Chapter 4

Complexity and Approximation in Reoptimization

Giorgio Ausiello, Vincenzo Bonifaci* and Bruno Escoffier

Sapienza University of Rome,
Department of Computer and Systems Science,
00185 Rome, Italy
E-mail: ausiello@dis.uniroma1.it

Sapienza University of Rome,
Department of Computer and Systems Science,
00185 Rome, Italy, and
University of L'Aquila,
Department of Electrical and Information Engineering,
67040 L'Aquila, Italy
E-mail: bonifaci@dis.uniroma1.it

LAMSADE,
Université Paris Dauphine and CNRS,
75775 Paris Cedex 16, France
E-mail: escoffier@lamsade.dauphine.fr

In this chapter the following model is considered: We assume that an instance I of a computationally hard optimization problem has been solved and that we know the optimum solution of such an instance. Then a new instance I' is proposed, obtained by means of a slight perturbation of instance I. How can we exploit the knowledge we have on the solution of instance I to compute an (approximate) solution of instance I' in an efficient way? This computation model is called *reoptimization* and is of practical interest in various circumstances. In this chapter we first discuss what kind of performance we can expect for specific classes of problems and then we present some classical optimization problems (i.e. Max Knapsack, Min Steiner Tree, Scheduling) in which this approach has been fruitfully applied. Subsequently, we address vehicle routing

*This work was partially supported by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).

problems and we show how the reoptimization approach can be used to obtain good approximate solutions in an efficient way for some of these problems.

Contents

4.1. Introduction

In this chapter we illustrate the role that a new computational paradigm called *reoptimization* plays in the solution of NP-*hard problems* in various practical circumstances. As it is well known a great variety of relevant optimization problems are intrinsically difficult and no solution algorithms running in polynomial time are known for such problems. Although the existence of efficient algorithms cannot be ruled out at the present state of knowledge, it is widely believed that this is indeed the case. The most renowned approach to the solution of NP-hard problems consists in resorting to *approximation algorithms* which, in polynomial time, provide a suboptimal solution whose quality (measured as the ratio between the values of the optimum and approximate solution) is somehow guaranteed. In the last twenty years the definition of better and better approximation algorithms and the classification of problems based on the quality of approximation that can be achieved in polynomial time have been among the most important research directions in theoretical computer science and have produced a huge flow of literature [4, 36].

More recently a new computational approach to the solution of NP-hard problems has been proposed [1]. This approach can be meaningfully adopted when the following situation arises: Given a problem Π, the instances of Π that we need to solve are indeed all obtained by means of a slight perturbation of a given reference instance I. In such a case we can devote enough time to the exact solution of the reference instance I and then, any time that the solution for a new instance I' is required, we can apply a simple heuristic that efficiently provides a good approximate solution to I'. Let us imagine, for example, that we know that a traveling salesman has to visit a set S of, say, one thousand cities plus a few more cities that may change from time to time. In such case it is quite reasonable to devote a conspicuous amount of time to the exact solution of the traveling salesman problem on the set S and then to *reoptimize* the solution whenever the modified instance is known, with a (hopefully) very small computational effort.

To make the concept more precise let us consider the following simple example (Max Weighted Sat): Let ϕ be a Boolean formula in conjunctive normal form, consisting of m weighted clauses over n variables, and let us suppose we know a truth assignment τ such that the weight of the clauses satisfied by τ is maximum; let this weight be W. Suppose that now a new clause c with weight w over the same set of variables is provided and that we have to find a "good" although possibly not optimum truth assignment τ' for the new formula $\phi' = \phi \wedge c$. A very simple heuristic can always guarantee a 1/2 approximate truth assignment in constant time. The heuristic is the following: If $W \geq w$ then put $\tau' = \tau$, otherwise take as τ' any truth assignment that satisfies c. It is easy to see that, in any case, the weight provided by this heuristic will be at least 1/2 of the optimum.

Actually the reoptimization concept is not new. A similar approach has been applied since the early 1980s to some polynomial time solvable optimization problems such as minimum spanning tree [16] and shortest path [14, 32] with the aim to maintain the optimum solution of the given problem under input modification (say elimination or insertion of an edge or update of an edge weight). A big research effort devoted to the study of efficient algorithms for the dynamic maintenance of the optimum solution of polynomial time solvable optimization problems followed the first results. A typical example of this successful line of research has been the design of algorithms for the partially or fully dynamic maintenance of a minimum spanning tree in a graph under edge insertion and/or edge elimination [12, 22] where at any update, the computation of the new optimum solution requires at most $O(n^{1/3} \log n)$ amortized time per operation, much less than recomputing the optimum solution from scratch.

A completely different picture arises when we apply the concept of reoptimization to NP-hard optimization problems. In fact, reoptimization provides very different results when applied to polynomial time optimization problems with respect to what happens in the case of NP-hard problems.

In the case of NP-hard optimization problems, unless P=NP polynomial time reoptimization algorithms can only help us to obtain approximate solutions, since if we knew how to maintain an optimum solution under input updates, we could solve the problem optimally in polynomial time (see Section 4.3.1).

The application of the reoptimization computation paradigm to NP-hard optimization problems is hence aimed at two possible directions: either at achieving an approximate solution of better quality than we would have obtained without knowing the optimum solution of the base instance, or achieving an approximate solution of the same quality but at a lower computational cost (as is the case in our previous example).

In the first place the reoptimization model has been applied to classical NP-hard optimization problems such as scheduling (see Bartusch et al. [6], Schäffter [34], or Bartusch et al. [7] for practical applications). More recently it has been applied to various other NP-hard problems such as Steiner Tree [9, 13] or the Traveling Salesman Problem [1, 5, 8]. In this chapter we will discuss some general issues concerning reoptimization of NP-hard optimization problems and we will review some of the most interesting applications.

The chapter is organized as follows. First, in Section 4.2 we provide basic definitions concerning complexity and approximability of optimization problems and we show simple preliminary results. Then in Section 4.3 the computational power of reoptimization is discussed and results concerning the reoptimization of various NP-hard optimization problems are shown. Finally Section 4.4 is devoted to the application of the reoptimization concept to a variety of vehicle routing problems. While most of the results contained in Section 4.3 and Section 4.4 derive from the literature, it is worth noting that a few of the presented results – those for which no reference is given – appear in this paper for the first time.

