the Minimum Steiner Tree (SMT) Problem). For a given instance (G, S), the feasible solutions in $\mathcal{R}((G, S))$ are the subgraphs of G spanning S. We represent solutions as sets of edges, i.e., $\text{SOL} \subseteq E(G)$ is a feasible solution to (G, S) if the edges in SOL span S. This determines the set of atoms: $Atoms((G, S)) = E(G)$. The cost function is defined as the sum of costs of edges in the solution: $cost((G, S), \text{SOL}) = \sum_{e \in \text{SOL}} c(e)$. The reduction function $\Delta((G, S), e)$ contracts edge e to a vertex and includes that vertex in the terminal set. Clearly, this might create parallel edges and self-loops. Nevertheless, any solution $\text{SOL}_e \in \mathcal{R}(\Delta((G, S), e))$ spans S and the vertex corresponding to e in the contracted graph. Moreover, all edges of SOL_e are edges in G. As a set of edges in G, SOL_e forms two connected components attached to the endpoints of e, and the component $\text{SOL}_e \cup \{e\}$ spans S in G.

Example 6 (Self-Reducibility of the Maximum Cut with Required Edges Problem). For a given instance (G, R), feasible solutions are sets of edges that extend R to a cut in G. This determines the set of atoms: $Atoms((G, R)) = E(G) \setminus R$. The cost function is, as in the previous example, defined as the sum of costs of edges in the solution: $cost((G, R), \text{SOL}) = \sum_{e \in \text{SOL}} c(e)$. The reduction function $\Delta((G, R), e)$ reduces the cost $c(e)$ to 0 and includes e in R. It is easy to see that the conditions of Definition 3 hold.

The method we now introduce improves the approximation ratio of a self-reducible optimization problem Π if every problem instance admits an optimal

Algorithm 1: $\mathcal{S}_{Alg}$ employing an approximation algorithm Alg

input : Instance I
forall the $\alpha \in Atoms(I)$ **do**
    $\textsc{Sol}(\alpha) := Alg(\Delta(I, \alpha))$;
end
output: The best among $\textsc{Sol}(\alpha) \cup \{\alpha\}$

solution containing an expensive atom. In essence, we exhaustively search for the expensive atom through all atoms. For each atom we reduce the instance, approximate the solution of a reduced instance using an approximation algorithm Alg for Π and add the missing atom (See Algorithm 1 for the corresponding algorithm $\mathcal{S}_{Alg}$). The next lemma and corollary show what we gain by using $\mathcal{S}_{Alg}$ as compared to using Alg only.

Lemma 3. *If $\Pi = (\mathcal{D}, \mathcal{R}, cost, goal)$ is self-reducible and Alg is a σ-approximation algorithm for Π, then, for every $\textsc{Sol} \in \mathcal{R}(I)$ and every atom $\gamma \in \textsc{Sol}$, it holds that $cost(I, \mathcal{S}_{Alg}(I)) \succeq \sigma\, cost(I, \textsc{Sol}) - (\sigma - 1) cost(I, \gamma)$.*

Proof. Fix $\textsc{Sol} \in \mathcal{R}(I)$ and $\gamma \in \textsc{Sol}$. The algorithm returns $\textsc{Sol}(\alpha) \cup \{\alpha\}$ for some atom α which gives the best cost. This solution is feasible for I due to the first condition of Definition 3. Further, since $\textsc{Sol} \in \mathcal{R}(I|\gamma)$, there is $\textsc{Sol}_\gamma$ such that $\textsc{Sol} = \textsc{Sol}_\gamma \cup \{\gamma\}$. The estimation of $cost(I, \mathcal{S}_{Alg}(I))$ follows:

$$
\begin{aligned}
&cost(I, \mathcal{S}_{Alg}(I)) \\
&\quad = cost(I, \textsc{Sol}(\alpha) \cup \{\alpha\}) \succeq cost(I, \textsc{Sol}(\gamma) \cup \{\gamma\}) \\
&\quad = cost(I, \gamma) + cost(I_\gamma, \textsc{Sol}(\gamma)) \succeq cost(I, \gamma) + \sigma\, cost(I_\gamma, \textsc{Sol}_\gamma) \\
&\quad = cost(I, \gamma) + \sigma(cost(I, \textsc{Sol}) - cost(I, \gamma)) \\
&\quad = \sigma\, cost(I, \textsc{Sol}) - (\sigma - 1) cost(I, \gamma)
\end{aligned}
$$

□

Corollary 1. *If every instance I of Π admits an optimal solution containing an atom α with $cost(I, \alpha) \succeq \delta \mathrm{Optcost}(I)$, then $\mathcal{S}_{Alg}$ is a $\sigma - \delta(\sigma - 1)$-approximation.*