4.2. Basic Definitions and Results

In order to characterize the performance of reoptimization algorithms and analyze their application to specific problems we have to provide first a basic introduction to the class of NP optimization problems (NPO problems) and to the notion of approximation algorithms and approximation classes. For a more extensive presentation of the theory of approximation the reader can refer to [4].

Definition 4.1. An NP *optimization (*NPO*) problem* Π is defined as a four-tuple $(\mathcal{I}, \mathrm{Sol}, m, \mathrm{opt})$ such that:

- $\mathcal{I}$ is the set of *instances* of Π and it can be recognized in polynomial time;
- given $I \in \mathcal{I}$, $\mathrm{Sol}(I)$ denotes the set of *feasible solutions* of I; for every $S \in \mathrm{Sol}(I)$, $|S|$ (the size of S) is polynomial in $|I|$ (the size of I); given any I and any S polynomial in $|I|$, one can decide in polynomial time if $S \in \mathrm{Sol}(I)$;
- given $I \in \mathcal{I}$ and $S \in \mathrm{Sol}(I)$, $m(I, S)$ denotes the value of S; m is polynomially computable and is commonly called *objective function*;
- $\mathrm{opt} \in \{\min, \max\}$ indicates the *type* of optimization problem.

As it is well known, several relevant optimization problems, known as NP-hard problems, are intrinsically difficult and no solution algorithms running in polynomial time are known for such problems. For the solution of NP-hard problems we have to resort to *approximation algorithms*, which in polynomial time provide a suboptimal solution of guaranteed quality.

Let us briefly recall the basic definitions regarding approximation algorithms and the most important approximation classes of NPO problems.

Given an NPO problem $\Pi = (\mathcal{I}, \mathrm{Sol}, m, \mathrm{opt})$, an optimum solution of an instance I of Π is denoted $S^*(I)$ and its measure $m(I, S^*(I))$ is denoted $\mathrm{opt}(I)$.

Definition 4.2. Given an NPO problem $\Pi = (\mathcal{I}, \mathrm{Sol}, m, \mathrm{opt})$, an *approximation algorithm* A is an algorithm that, given an instance I of Π, returns a feasible solution $S \in \mathrm{Sol}(I)$.

If A runs in polynomial time with respect to $|I|$, A is called a *polynomial time approximation algorithm* for Π.

The quality of an approximation algorithm is usually measured as the ratio $\rho_A(I)$, *approximation ratio*, between the value of the approximate solution, $m(I, A(I))$, and the value of the optimum solution $\mathrm{opt}(I)$. For minimization problems, therefore, the approximation ratio is in $[1, \infty)$, while for maximization problems it is in $[0, 1]$. According to the quality of approximation algorithms that can be designed for their solution, NPO problems can be classified as follows:

Definition 4.3. An NPO problem Π belongs to the class APX if there exists a polynomial time approximation algorithm A and a rational value r such that, given any instance I of Π, $\rho_A(I) \leqslant r$ (resp. $\rho_A(I) \geqslant r$) if Π is a minimization problem (resp. a maximization problem). In such case A is called an r-approximation algorithm.

Examples of combinatorial optimization problems belonging to the class APX are Max Weighted Sat, Min Vertex Cover, and Min Metric TSP.

For particular problems in APX a stronger form of approximability can indeed be shown. For such problems, given any rational $r > 1$ (or $r \in (0,1)$ for a maximization problem), there exists an algorithm A_r and a suitable polynomial p_r such that A_r is an r-approximation algorithm whose running time is bounded by p_r as a function of $|I|$. The family of algorithms A_r parametrized by r is called a *polynomial time approximation scheme* (PTAS).

Definition 4.4. An NPO problem Π belongs to the class PTAS if it admits a polynomial time approximation scheme A_r.

Examples of combinatorial optimization problems belonging to the class PTAS are Min Partitioning, Max Independent Set on Planar Graphs, and Min Euclidean TSP.

Notice that in the definition of PTAS, the running time of A_r is polynomial in the size of the input, but it may be exponential (or worse) in the inverse of $|r-1|$. A better situation arises when the running time is polynomial in both the input size and the inverse of $|r-1|$. In the favorable case when this happens, the algorithm is called a *fully polynomial time approximation scheme* (FPTAS).

Definition 4.5. An NPO problem Π belongs to the class FPTAS if it admits a fully polynomial time approximation scheme.

It is important to observe that, under the (reasonable) hypothesis that $\mathsf{P} \neq \mathsf{NP}$, it is possible to prove that $\mathsf{FPTAS} \subsetneq \mathsf{PTAS} \subsetneq \mathsf{APX} \subsetneq \mathsf{NPO}$.

4.3. Reoptimization of NP-hard Optimization Problem

As explained in the introduction, the reoptimization setting leads to interesting optimization problems for which the complexity properties and the existence of good approximation algorithms have to be investigated. This section deals with this question, and is divided into two parts: In Subsection 4.3.1, we give some general considerations on these reoptimization problems, both on the positive side (obtaining good approximate solutions) and on the negative side (hardness of reoptimization). In Subsection 4.3.2, we survey some results achieved on reoptimizing three well-known problems (the Min Steiner Tree problem, a scheduling problem, and the Max Knapsack problem).

4.3.1. *General properties*

As mentioned previously, if one wishes to get an approximate solution on the perturbed instance, she/he can compute it by applying directly, from scratch, a known approximation algorithm for the problem dealt (on the modified instance). In other words, reoptimizing is at least as easy as approximating. The goal of reoptimization is to determine if it is possible to fruitfully use our knowledge on the initial instance in order to:

- either achieve better approximation ratios;
- or devise much faster algorithms;
- or both!

In this section, we present some general results dealing with reoptimization properties of some NPO problems. We first focus on a class of hereditary problems, then we discuss the differences between weighted and unweighted versions of classical problems, and finally present some ways to achieve hardness results in reoptimization.

Of course, many types of problems can be considered, and for each of them many ways to modify the instances might be investigated. We mainly focus here on graph problems where a modification consists of adding a new vertex on the instance, but show with various examples that the approaches we present are also valid in many other cases.

4.3.1.1. *Hereditary problems*

We say that a property on graphs is hereditary if the following holds: If $G = (V, E)$ satisfies this property, then for any $V' \subseteq V$, the subgraph $G[V']$ induced by V' verifies the property. Following this definition, for instance, being independent [a], being bipartite, or being planar are three hereditary properties. Now, let us define problems based on hereditary properties.

Definition 4.6. We call Hered the class of problems consisting, given a vertex-weighted graph $G = (V, E, w)$, of finding a subset of vertices S (i) such that $G[S]$ satisfies a given hereditary property (ii) that maximizes $w(S) = \sum_{v \in S} w(v)$.

[a] i.e. having no edge.

Hereditary problems have been studied before as a natural generalization of important combinatorial problems [27]. For instance, Max Weighted Independent Set, Max Weighted Bipartite Subgraph, Max Weighted Planar Subgraph are three famous problems in Hered that correspond to the three hereditary properties given above.

For all these problems, we have a simple reoptimization strategy that achieves a ratio 1/2, based on the same idea used in the introduction. Note that this is a huge improvement for some problems respect to their approximability properties; for instance, it is well known that Max Weighted Independent Set is not approximable within any constant ratio, if $\mathsf{P} \neq \mathsf{NP}$[b].

Theorem 4.1. *Let* Π *be a problem in* Hered. *Under a vertex insertion, reoptimizing* Π *is approximable within ratio 1/2 (in constant time).*

Proof. Let $I = (G, w)$ be the initial instance of Π, $I' = (G', w')$ be the final instance (a new vertex v has been inserted), S^* be an optimum solution on I, and $S^*_{I'}$ be an optimum solution on I'. Notice that $w'(u) = w(u)$ for all $u \neq v$.

Getting a 1/2-approximate solution is very easy: just consider the best solution among S^* and (if feasible) $S_1 := \{v\}$. Solution S^* is feasible by heritability. We can also assume S_1 feasible, as otherwise by heritability no feasible solution can include v, and S^* must be optimal. Finally, by heritability, $S^*_{I'} \setminus \{v\}$ is a feasible solution on the initial instance. Then, $w'(S^*_{I'}) \leq w'(S^*) + w'(v) = w'(S^*) + w'(S_1) \leq 2 \max(w'(S^*), w'(S_1))$. $\square$

Now, let us try to outperform this trivial ratio 1/2. A first idea that comes to mind is to improve the solution S_1 of the previous proof since it only contains one vertex. In particular, one can think of applying an approximation algorithm on the "remaining instance after taking v". Consider for instance Max Weighted Independent Set, and revisit the proof of the previous property. If $S^*_{I'}$ does not take the new vertex v, then our initial solution S^* is optimum. If $S^*_{I'}$ takes v, then consider the remaining instance $I_{\overline{v}}$ after having removed v and its neighbors. Suppose that we have a ρ-approximate solution S_2 on this instance $I_{\overline{v}}$. Then $S_2 \cup \{v\}$ is a feasible solution of weight:

$$w(S_2 \cup \{v\}) \geq \rho(w(S^*_{I'}) - w(v)) + w(v) = \rho w(S^*_{I'}) + (1 - \rho) w(v). \quad (4.1)$$

On the other hand, of course :

$$w(S^*) \geq w(S^*_{I'}) - w(v). \quad (4.2)$$

[b]And not even within ratio $n^{1-\varepsilon}$ for any $\varepsilon > 0$, under the same hypothesis [37].

If we output the best solution S among S^* and $S_2 \cup \{v\}$, then, by adding equations (4.1) and (4.2) with coefficients 1 and $(1-\rho)$, we get:

$$w(S) \geq \frac{1}{2-\rho} w(S^*_{I'}).$$

Note that this ratio is always better than ρ.

This technique is actually quite general and applies to many problems (not only graph problems and maximization problems). We illustrate this on two well-known problems: Max Weighted Sat (Theorem 4.2) and Min Vertex Cover (Theorem 4.3). We will also use it for Max Knapsack in Section 4.3.2.

Theorem 4.2. *Under the insertion of a clause, reoptimizing* Max Weighted Sat *is approximable within ratio 0.81.*

Proof. Let ϕ be a conjunction of clauses over a set of binary variables, each clause being given with a weight, and let $\tau^*(\phi)$ be an initial optimum solution. Let $\phi' := \phi \wedge c$ be the final formula, where the new clause $c = l_1 \vee l_2 \vee \ldots \vee l_k$ (where l_i is either a variable or its negation) has weight $w(c)$.

We consider k solutions τ_i, $i = 1, \ldots, k$. Each τ_i is built as follows:

- We set l_i to true;
- We replace in ϕ each occurrence of l_i and $\overline{l_i}$ with its value;
- We apply a ρ-approximation algorithm on the remaining instance (note that the clause c is already satisfied); together with l_i, this is a particular solution τ_i.

Then, our reoptimization algorithm outputs the best solution τ among $\tau^*(\phi)$ and the τ_is.

As previously, if the optimum solution $\tau^*(\phi')$ on the final instance does not satisfy c, then $\tau^*(\phi)$ is optimum. Otherwise, at least one literal in c, say l_i, is true in $\tau^*(\phi')$. Then, it is easy to see that

$$w(\tau_i) \geq \rho(w(\tau^*(\phi')) - w(c)) + w(c) = \rho w(\tau^*(\phi')) + (1-\rho)w(c).$$

On the other hand, $w(\tau^*(\phi)) \geq w(\tau^*(\phi')) - w(c)$, and the following result follows:

$$w(\tau) \geq \frac{1}{2-\rho} w(\tau^*(\phi')).$$

The fact that Max Weighted Sat is approximable within ratio $\rho = 0.77$ [3] concludes the proof. □

It is worth noticing that the same ratio $(1/(2-\rho))$ is achievable for other satisfiability or constraint satisfaction problems. For instance, using the result of Johnson [24], reoptimizing Max Weighted E3SAT[c] when a new clause is inserted is approximable within ratio 8/9.

Let us now focus on a minimization problem, namely Min Vertex Cover. Given a vertex-weighted graph $G = (V, E, w)$, the goal in this problem is to find a subset $V' \subseteq V$ such that (i) every edge $e \in E$ is incident to at least one vertex in V', and (ii) the global weight of V', that is, $\sum_{v \in V'} w(v)$ is minimized.

Theorem 4.3. *Under a vertex insertion, reoptimizing* Min Vertex Cover *is approximable within ratio 3/2.*

Proof. Let v denote the new vertex and S^* the initial given solution. Then, $S^* \cup \{v\}$ is a vertex cover on the final instance. If $S^*_{I'}$ takes v, then $S^* \cup \{v\}$ is optimum.