In what follows, we generalize the idea by introducing the concept of *guessable* sets of atoms. We say that a set of atoms $A \subseteq \textsc{Sol} \in \mathcal{R}(I)$ is $\mathrm{F}(n)$-guessable for some instance I of a self-reducible problem Π, $|I| = n$, if we can determine a collection of sets of atoms $\mathcal{G}$ with $|\mathcal{G}| \in \mathcal{O}(\mathrm{F}(n))$ such that $A \in \mathcal{G}$. We then call $\mathcal{G}$ a set of guesses for A. Guessing A boils down to an exhaustive search through $\mathcal{G}$. This is useful when we can prove the existence of a certain set A with some characteristic, but we have no knowledge what the elements of A are. If A is $F(n)$-guessable, it is enough to iterate through $\mathcal{O}(F(n))$ candidates in $\mathcal{G}$ to be guaranteed that at some point we look at A. For example, a subset of $Atoms(I)$ of size b of maximum cost among the subsets of size b is n^b-guessable: $\mathcal{G}$ is in that case a collection of subsets of $Atoms(I)$ containing exactly b atoms. Hence,

Algorithm 2: $\mathcal{S}^{\mathcal{G}}_{Alg}$ employing an approximation algorithm Alg

input : Instance I
forall the $A = \{\alpha_1, \ldots, \alpha_{|A|}\} \in \mathcal{G}$ **do**
  $I_0 := I$;
  for $(j = 1, \ldots, |A|)$ **do**
    $I_j := \Delta(I_{j-1}, \alpha_j)$;
  end
  $\widetilde{\text{SOL}}_{|A|} := Alg(I_{|A|})$;
  for $(j = 0, \ldots, |A| - 1)$ **do**
    $\widetilde{\text{SOL}}_{|A|-j-1} := \widetilde{\text{SOL}}_{|A|-j} \cup \{\alpha_{|A|-j}\}$;
  end
end
output: The best $\widetilde{\text{SOL}}_0$ over all considered $A \subseteq Atoms(I)$

it is Poly(n)-guessable if b is constant (recall that we assume that $|Atoms(I)| \leq$ Poly(n)). Another example of a Poly(n)-guessable set applies to problems where the instances are graphs and the solutions are the sets of vertices in the graph. In that case, we let $A \subset Atoms(I)$ be the neighborhood of a vertex that belongs to some optimal solution (any such vertex). We know that such a set A exists if optimum is not empty. The size of the neighborhood may not be constant as in the previous example, however, the neighborhood itself is determined by a single vertex. Setting $\mathcal{G} = \{\Gamma_G(v)\}_{v \in V(G)}$ brings us to the conclusion that A is n-guessable as $|\mathcal{G}| = |V(G)| < n$. Note that every set of atoms (including the optimal solution) is $2^{|Atoms(I)|}$-guessable.

We modify $\mathcal{S}_{Alg}$ to handle guessable sets. Let $\mathcal{G}$ be the set of valid guesses for a set of atoms. The modified algorithm $\mathcal{S}^{\mathcal{G}}_{Alg}$ runs through all sets $A \in \mathcal{G}$ and reduces the instance using the atoms in A one by one. It applies Alg on the obtained instance and combines the solution with the atoms in A. The resulting algorithm $\mathcal{S}^{\mathcal{G}}_{Alg}$ is presented in Algorithm 2.

Algorithm $\mathcal{S}^{\mathcal{G}}_{Alg}$ runs in time $\mathcal{O}(|\mathcal{G}|(|Atoms(I)|\text{Time}(\Delta) + \text{Time}(Alg)))$, so it is a polynomial-time algorithm for $|\mathcal{G}| = \text{Poly}(n)$.

Lemma 4. *If* $\Pi = (\mathcal{D}, \mathcal{R}, cost, goal)$ *is self-reducible and* Alg *is a* σ*-approximation algorithm for* Π *then, for every* $\text{SOL} \in \mathcal{R}(I)$ *and every set* $A \in \mathcal{G}$ *such that* $A \subseteq \text{SOL}$*, it holds that* $cost(I, \mathcal{S}^{\mathcal{G}}_{Alg}(I)) \succeq \sigma\, cost(I, \text{SOL}) - (\sigma - 1)cost(I, A)$.

Proof. Let $A = \{\alpha_1, \ldots, \alpha_{|A|}\}$ and a solution SOL to instance I be as in the claim of the lemma. Due to Definition 3, SOL corresponds to a solution SOL_{α_1} to instance $I_1 = \Delta(I, \alpha_1)$ such that $\text{SOL} = \text{SOL}_{\alpha_1} \cup \{\alpha_1\}$. Thus all atoms of SOL, except of possibly α_1, are contained in $\text{SOL}_{\alpha_1} \in \mathcal{R}(I_{\alpha_1})$. Hence, we can reduce I_1 further using α_2 as a reduction atom. The argument carries on to all the instances constructed by $\mathcal{S}^{\mathcal{G}}_{Alg}$. Again due to Definition 3, $\widetilde{\text{SOL}}_j = \widetilde{\text{SOL}}_{j+1} \cup \{\alpha_{j+1}\}$ is feasible to I_j for $j = 0, \ldots, |A|$, and the following estimation of the cost of the output holds:

$$\begin{aligned} cost(I, \mathcal{S}^{\mathcal{G}}_{Alg}(I)) &\succeq cost(I, \widetilde{\mathrm{SOL}}_0) \succeq cost(I, \widetilde{\mathrm{SOL}}_1 \cup \{\alpha_1\}) \\ &= cost(I, \alpha_1) + cost(I_1, \widetilde{\mathrm{SOL}}_1) \\ &\succeq \cdots \succeq \sum_{j=1}^{|A|} cost(I_{j-1}, \alpha_j) + cost(I_{|A|}, \widetilde{\mathrm{SOL}}_{|A|}) \end{aligned}$$

Now let $\mathrm{SOL}_0 := \mathrm{SOL}$ and SOL_{j+1} be the solution for I_{j+1} which satisfies $\mathrm{SOL}_j = \mathrm{SOL}_{j+1} \cup \{\alpha_{j+1}\}$ for $j := 0, \ldots, |A| - 1$. We continue our estimation of the cost. Based on the second condition of Definition 3 and the fact that $\widetilde{\mathrm{SOL}}_{|A|}$ is a σ-approximation the following holds.

$$\begin{aligned} cost(I, \mathcal{S}^{\mathcal{G}}_{Alg}(I)) &\succeq cost(I, A) + \sigma\, cost(I_{|A|}, \mathrm{SOL}_{|A|}) \\ &= cost(I, A) + \sigma(cost(I_{|A|-1}, \mathrm{SOL}_{|A|-1}) - cost(I_{|A|-1}, \alpha_{|A|})) \\ &= cost(I, A) + \sigma(cost(I, \mathrm{SOL}) - \sum_{j=1}^{|A|} cost(I_{j-1}, \alpha_j)) \\ &= cost(I, A) + \sigma(cost(I, \mathrm{SOL}) - cost(I, A)) \end{aligned}$$

□

Corollary 2. *If every instance I admits an optimal solution containing a set $A \in \mathcal{G}$ with $cost(I, A) \succeq \delta \mathrm{Optcost}(I)$, then $\mathcal{S}^{\mathcal{G}}_{Alg}$ is a $(\sigma - \delta(\sigma - 1))$-approximation.*

4 Modifications

The main problem with the algorithm $\mathcal{S}^{\mathcal{G}}_{Alg}$ introduced in the previous section is that it does not help if the optima do not contain an efficiently guessable expensive set of atoms. Here, the reoptimization comes into play. Knowing the solution to a similar problem instance, we can construct a greedy solution which is an alternative to the solution computed by $\mathcal{S}^{\mathcal{G}}_{Alg}$. Luckily, the bad instances for $\mathcal{S}^{\mathcal{G}}_{Alg}$ are exactly the good instances for the greedy algorithm. For the remainder of this section, let $\Pi = (\mathcal{D}, \mathcal{R}, cost, goal)$ be a self-reducible optimization problem and let $\mathfrak{Re}^{\rho}_{\mathcal{M}}(\Pi)$ be the corresponding reoptimization problem. Also, for the remainder of this section, let Alg be a σ-approximation algorithm for Π.

The greedy algorithm depends on the type of modification. For example, if the input solution SOL happens to be feasible to the modified instance I', we can use it as the greedy solution. We classify in this section three types of modifications by a combination of two classification criteria and provide general approximation algorithms for them. For the so-called *progressing* modifications, feasible solutions to the original instance I remain feasible to the modified instance I'. For example, after removing an edge from a graph, independent sets remain independent. This is not the case when adding an edge. Then, however, an independent set in the modified instance I' is also independent in the original instance I. The latter is how we define *regressing* modifications.

Table 1. Different types of local modifications with the corresponding approximation ratios for $\mathfrak{Re}_{\mathcal{M}}(\Pi)$

Modification		
Progressing		Regressing
Subtractive	Additive	Additive
$\frac{1}{2-\sigma}$	$\frac{2\sigma-1}{\sigma}$	$\frac{2\sigma-1}{\sigma}$

Once we figured out if the modification is progressing or regressing, we can use the solutions to one of the input instances as the solution to the other one. Unfortunately, this does not work in the opposite direction. Our second classification criterion describes how to modify a solution to this other instance in order to make it feasible for the instance it is not feasible for. As a result, progressing and regressing modifications are subdivided into two further types: *subtractive* and *additive*. They strictly correspond to whether the problem is a maximization or a minimization problem. The classification together with the corresponding approximation ratios is shown in Table 1. For our approximation algorithms to work, feasibility preservation must be accompanied by cost preservation. We start with a few definitions that let us preserve the cost of feasible solutions in a convenient way.

Definition 4. *We say that a cost function cost is:*

- pseudo-additive *if the sum of the costs of two sets of atoms is greater than or equal to the cost of the union of the sets. Formally, for any* $A_1, A_2 \subseteq Atoms(I)$ *such that* $A_1 \cap A_2 = \emptyset$ *it holds that* $(cost(I, A_1 \cup A_2) \leq cost(I, A_1) + cost(I, A_2)$ *and, if* $A_1 \subseteq A_2$, *then* $cost(I, A_1) \leq cost(I, A_2))$,
- additive *if the cost of a set of atoms is the sum of the measures of its disjoint subsets. Formally, for any* $A_1, A_2 \subseteq Atoms(I)$ *such that* $A_1 \cap A_2 = \emptyset$ *it holds that* $cost(I, A_1 \cup A_2) = cost(I, A_1) + cost(I, A_2)$.