From now on, suppose that $S^*_{I'}$ does not take v. Then it has to take all its neighbors $N(v)$. $S^* \cup N(v)$ is a feasible solution on the final instance. Since $w(S^*) \leq w(S^*_{I'})$, we get:

$$w(S^* \cup N(v)) \leq w(S^*_{I'}) + w(N(v)). \tag{4.3}$$

Then, as for Max Weighted Independent Set, consider the following feasible solution S_1:

- Take all the neighbors $N(v)$ of v in S_1;
- Remove v and its neighbors from the graph;
- Apply a ρ-approximation algorithm on the remaining graph and add these vertices to S_1.

Since we are in the case where $S^*_{I'}$ does not take v, it has to take all its neighbors, and finally:

$$w(S_1) \leq \rho(w(S^*_{I'}) - w(N(v))) + w(N(v)) = \rho w(S^*_{I'}) - (\rho - 1) w(N(v)). \tag{4.4}$$

Of course, we take the best solution S among $S^* \cup N(v)$ and S_1. Then, a convex combination of equations (4.3) and (4.4) leads to:

$$w(S) \leq \frac{2\rho - 1}{\rho} w(S^*_{I'}).$$

The results follows since Min Vertex Cover is well known to be approximable within ratio 2. □

[c]Restriction of Max Weighted Sat when all clauses contain exactly three literals.

To conclude this section, we point out that these results can be generalized when several vertices are inserted. Indeed, if a constant number $k > 1$ of vertices are added, one can reach the same ratio with similar arguments by considering all the 2^k possible subsets of new vertices in order to find the ones that will belong to the new optimum solution. This brute force algorithm is still very fast for small constant k, which is the case in the reoptimization setting with slight modifications of the instance.

4.3.1.2. *Unweighted problems*

In the previous subsection, we considered the general cases where vertices (or clauses) have a weight. It is well known that all the problems we focused on are already NP-hard in the unweighted case, i.e. when all vertices/clauses receive weight 1. In this (very common) case, the previous approximation results on reoptimization can be easily improved. Indeed, since only one vertex is inserted, the initial optimum solution has an *absolute error* of at most one on the final instance, i.e.:

$$|S^*| \geq |S^*_{I'}| - 1.$$

Then, in some sense we don't really need to reoptimize since S^* is already a very good solution on the final instance (note also that since the reoptimization problem is NP-hard, we cannot get rid of the constant -1). Dealing with approximation ratio, we derive from this remark, with a standard technique, the following result:

Theorem 4.4. *Under a vertex insertion, reoptimizing any unweighted problem in* Hered *admits a PTAS.*

Proof. Let $\varepsilon > 0$, and set $k = \lceil 1/\varepsilon \rceil$. We consider the following algorithm:

(1) Test all the subsets of V of size at most k, and let S_1 be the largest one such that $G[S_1]$ satisfies the hereditary property;
(2) Output the largest solution S between S_1 and S^*.

Then, if $S^*_{I'}$ has size at most $1/\varepsilon$, we found it in step 1. Otherwise, $|S^*_{I'}| \geq 1/\varepsilon$ and:

$$\frac{|S^*|}{|S^*_{I'}|} \geq \frac{|S^*_{I'}| - 1}{|S^*_{I'}|} \geq 1 - \varepsilon.$$

Of course, the algorithm is polynomial as long as ε is a constant. □

In other words, the PTAS is derived from two properties: the absolute error of 1, and the fact that problems considered are *simple*. Following [30], a problem is called *simple* if, given any fixed constant k, it is polynomial to determine whether the optimum solution has value at most k (maximization) or not.

This result easily extends to other simple problems, such as Min Vertex Cover, for instance. It also generalizes when several (a constant number of) vertices are inserted, instead of only 1.

However, it is interesting to notice that, for some other (unweighted) problems, while the absolute error 1 still holds, we cannot derive a PTAS as in Theorem 4.4 because they are not simple. One of the most famous such problems is the Min Coloring problem. In this problem, given a graph $G = (V, E)$, one wishes to partition V into a minimum number of independent sets (called colors) $V_1, \ldots, V_k$. When a new vertex is inserted, an absolute error 1 can be easily achieved while reoptimizing. Indeed, consider the initial coloring and add a new color which contains only the newly inserted vertex. Then this coloring has an absolute error of 1 since a coloring on the final graph cannot use fewer colors than an optimum coloring on the initial instance.

However, deciding whether a graph can be colored with 3 colors is an NP-hard problem. In other words, Min Coloring is not simple. We will discuss the consequence of this fact in the section on hardness of reoptimization.

To conclude this section, we stress the fact that there exist, obviously, many problems that do not involve weights and for which the initial optimum solution cannot be directly transformed into a solution on the final instance with absolute error 1. Finding the longest cycle in a graph is such a problem: adding a new vertex may change considerably the size of an optimum solution.

4.3.1.3. *Hardness of reoptimization*

As mentioned earlier, the fact that we are interested in slight modifications of an instance on which we have an optimum solution makes the problem somehow simpler, but unfortunately does not generally allow a jump in complexity. In other words, reoptimizing is generally NP-hard when the underlying problem is NP-hard.

In some cases, the proof of NP-hardness is immediate. For instance, consider a graph problem where modifications consists of inserting a new vertex. Suppose that we had an optimum reoptimization algorithm for this problem. Then, starting from the empty graph, and adding the vertices one by one, we could find an optimum solution on any graph on n vertices by using iteratively n times the reoptimization algorithm. Hence, the underlying problem would be polynomial. In conclusion, the reoptimization version is also NP-hard when the underlying problem is NP-hard. This argument is also valid for other problems under other kinds of modifications. Actually, it is valid as soon as, *for any instance I, there is a polynomial-time solvable instance I' (the empty graph in our example) that can be generated in polynomial time and such that a polynomial number of modifications transform I' into I.*

In other cases, the hardness does not directly follow from this argument, and a usual polynomial time reduction has to be provided. This situation occurs, for instance, in graph problems where the modification consists of deleting a vertex. As we will see later, such hardness proofs have been given, for instance, for some vehicle routing problems (in short, VRP).

Let us now focus on the hardness of approximation in the reoptimization setting. As we have seen in particular in Theorem 4.4, the knowledge of the initial optimum solution may help considerably in finding an approximate solution on the final instance. In other words, it seems quite hard to prove a lower bound on reoptimization. And in fact, few results have been obtained so far.