Definition 5. *A modification* $\mathcal{M}$ *is cost-preserving if, for all* $(I, I') \in \mathcal{M}$ *and all* $\textsc{Sol} \in \mathcal{R}(I)$, *it holds that* $cost(I, \textsc{Sol}) \preceq cost(I', \textsc{Sol})$.

Definition 6. *A modification* $\mathcal{M}$ *is reversely cost-preserving if, for all* $(I, I') \in \mathcal{M}$ *and all* $\textsc{Sol}' \in \mathcal{R}(I')$, *it holds that* $cost(I', \textsc{Sol}') \preceq cost(I, \textsc{Sol}')$.

Definition 7. *A modification* $\mathcal{M}$ *is:*

1. progressing *if feasible solutions to the original instance* I *are feasible also to the modified instance: for every pair* $(I, I') \in \mathcal{M}$ *it holds that* $[\mathcal{R}(I)]_{I'} \subseteq \mathcal{R}(I')$,
2. regressing *if feasible solutions to the modified instance* I' *remain feasible for the original instance* I*: for every* $(I, I') \in \mathcal{M}$ *it holds that* $[\mathcal{R}(I')]_I \subseteq \mathcal{R}(I)$.

Definition 8. *A progressing modification $\mathcal{M}$ is:*

1. subtractive *(in maximization problems) if there is an optimal solution to the modified instance I' which, after removing a part of it, becomes feasible for I and not less valuable:*

$$\begin{array}{c} \exists \textsc{Opt}' \in \textsc{Optima}(I') \\ \exists A' \subseteq \textsc{Opt}' \end{array} \left| \begin{array}{c} \textsc{Opt}' \setminus A' \in \mathcal{R}(I) \\ cost(I, \textsc{Opt}' \setminus A') \geq cost(I', \textsc{Opt}' \setminus A'), \end{array} \right.$$

2. additive *(in minimization problems) if, for every optimal solution* Opt *to the original instance I, there is an optimal solution* Opt′ *to the modified instance I' such that one can be transformed into the other as follows*
   - Opt′ *(minus possibly some atoms) becomes feasible to I after adding to it a part of* Opt
   - *If we remove this part from* Opt*, it may not be feasible anymore. It becomes feasible to I' after adding a part of* Opt′*. Formally:*

$$\begin{array}{c} \forall \textsc{Opt} \in \textsc{Optima}(I) \\ \exists \textsc{Opt}' \in \textsc{Optima}(I') \\ \exists A \subseteq \textsc{Opt} \\ \exists A', A'' \subseteq \textsc{Opt}' \end{array} \left| \begin{array}{c} \textsc{Opt}' \setminus A'' \cup A \in \mathcal{R}(I) \\ cost(I, \textsc{Opt}' \setminus A'' \cup A) \leq cost(I', \textsc{Opt}' \setminus A'' \cup A) \\ (\textsc{Opt} \setminus A) \cup A' \in \mathcal{R}(I'). \end{array} \right.$$

We now introduce the general approximation algorithms.

Theorem 1. *Let cost be pseudo-additive, $\mathcal{M}$ be cost preserving and progressing subtractive and let A' be as in Definition 8.1. Then, there is a $\frac{1}{2-\sigma}$-approximation algorithm for $\mathfrak{Re}_{\mathcal{M}}(\Pi)$ that runs in time $\mathcal{O}(\mathrm{F}(n)(\mathrm{Time}(Alg) + \mathrm{Poly}(n)))$ if A' is $\mathrm{F}(n)$-guessable.*

Proof. Let $(I, I', \textsc{Opt})$ be an input instance in $\mathcal{D}_{\mathfrak{Re}^1_{\mathcal{M}}(\Pi)}$. The approximation algorithm for $\mathfrak{Re}^1_{\mathcal{M}}(\Pi)$ returns the better one of Opt and $\mathcal{S}^{\mathcal{G}}_{Alg}(I')$. Since $\mathcal{M}$ is progressing, $\textsc{Opt} \in \mathcal{R}(I')$. Let Opt′ be an optimal solution for I'. Then

$$\begin{aligned} cost(I', \textsc{Opt}) &\underset{\text{cost pres.}}{\geq} cost(I, \textsc{Opt}) && (1) \\ &\underset{\text{prog. subt., optimality}}{\geq} cost(I, \textsc{Opt}' \setminus A') && (2) \\ &\underset{\text{prog. subt.}}{\geq} cost(I', \textsc{Opt}' \setminus A') && (3) \\ &\underset{\text{ps. add.}}{\geq} cost(I', \textsc{Opt}') - cost(I', A'). && (4) \end{aligned}$$

By Lemma 4, $cost(I', \mathcal{S}^{\mathcal{G}}_{Alg}(I')) \geq \sigma\, cost(I', \textsc{Opt}') - (\sigma - 1) cost(I', A')$. Simple calculations show that at least one of the two provided solutions gives an approximation ratio of $\frac{1}{2-\sigma}$. □

Note that among (1) to (4), only (4) is crucial for the theorem to hold. Hence,

Remark 1. If we are able by some means to obtain a solution with a cost bounded from below as (4) dictates, then the claim of Theorem 1 holds.