One method is to transform the reduction used in the proof of NP-hardness to get an inapproximability bound. Though more difficult than in the usual setting, such proofs have been provided for reoptimization problems, in particular for VRP problems, mainly by introducing very large distances (see Section 4.4).

Let us now go back to Min Coloring. As we have said, it is NP-hard to determine whether a graph is colorable with 3 colors or not. In the usual setting, this leads to an inapproximability bound of $4/3 - \varepsilon$ for any $\varepsilon > 0$. Indeed, an approximation algorithm within ratio $\rho = 4/3 - \varepsilon$ would allow us to distinguish between 3-colorable graphs and graphs for which we need at least 4 colors. Now, we can show that this result *remains true for the reoptimization of the problem*:

Theorem 4.5. *Under a vertex insertion, reoptimizing* Min Coloring *cannot be approximated within a ratio $4/3 - \varepsilon$, for any $\varepsilon > 0$.*

Proof. The proof is actually quite straightforward. Assume you have such a reoptimization algorithm A within a ratio $\rho = 4/3 - \varepsilon$. Let $G = (V, E)$ be a graph with $V = \{v_1, \cdots, v_n\}$. We consider the subgraphs G_i of G induced by $V_i = \{v_1, v_2, \cdots, v_i\}$ (in particular $G_n = G$). Suppose that you have a 3-coloring of G_i, and insert v_{i+1}. If G_{i+1} is 3-colorable, then A outputs a 3-coloring. Moreover, if G_i is not 3-colorable, then neither is G_{i+1}. Hence, starting from the empty graph, and iteratively applying A, we get a 3-coloring of G_i if and only if G_i is 3-colorable. Eventually, we are able to determine whether G is 3-colorable or not. □

This proof is based on the fact that Min Coloring is not simple (according to the definition previously given). A similar argument, leading to inapproximability results in reoptimization, can be applied to other non simple problems (under other modifications). It has been in particular applied to a scheduling problem (see Section 4.3.2).

For other optimization problems however, such as MinTSP in the metric case, finding a lower bound in approximability (if any!) seems a challenging task.

Let us finally mention another kind of negative result. In the reoptimization setting, we look somehow for a possible stability when slight modifications occur on the instance. We try to measure how much the knowledge of a solution on the initial instance helps to solve the final one. Hence, it is natural to wonder whether one can find a good solution in the "neighborhood" of the initial optimum solution, or if one has to change almost everything. Do neighboring instances have neighboring optimum/good solutions? As an answer to these questions, several results show that, for several problems, approximation algorithms that only "slightly" modify the initial optimum solution cannot lead to good approximation ratios. For instance, for reoptimizing MinTSP in the metric case, if you want a ratio better than 3/2 (guaranteed by a simple heuristic), then you have to change (on some instances) a significant part of your initial solution [5]. This kind of result, weaker than an inapproximability bound, provides information on the stability under modifications and lower bounds on classes of algorithms.

4.3.2. *Results on some particular problems*

In the previous section, we gave some general considerations on the reoptimization of NP-hard optimization problems. The results that have been presented follow, using simple methods, from the structural properties of the problem dealt with and/or from known approximation results. We now focus on particular problems for which specific methods have been devised, and briefly mention, without proofs, the main results obtained so far. We concentrate on the Min Steiner Tree problem, on a scheduling problem, and on the Max Knapsack problem. Vehicle routing problems, which concentrated a large attention in reoptimization, deserve, in our opinion, a full section (Section 4.4), in which we also provide some of the most interesting proofs in the literature together with a few new results.

4.3.2.1. *Min Steiner Tree*

The Min Steiner Tree problem is a generalization of the Min Spanning Tree problem where only a subset of vertices (called terminal vertices) have to be spanned. Formally, we are given a graph $G = (V, E)$, a non-negative distance $d(e)$ for any $e \in E$, and a subset $R \subseteq V$ of *terminal* vertices. The goal is to connect the terminal vertices with a minimum global distance, i.e. to find a tree $T \subseteq E$ that spans all vertices in R and minimizes $d(T) = \sum_{e \in T} d(e)$. It is generally assumed that the graph is complete, and the distance function is metric (i.e. $d(x, y) + d(y, z) \geq d(x, z)$ for any vertices x, y, z): indeed, the general problem reduces to this case by initially computing shortest paths between pairs of vertices.

Min Steiner Tree is one of the most famous network design optimization problems. It is NP-hard, and has been studied intensively from an approximation viewpoint (see [18] for a survey on these results). The best known ratio obtained so far is $1 + \ln(3)/2 \simeq 1.55$ [31].

Reoptimization versions of this problem have been studied with modifications on the vertex set [9, 13]. In Escoffier et al. [13], the modification consists of the insertion of a new vertex. The authors study the cases where the new vertex is terminal or non-terminal.

Theorem 4.6 ([13]). *When a new vertex is inserted (either terminal or not), then reoptimizing the* Min Steiner Tree *problem can be approximated within ratio 3/2.*

Moreover, the result has been generalized to the case in which several vertices are inserted. Interestingly, when p non-terminal vertices are inserted, then reoptimizing the problem is still 3/2-approximable (but the running time grows very fast with p). On the other hand, when terminal vertices are added, the obtained ratio decreases (but the running time remains very low). The strategies consist, roughly speaking, of merging the initial optimum solution with Steiner trees computed on the set of new vertices and/or terminal vertices. The authors tackle also the case where a vertex is removed from the vertex set, and provide a lower bound for a particular class of algorithms.

Böckenhauer et al. [9] consider a different instance modification. Rather than inserting/deleting a vertex, the authors consider the case where the status of a vertex changes: either a terminal vertex becomes non-terminal, or vice versa. The obtained ratio is also 3/2.

Theorem 4.7 ([9]). *When the status (terminal / non-terminal) of a vertex changes, then reoptimizing the* Min Steiner Tree *problem can be approximated within ratio 3/2.*

Moreover, they exhibit a case where this ratio can be improved. When all the distances between vertices are in $\{1, 2, \cdots, r\}$, for a fixed constant r, then reoptimizing Min Steiner Tree (when changing the status of one vertex) is still NP-hard but admits a PTAS.

Note that in both cases (changing the status of a vertex or adding a new vertex), no inapproximability results have been achieved, and this is an interesting open question.

4.3.2.2. *Scheduling*

Due to practical motivations, it is not surprising that scheduling problems received attention dealing with the reconstruction of a solution (often called *rescheduling*) after an instance modification, such as a machine breakdown, an increase of a job processing time, etc. Several works have been proposed to provide a sensitivity analysis of these problems under such modifications. A typical question is to determine under which modifications and/or conditions the initial schedule remains optimal. We refer the reader to the comprehensive article [20] where the main results achieved in this field are presented.

Dealing with the reoptimization setting we develop in this chapter, Schäffter [34] proposes interesting results on a problem of *scheduling with forbidden sets*. In this problem, we have a set of jobs $V = \{v_1, \cdots, v_n\}$, each job having a processing time. The jobs can be scheduled in parallel (the number of machines is unbounded), but there is a set of constraints on these parallel schedules: A constraint is a set $F \subseteq V$ of jobs that cannot be scheduled in parallel (all of them at the same time). Then, given a set $\mathcal{F} = \{F_1, \cdots, F_k\}$ of constraints, the goal is to find a schedule that respects each constraint and that minimizes the latest completion time (makespan). Many situations can be modeled this way, such as the m-Machine Problem (for fixed m), hence the problem is NP-complete (and even hard to approximate).

Schäffter considers reoptimization when either a new constraint F is added to $\mathcal{F}$, or a constraint $F_i \in \mathcal{F}$ disappears. Using reductions from the Set Splitting problem and from the Min Coloring problem, he achieves the following inapproximability results:

Theorem 4.8 ([34]). *If* $\mathsf{P} \neq \mathsf{NP}$, *for any* $\varepsilon > 0$, *reoptimizing the scheduling with forbidden sets problem is inapproximable within ratio* $3/2 - \varepsilon$ *under a constraint insertion, and inapproximable within ratio* $4/3 - \varepsilon$ *under a constraint deletion.*

Under a constraint insertion Schäffter also provides a reoptimization strategy that achieves approximation ratio 3/2, thus matching the lower bound of Theorem 4.8. It consists of a simple local modification of the initial scheduling, by shifting one task (at the end of the schedule) in order to ensure that the new constraint is satisfied.

4.3.2.3. *Max Knapsack*

In the Max Knapsack problem, we are given a set of n objects $O = \{o_1, \ldots, o_n\}$, and a capacity B. Each object has a weight w_i and a value v_i. The goal is to choose a subset O' of objects that maximizes the global value $\sum_{o_i \in O'} v_i$ but that respects the capacity constraint $\sum_{o_i \in O'} w_i \leq B$.

This problem is (weakly) NP-hard, but admits an FPTAS [23]. Obviously, the reoptimization version admits an FPTAS too. Thus, Archetti et al. [2] are interested in using classical approximation algorithms for Max Knapsack to derive reoptimization algorithms with better approximation ratios but with the same running time. The modifications considered consist of the insertion of a new object in the instance.

Though not being a graph problem, it is easy to see that the Max Knapsack problem satisfies the required properties of heritability given in Section 4.3.1 (paragraph on hereditary problems). Hence, the reoptimization version is 1/2-approximable in constant time; moreover, if we have a ρ-approximation algorithm, then the reoptimization strategy presented in Section 4.3.1 has ratio $\frac{1}{2-\rho}$ [2]. Besides, Archetti et al. [2] show that this bound is tight for several classical approximation algorithms for Max Knapsack.

Finally, studying the issue of sensitivity presented earlier, they show that any reoptimization algorithm that does not consider objects discarded by the initial optimum solution cannot have ratio better than 1/2.

4.4. Reoptimization of Vehicle Routing Problems

In this section we survey several results concerning the reoptimization of vehicle routing problems under different kinds of perturbations. In particular, we focus on several variants of the Traveling Salesman Problem (TSP), which we define below.

The TSP is a well-known combinatorial optimization problem that has been the subject of extensive studies – here we only refer the interested reader to the monographs by Lawler et al. [26] and Gutin and Punnen [19]. The TSP has been used since the inception of combinatorial optimization as a testbed for experimenting a whole array of algorithmic paradigms and techniques, so it is just natural to also consider it from the point of view of reoptimization.

Definition 4.7. An instance I_n of the Traveling Salesman Problem is given by the distance between every pair of n nodes in the form of an $n \times n$ matrix d, where $d(i,j) \in \mathbb{Z}_+$ for all $1 \leq i,j \leq n$. A feasible solution for I_n is a *tour*, that is, a directed cycle spanning the node set $N := \{1, 2, \ldots, n\}$.

Notice that we have not defined an objective function yet; so far we have only specified the structure of instances and feasible solutions. There are several possibilities for the objective function and each of them gives rise to a different optimization problem. We need a few definitions. The *weight* of a tour T is the quantity $w(T) := \sum_{(i,j)\in T} d(i,j)$. The *latency of a node* $i \in N$ with respect to a given tour T is the total distance along the cycle T from node 1 to node i. The *latency* of T, denoted by $\ell(T)$, is the sum of the latencies of the nodes of T.

Table 4.1. Best known results on the approximability of the standard and reoptimization versions of vehicle routing problems (AR = approximation ratio, $\Pi+$ = vertex insertion, $\Pi-$ = vertex deletion, $\Pi\pm$ = distance variation).

Problem Π	AR(Π)	**Ref.**	AR($\Pi+$)	AR($\Pi-$)	AR($\Pi\pm$)	**Ref.**
Min TSP	unbounded	[33]	unb.	unb.	unb.	[5, 8]
Min MTSP	1.5	[11]	1.34	-	1.4	[1, 9]
Min ATSP	$O(\log n)$	[15]	2	2	-	this work
Max TSP	0.6	[25]	$0.66 - O(n^{-1})$	-	-	this work
Max MTSP	0.875	[21]	$1 - O(n^{-1/2})$	-	-	[5]
MLP	3.59	[10]	3	-	-	this work

The matrix d obeys the *triangle inequality* if for all $i, j, k \in N$ we have $d(i, j) \leq d(i, k) + d(k, j)$. The matrix d is said to be a *metric* if it obeys the triangle inequality and $d(i, j) = d(j, i)$ for all $i, j \in N$.

In the rest of the section we will consider the following problems:

(1) Minimum Traveling Salesman Problem (Min TSP): find a tour of minimum weight;
(2) Minimum Metric TSP (Min MTSP): restriction of Min TSP to the case when d is a metric;
(3) Minimum Asymmetric TSP (Min ATSP): restriction of Min TSP to the case when d obeys the triangle inequality;
(4) Maximum TSP (Max TSP): find a tour of maximum weight;
(5) Maximum Metric TSP (Max MTSP): restriction of Max TSP to the case when d is a metric;
(6) Minimum Latency Problem (MLP): find a tour of minimum latency; d is assumed to be a metric.