We next show how to generalize Theorem 1 to the case where we can guess a larger part of $\textsc{Opt}'$ than the part that we need for estimating the cost of $\textsc{Opt}$.

Definition 9. *A progressing modification is α-subtractive, $\alpha \leq 1$, if*

$$\begin{array}{r|c} \exists \textsc{Opt}' \in \textsc{Optima}(I') & \textsc{Opt}' \setminus A'' \in \mathcal{R}(I) \\ \exists A' \subseteq \textsc{Opt}' & cost(I, \textsc{Opt}' \setminus A'') \geq cost(I', \textsc{Opt}' \setminus A'') \\ \exists A'' \subseteq A' & cost(I', A'') \leq \alpha\, cost(I', A') \end{array}$$

Note that a 1-progressing subtractive modification is simply the progressive subtractive modification from Definition 8.1.

Corollary 3. *Let cost be pseudo-additive, $\mathcal{M}$ be cost-preserving and α-progressing subtractive and let A' be as in Definition 9. Then, there is a $\frac{1-\sigma(1-\alpha)}{1-\sigma+\alpha}$-approximation algorithm for $\mathfrak{Re}_{\mathcal{M}}(\Pi)$ that runs in time $\mathcal{O}(\mathrm{F}(n)(\mathrm{Time}(Alg) + \mathrm{Poly}(n)))$ if A' is $\mathrm{F}(n)$-guessable.*

Proof. Analogous to the proof of Theorem 1. In (2) we substitute A' with A''. Then (4) gives $cost(I', \textsc{Opt}) \geq cost(I', \textsc{Opt}') - cost(I', A'')$. Using the fact that $cost(I', A'') \leq \alpha\, cost(I', A')$, we obtain a lower bound on $cost(I', \textsc{Opt})$ given by $cost(I', \textsc{Opt}) \geq cost(I', \textsc{Opt}') - \alpha\, cost(I', A')$. The alternative solution, $\mathcal{S}^{\mathcal{G}}_{Alg}(I')$, by Lemma 3 satisfies $cost(I', \mathcal{S}^{\mathcal{G}}_{Alg}(I')) \geq \sigma\, cost(I', \textsc{Opt}') - (\sigma - 1) cost(I', A')$. Again simple calculations lead to the approximation ratio as claimed. □

Corollary 4. *Let the assumptions be as in Corollary 3. Then, there is a $\rho\frac{1-\sigma(1-\alpha)}{1-\sigma+\alpha\rho}$-approximation algorithm for $\mathfrak{Re}^{\rho}_{\mathcal{M}}(\Pi)$ that runs in time $\mathcal{O}(\mathrm{F}(n)(\mathrm{Time}(Alg) + \mathrm{Poly}(n)))$ if A' is $\mathrm{F}(n)$-guessable.*

Proof. We plug in a multiplicative factor of ρ in (2). The rest of the proof is analogous to the proof of Corollary 3. □

The use of Corollary 4 is illustrated by the following example.

Example 7. Consider *wMax-k-SAT* under clause addition: $\mathfrak{Re}^{\rho}_{\mathcal{M}}(wMax\text{-}k\text{-}SAT)$ where $(\Phi, \Phi') \in \mathcal{M}$ if and only if $\Phi' = \Phi \wedge C_{\text{new}}$ for $C_{\text{new}} \notin \mathcal{C}(\Phi)$. Recall that solutions to Φ are partial assignments and $cost(\Phi, \textsc{Sol}) = \sum_{C \in Sats(\textsc{Sol})} c(C)$. Observe that the cost function is pseudo-additive (as in Definition 4). Since $Var(\Phi) \subseteq Var(\Phi')$, a feasible assignment to Φ is feasible to Φ', and its cost does not decrease, hence $\mathcal{M}$ is progressive and cost-preserving (as in Definitions 5 and 7). To prove that it is also additive, let A' be the set of atoms in $\textsc{Opt}'$ that satisfy C_{new}, i.e., the atoms of form $(x, 1)$ for $x \in C_{\text{new}}$ and $(x, 0)$ for $\overline{x} \in C_{\text{new}}$. Clearly, $|A'| \leq k$, so $|A'|$ is n^k-guessable if n is the size of the instance. The conditions of Definition 8.1 are satisfied, as $\textsc{Opt}' \setminus A'$ is feasible to Φ and its cost is the same with respect to both Φ' and Φ. Thus, Corollary 4 applies with $\alpha = 1$ and we obtain a $\frac{\rho}{1-\sigma+\rho}$-approximation algorithm that runs in polynomial time if k is constant. For instance, for $\mathfrak{Re}_{\mathcal{M}}(wMax\text{-}3\text{-}SAT)$ this gives an approximation ratio of 0.83, whereas *wMax-3-SAT* admits an approximation ratio of 0.8. As a matter of fact, following Remark 1 we can apply our

Table 2. Some reoptimization problems where Corollary 4 applies. In the remarks column we state if we meet the earlier approximation ratio for the same problem. Corollary 4 gives polynomial-time algorithms even if a constant number of items is added, removed, or change their cost.