TSP-like problems other than those above have also been considered in the literature from the point of view of reoptimization; in particular, see Böckenhauer et al. [8] for a hardness result on the TSP with deadlines.

Given a vehicle routing problem Π from the above list, we will consider the following reoptimization variants, each corresponding to a different type of perturbation of the instance: insertion of a node ($\Pi+$), deletion of a node ($\Pi-$), and variation of a single entry of the matrix d ($\Pi\pm$).

In the following, we will sometimes refer to the initial problem Π as the *static* problem. In Table 4.1 we summarize the approximability results known for the static and reoptimization versions of the problems above under these types of perturbations.

Some simple solution methods are common to several of the problems we study in this section. We define here two such methods; they will be used in the remainder of the section.

Algorithm 1 (Nearest Insertion). Given an instance I_{n+1} and a tour T on the set $\{1, \dots, n\}$, find a node $i^* \in \text{argmin}_{1 \le i \le n}\, d(i, n+1)$. Obtain the solution by inserting node $n+1$ either immediately before or immediately after i^* in the tour (depending on which of these two solutions is best).

Algorithm 2 (Best Insertion). Given an instance I_{n+1} and a tour T on the set $\{1, \dots, n\}$, find a pair $(i^*, j^*) \in \text{argmin}_{(i,j) \in T}\, d(i, n+1) + d(n+1, j) - d(i, j)$. Obtain the solution by inserting node $n+1$ between i^* and j^* in the tour.

4.4.1. *The Minimum Traveling Salesman Problem*

4.4.1.1. *The general case*

We start by considering the Min TSP. It is well known that in the standard setting the problem is very hard to approximate in the sense that it cannot be approximated within any factor that is polynomial in the number of nodes [33]. It turns out that the same result also holds for the reoptimization versions of the problem, which shows that in this particular case the extra information available through the optimal solution to the original instance does not help at all.

Theorem 4.9 ([5, 8]). *Let p be a polynomial. Then each of* Min TSP+, Min TSP−, *and* Min TSP± *is not $2^{p(n)}$-approximable, unless* P=NP.

Proof. We only give the proof for Min TSP−; the other proofs follow a similar approach. We use the so-called *gap technique* from Sahni and Gonzales [33]. Consider the following problem, Restricted Hamiltonian Cycle (RHC): Given an undirected graph $G = (V, E)$ and a Hamiltonian path P between two nodes a and b in G, determine whether there exists a Hamiltonian cycle in G. This problem is known to be NP-complete [28]. We prove the claim of the theorem by showing that any approximation algorithm for Min TSP− with ratio $2^{p(n)}$ can be used to solve RHC in polynomial time.

Consider an instance of RHC, that is, a graph $G = (V, E)$ on n nodes, two nodes $a, b \in V$ and a Hamiltonian path P from a to b. Without loss of generality we can assume that $V = \{1, \dots, n\}$. We can construct in polynomial time the following TSP instance I_{n+1} on node set $\{1, \dots, n, n+1\}$:

- $d(i,j) = 1$ if $(i,j) \in E$;
- $d(n+1,a) = d(b,n+1) = 1$;
- all other entries of the matrix d have value $2^{p(n)} \cdot n + 1$.

Since all entries are at least 1, the tour $T^*_{n+1} := P \cup \{(b,n+1),(n+1,a)\}$ is an optimum solution of I_{n+1}, with weight $w(T^*_{n+1}) = n+1$. Thus, (I_{n+1}, T^*_{n+1}) is an instance of Min TSP$-$. Let T^*_n be an optimum solution of instance I_n. Then $w(T^*_n) = n$ if and only if G has a Hamiltonian cycle. Finally, a $2^{p(n)}$-approximation algorithm for Min TSP$-$ allows us to decide whether $w(T^*_n) = n$. □

4.4.1.2. *Minimum Metric TSP*

In the previous section we have seen that no constant-factor approximation algorithm exists for reoptimizing the Minimum TSP in its full generality. To obtain such a result, we are forced to restrict the problem somehow. A very interesting case for many applications is when the matrix d is a metric, that is, the Min MTSP. This problem admits a 3/2-approximation algorithm, due to Christofides [11], and it is currently open whether this factor can be improved. Interestingly, it turns out that the reoptimization version Min MTSP+ is (at least if one considers the currently best known algorithms) easier than the static problem: It allows a 4/3-approximation – although, again, we do not know whether even this factor may be improved via a more sophisticated approach.

Theorem 4.10 ([5]). Min MTSP+ *is approximable within ratio* 4/3.

Proof. The algorithm used to prove the upper bound is a simple combination of Nearest Insertion and of the well-known algorithm by Christofides [11]; namely, both algorithms are executed and the solution returned is the one having the lower weight.

Consider an optimum solution T^*_{n+1} of the final instance I_{n+1}, and the solution T^*_n available for the initial instance I_n. Let i and j be the two neighbors of vertex $n+1$ in T^*_{n+1}, and let T_1 be the tour obtained from T^*_n with the Nearest Insertion rule. Furthermore, let v^* be the vertex in $\{1,\ldots,n\}$ whose distance to $n+1$ is the smallest.

Using the triangle inequality, we easily get $w(T_1) \le w(T^*_{n+1}) + 2d(v^*, n+1)$ where, by definition of v^*, $d(v^*,n+1) = \min\{d(k,n+1) : k = 1,\ldots,n\}$. Thus

$$w(T_1) \le w(T^*_{n+1}) + 2\max(d(i,n+1), d(j,n+1)). \qquad (4.5)$$

Now consider the algorithm of Christofides applied on I_{n+1}. This gives a tour T_2 of length at most $(1/2)w(T^*_{n+1}) + \mathrm{MST}(I_{n+1})$, where $\mathrm{MST}(I_{n+1})$ is the weight of a minimum spanning tree on I_{n+1}. Note that $\mathrm{MST}(I_{n+1}) \leq w(T^*_{n+1}) - \max(d(i, n+1), d(j, n+1))$. Hence

$$w(T_2) \leq \frac{3}{2}w(T^*_{n+1}) - \max(d(i, n+1), d(j, n+1)). \tag{4.6}$$

The result now follows by combining equations (4.5) and (4.6), because the weight of the solution given by the algorithm is $\min(w(T_1), w(T_2)) \leq (1/3)w(T_1) + (2/3)w(T_2) \leq (4/3)w(T^*_{n+1})$. □

The above result can be generalized to the case when more than a single vertex is added in the perturbed instance. Let Min MTSP+k be the corresponding problem when k vertices are added. Then it is possible to give the following result, which gives a trade-off between the number of added vertices and the quality of the approximation guarantee.