Problem	Modification	Remarks
wMaxIndSet	Edge removal	Corollary 4 applies with $\alpha = 0.5$ for edge removal; implies ratio from [8]
	Vertex addition	
wMaxCli	Edge addition	Analogous to *wMaxIndSet*; implies ratio from [8]
	Vertex addition	
wMax-k-SAT, const. k	Clause addition	See Example 7
wMaxSAT	Addition of a variable to arbitrary number of clauses	
Knapsack	Item addition	Implies ratio from [2]
ReqMaxCut	Edge cost increase	See Example 6 for self-reducibility; implies approximation algorithm for *MaxCut* under edge cost increase with the same ratio

algorithm to the *wMaxSAT* problem under addition of a clause as well. It suffices to observe that, for any atom $x_i = j \in \textsc{Opt}'$ satisfying C_{new}, it holds that $cost(I', \textsc{Opt}) \geq cost(I', \textsc{Opt}') - c(C_{\text{new}}) = cost(I', \textsc{Opt}') - cost(I', (x_i, j))$.

Table 2 shows the power of Corollary 4, i.e., we list there the problems, where Corollary 4 directly applies.

Theorem 2. *Let cost be additive, $\mathcal{M}$ be cost preserving and progressing additive and A, A' be as in Definition 8.2. Then there is a $\frac{2\sigma-1}{\sigma}$-approximation algorithm for $\mathfrak{Re}_{\mathcal{M}}(\Pi)$ that runs in time $\mathcal{O}(\mathrm{F}(n)(\mathrm{Time}(Alg) + \mathrm{Poly}(n)))$ if A and A' are $\mathrm{F}(n)$-guessable.*

Proof. Let $\textsc{Opt}$ and $\textsc{Opt}'$ be the optima of I and I'. The algorithm computes two solutions $(\textsc{Opt} \setminus A) \cup A'$ and $\mathcal{S}^{\mathcal{G}}_{Alg}(I')$, and returns the better one. Let us first estimate the cost of $(\textsc{Opt} \setminus A) \cup A'$:

$$
\begin{aligned}
& cost(I', (\textsc{Opt} \setminus A) \cup A') \\
& \quad \underset{\text{ps. add.}}{\leq} \quad cost(I', \textsc{Opt} \setminus A) + cost(I', A') \\
& \quad \underset{\text{additivity}}{\leq} \quad cost(I', \textsc{Opt}) - cost(I', A) + cost(I', A') \\
& \quad \underset{\text{cost pres.}}{\leq} \quad cost(I, \textsc{Opt}) - cost(I', A) + cost(I', A') \\
& \quad \underset{\text{optimality}}{\leq} \quad cost(I, (\textsc{Opt}' \setminus A'') \cup A) - cost(I', A) + cost(I', A') \\
& \quad \underset{\text{additivity}}{\leq} \quad cost(I', (\textsc{Opt}' \setminus A'') \cup A) - cost(I', A) + cost(I', A')
\end{aligned}
$$

$$\underset{\text{ps. add.}}{\leq} \quad cost(I', \textsc{Opt}') + cost(I', A').$$

By Lemma 4, $cost(I', \mathcal{S}^{\mathcal{G}}_{Alg}(I')) \leq \sigma\, cost(I', \textsc{Opt}') - (\sigma - 1)cost(I', A')$. Putting the inequalities together gives the desired result. □

Theorem 2 does not generalize to the case where an approximate solution is given as a part of the input, see [8] for the proof of this fact. A straightforward example of a problem where Theorem 2 applies is the weighted minimum vertex cover problem under edge removal.

Example 8. We illustrate Theorem 2 on the example of the weighted minimum vertex cover problem under edge removal, i.e., $\mathfrak{Re}_{\mathcal{M}}(wMinVerCov)$, where $(G, G') \in \mathcal{M}$ if and only if $V(G) = V(G')$ and $E(G') = E(G) \setminus \{e\}$ for some $e \in E(G)$. For an instance G of *wMinVerCov*, a feasible solution is a set of vertices $\textsc{Sol} \subseteq V$ covering all the edges in G. This defines the set of atoms of G as $Atoms(G) = V(G)$. The cost function $cost(G, \textsc{Sol}) = \sum_{v \in \textsc{Sol}} c(v)$ is clearly additive. The self-reducibility manifests itself with the reduction function Δ which removes a vertex from the graph together with the adjacent (covered) edges: $\Delta(G, v) = G - v$. A vertex cover remains feasible when an edge is removed from the graph, hence $\mathcal{M}$ is progressing. The cost of any set of vertices is the same for G and G', hence $\mathcal{M}$ is cost-preserving. Let $\textsc{Opt}$ be an optimal solution for the original graph and $\textsc{Opt}'$ for the modified graph. We expose the sets A, A' and $A'' = \emptyset$ satisfying Definition 8.2. Adding any endpoint of the removed edge e to $\textsc{Opt}'$ makes it feasible for the original instance, as it covers e. One of the endpoints of e however, say v, must be in $\textsc{Opt}$. We set $A = \{v\}$. Observe that $v \notin \textsc{Opt}'$ implies $\Gamma_G(v) \subseteq \textsc{Opt}'$ and set $A' = \Gamma_G(v)$ to be the second augmenting set. Since both A and A' are $\mathrm{Poly}(n)$-guessable, there is a polynomial-time 1.5-approximation algorithm for this variant of *wMinVerCov* reoptimization by Theorem 2 (the approximation ratio for *wMinVerCov* is $\sigma = 2$ [4]).

Other reoptimization problems where Theorem 2 applies are the minimum Steiner tree problem under removing a vertex from the terminal set and under decreasing the cost of an edge. The direct application of Theorem 2 results in approximation algorithms with $\mathcal{O}(n^{\log n + b})$ running time for both these problems where b is a constant, however parametrization allows reducing the running time to polynomial by an arbitrary small constant increase in the approximation ratio, see [18] for the details.

Definition 10. *A regressing modification $\mathcal{M}$ is* additive *(in minimization problems) if every feasible solution (possibly without some atoms) of the original instance I becomes feasible by adding a part of an optimal solution for the modified instance I':*

$$\begin{array}{r|l} \forall \textsc{Sol} \in \mathcal{R}(I) & \\ \exists \textsc{Opt}' \in \textsc{Optima}(I') & (\textsc{Sol} \setminus A) \cup A' \in \mathcal{R}(I') \\ \exists A \subseteq \textsc{Sol} & cost(I', \textsc{Sol} \setminus A) \leq cost(I, \textsc{Sol} \setminus A) \\ \exists A' \subseteq \textsc{Opt}' & \end{array}$$

Theorem 3. *Let cost be pseudo-additive, $\mathcal{M}$ be reversely cost-preserving (Definitions 5 and 6) and regressing additive, and A' be as in Definition 10. Then there is a $\frac{2\sigma-1}{\sigma}$-approximation algorithm for $\mathfrak{Re}_{\mathcal{M}}(\Pi)$ with $\mathcal{O}(\mathrm{F}(n)(\mathrm{Time}(Alg)+\mathrm{Poly}(n)))$ running time if A, A' are $\mathrm{F}(n)$-guessable.*

Proof. The approximation algorithm for reoptimization receives $(I, I', \textsc{Opt})$ as input. Let $\textsc{Opt}' \in \textsc{Optima}(I')$. The algorithm returns the better of $(\textsc{Opt} \setminus A) \cup A'$ and $\mathcal{S}^{\mathcal{G}}_{Alg}(I')$. Observe that, by Definitions 6 and 7.2,

$$\begin{aligned} cost(I', \textsc{Opt} \setminus A) &\underset{\text{reg. add.}}{\leq} cost(I, \textsc{Opt} \setminus A) \\ &\underset{\text{ps. add.}}{\leq} cost(I, \textsc{Opt}) \\ &\underset{\text{regressing}}{\leq} cost(I, \textsc{Opt}') \\ &\underset{\text{rev. cost p.}}{\leq} cost(I', \textsc{Opt}'). \end{aligned} \tag{5}$$

We estimate what follows based on pseudo-additivity and the above observation:

$$\begin{aligned} cost(I', \textsc{Opt} \setminus A \cup A') &\underset{\text{ps. add.}}{\leq} cost(I', \textsc{Opt} \setminus A) + cost(I', A') \\ &\underset{\text{obs.}}{\leq} cost(I', \textsc{Opt}') + cost(I', A'). \end{aligned}$$

By Lemma 3, $cost(I', \mathcal{S}^{\mathcal{G}}_{Alg}(I')) \leq \sigma\, cost(I', \textsc{Opt}') - (\sigma - 1) cost(I', A')$. Simple calculations show that one of the provided solutions gives an approximation ratio of $\frac{2\sigma-1}{\sigma}$. □

Corollary 5. *Let the assumptions be as in Theorem 3. Then $\mathfrak{Re}^{\rho}_{\mathcal{M}}(\mathcal{D}_{\Pi})$ for any $\rho \leq \sigma$ admits a $\frac{\sigma-\rho+\sigma\rho}{\sigma}$-approximation algorithm with $\mathcal{O}(\mathrm{F}(n)(\mathrm{Time}(Alg) + \mathrm{Poly}(n)))$ running time if A' is $\mathrm{F}(n)$-guessable.*

Proof. Plug in a factor of ρ in (5) in the proof of Theorem 3. The rest of the proof is the same. □

Below we provide an example application of Theorem 3. In addition to that Table 3 lists other problems where Corollary 5 applies.

Example 9. We illustrate Theorem 3 with the weighted minimum vertex cover problem under edge addition: $\mathfrak{Re}_{\mathcal{M}}(wMinVerCov)$. The self-reducible representation of the *wMinVerCov* problem was introduced in Example 8. The addition of an edge can make a vertex cover infeasible, but the solution in the modified instance must cover this edge with one of its endpoints. Therefore, adding this endpoint (A' satisfying Definition 10 contains only this endpoint) to the solution for the original instance makes it feasible for the modified one. Hence, Theorem 3 implies a 1.5-approximation algorithm that runs in polynomial time.

Table 3. The lists of the reoptimization problems where Corollary 5 applies. In case of the *SMT* problem, the resulting running times are sub-exponential, but can be reduced to polynomial by a simple parametrization. For the other problems, the corollary gives polynomial-time algorithms even if a constant number of items is altered.

Problem	Modification	Remarks
wMinVerCov	Edge addition	Implies ratios from [8]
	Vertex addition	
wMinDomSet	Edge removal	Implies ratio from [8]
wMinSetCov	Removal of an element from an arbitrary number of sets	See [18]
	Removal of a set of a bounded size	
	Removal of a bounded number of elements from a set	Implies ratios from [8,15]
SMT	A non-terminal becomes a terminal	See [18,19]
	Edge weight increase	
wMinFvs	Edge addition	

References

1. Archetti, C., Bertazzi, L., Speranza, M.G.: Reoptimizing the traveling salesman problem. Networks **42**(3), 154–159 (2003)
2. Archetti, C., Bertazzi, L., Speranza, M.G.: Reoptimizing the 0–1 knapsack problem. Technical report 267, University of Brescia (2006)
3. Ausiello, G., Escoffier, B., Monnot, J., Paschos, V.T.: Reoptimization of minimum and maximum traveling salesman's tours. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 196–207. Springer, Heidelberg (2006). https://doi.org/10.1007/11785293_20
4. Bar-Yehuda, R., Even, S.: A local-ratio theorem for approximating the weighted vertex cover problem. In: Analysis and Design of Algorithms for Combinatorial Problems, vol. 109, pp. 27–45 (1985)
5. Berg, T., Hempel, H.: Reoptimization of traveling salesperson problems: changing single edge-weights. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 141–151. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00982-2_12
6. Bilò, D., et al.: Reoptimization of steiner trees. In: Gudmundsson, J. (ed.) SWAT 2008. LNCS, vol. 5124, pp. 258–269. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69903-3_24
7. Bilò, D., et al.: Reoptimization of the shortest common superstring problem. Algorithmica **61**(2), 227–251 (2011)

8. Bilò, D., Widmayer, P., Zych, A.: Reoptimization of weighted graph and covering problems. In: Bampis, E., Skutella, M. (eds.) WAOA 2008. LNCS, vol. 5426, pp. 201–213. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-93980-1_16
9. Böckenhauer, H.-J., et al.: Reusing optimal TSP solutions for locally modified input instances. In: Navarro, G., Bertossi, L., Kohayakawa, Y. (eds.) TCS 2006. IIFIP, vol. 209, pp. 251–270. Springer, Boston, MA (2006). https://doi.org/10.1007/978-0-387-34735-6_21
10. Böckenhauer, H.-J., Hromkovič, J., Mömke, T., Widmayer, P.: On the hardness of reoptimization. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 50–65. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77566-9_5
11. Böckenhauer, H.-J., Hromkovič, J., Královič, R., Mömke, T., Rossmanith, P.: Reoptimization of steiner trees: changing the terminal set. Theor. Comput. Sci. **410**, 3428–3435 (2009)
12. Böckenhauer, H.-J., Komm, D.: Reoptimization of the metric deadline TSP. J. Discrete Algorithms **8**(1), 87–100 (2010)
13. Böckenhauer, H.-J., Freiermuth, K., Hromkovič, J., Mömke, T., Sprock, A., Steffen, B.: The steiner tree reoptimization problem with sharpened triangle inequality. In: Calamoneri, T., Diaz, J. (eds.) CIAC 2010. LNCS, vol. 6078, pp. 180–191. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13073-1_17
14. Escoffier, B., Ausiello, G., Bonifaci, V.: Complexity and approximation in reoptimization. In: Computability in Context: Computation and Logic in the Real World. Imperial College Press (2011)
15. Mikhailyuk, V.A.: Reoptimization of set covering problems. Cybern. Syst. Anal. **46**, 879–883 (2010)
16. Schäffter, M.W.: Scheduling with forbidden sets. Discrete Appl. Math. **72**(1–2), 155–166 (1997)
17. Vazirani, V.V.: Approximation Algorithms (2001)
18. Zych, A.: Reoptimization of NP-hard problems. PhD thesis, ETH Zürich (2012)
19. Zych, A., Bilò, D.: New reoptimization techniques applied to steiner tree problem. ENDM **37**, 387–392 (2011). LAGOS 2011
20. Goyal, K., Mömke, T.: Robust reoptimization of steiner trees. In: 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015) (2015)
21. Böckenhauer, H.J., Hromkovič, J., Komm, D.: Reoptimization of hard optimization problems. In: AAM Handbook of Approximation Algorithms and Metaheuristics, 2nd edn, chap. 25 (2018, to appear)