Theorem 4.11 ([5]). *For any* $k \geq 1$*,* Min MTSP+k *is approximable within ratio* $3/2 - 1/(4k + 2)$*.*

Reoptimization under variation of a single entry of the distance matrix (that is, problem Min MTSP±) has been considered by Böckenhauer et al. [9].

Theorem 4.12 ([9]). Min MTSP± *is approximable within ratio* 7/5*.*

4.4.1.3. *Minimum Asymmetric TSP*

The Minimum Asymmetric Traveling Salesman Problem is another variant of the TSP that is of interest for applications, as it generalizes the Metric TSP. Unfortunately, in the static case there seems to be a qualitative difference with respect to the approximability of Minimum Metric TSP: While in the latter case a constant approximation is possible, for Min ATSP the best known algorithms give an approximation ratio of $\Theta(\log n)$. The first such algorithm was described by Frieze et al. [17] and has an approximation guarantee of $\log_2 n$. The currently best algorithm is due to Feige and Singh [15] and gives approximation $(2/3)\log_2 n$. The existence of a constant approximation for Min ATSP is an important open problem.

Turning now to reoptimization, there exists a non-negligible gap between the approximability of the static version and of the reoptimization version. In fact, reoptimization drastically simplifies the picture: Min ATSP+ is approximable within ratio 2, as we proceed to show.

Theorem 4.13. Min ATSP+ *is approximable within ratio* 2.

Proof. The algorithm used to establish the upper bound is extremely simple: just add the new vertex between an arbitrarily chosen pair of consecutive vertices in the old optimal tour. Let T be the tour obtained by inserting node $n+1$ between two consecutive nodes i and j in T_n^*. We have:

$$w(T) = w(T_n^*) + d(i, n+1) + d(n+1, j) - d(i, j).$$

By triangle inequality, $d(n+1, j) \leq d(n+1, i) + d(i, j)$. Hence

$$w(T) \leq w(T_n^*) + d(i, n+1) + d(n+1, i).$$

Again by triangle inequality, $w(T_n^*) \leq w(T_{n+1}^*)$, and $d(i, n+1) + d(n+1, i) \leq w(T_{n+1}^*)$, which concludes the proof. □

We remark that the above upper bound of 2 on the approximation ratio is tight, even if we use Best Insertion instead of inserting the new vertex between an arbitrarily chosen pair of consecutive vertices.

Theorem 4.14. Min ATSP− *is approximable within ratio 2.*

Proof. The obvious idea is to skip the deleted node in the new tour, while visiting the remaining nodes in the same order. Thus, if i and j are respectively the nodes preceding and following $n+1$ in the tour T_{n+1}^*, we obtain a tour T such that

$$w(T) = w(T_{n+1}^*) + d(i, j) - d(i, n+1) - d(n+1, j). \tag{4.7}$$

Consider an optimum solution T_n^* of the modified instance I_n, and the node l that is consecutive to i in this solution. Since inserting $n+1$ between i and l would yield a feasible solution to I_{n+1}, we get, using triangle inequality:

$$\begin{aligned} w(T_{n+1}^*) &\leq w(T_n^*) + d(i, n+1) + d(n+1, l) - d(i, l) \\ &\leq w(T_n^*) + d(i, n+1) + d(n+1, i). \end{aligned}$$

By substituting in (4.7) and using triangle inequality again,

$$w(T) \leq w(T_n^*) + d(i, j) + d(j, i).$$

Hence, $w(T) \leq 2w(T_n^*)$. □

4.4.2. *The Maximum Traveling Salesman Problem*

4.4.2.1. *Maximum TSP*

While the typical applications of the Minimum TSP are in vehicle routing and transportation problems, the Maximum TSP has applications to DNA sequencing and data compression [25]. Like the Minimum TSP, the Maximum TSP is also NP-hard, but differently from what happens for the Minimum TSP, it is approximable within a constant factor even when the distance matrix can be completely arbitrary. In the static setting, the best known result for Max TSP is a 0.6-approximation algorithm due to Kosaraju et al. [25]. Once again, the knowledge of an optimum solution to the initial instance is useful, as the reoptimization problem under insertion of a vertex can be approximated within a ratio of 0.66 (for large enough n), as we show next.

Theorem 4.15. Max TSP+ *is approximable within ratio* $(2/3) \cdot (1 - 1/n)$.

Proof. Let i and j be such that $(i, n+1)$ and $(n+1, j)$ belong to T^*_{n+1}. The algorithm is the following:

(1) Apply Best Insertion to T^*_n to get a tour T_1;
(2) Find a maximum cycle cover $\mathcal{C} = (C_0, \ldots, C_l)$ on I_{n+1} such that:
    (a) $(i, n+1)$ and $(n+1, j)$ belong to C_0;
    (b) $|C_0| \geq 4$;
(3) Remove the minimum-weight arc of each cycle of $\mathcal{C}$ and patch the paths obtained to get a tour T_2;
(4) Select the best solution between T_1 and T_2.

Note that Step 2 can be implemented in polynomial time as follows: We replace $d(i, n+1)$ and $d(n+1, j)$ by a large weight M, and $d(j, i)$ by $-M$ (we do not know i and j, but we can try each possible pair of vertices and return the best tour constructed by the algorithm). Hence, this cycle cover will contain $(i, n+1)$ and $(n+1, j)$ but not (j, i), meaning that the cycle containing $n+1$ will have at least 4 vertices.

Let $a := d(i, n+1) + d(n+1, j)$. Clearly, $w(T^*_{n+1}) \leq w(T^*_n) + a$. Now, by inserting $n+1$ in each possible position, we get

$$w(T_1) \geq (1 - 1/n) w(T^*_n) \geq (1 - 1/n)(w(T^*_{n+1}) - a).$$

Since C_0 has size at least 4, the minimum-weight arc of C_0 has cost at most $(w(C_0) - a)/2$. Since each cycle has size at least 2, we get a tour T_2

of value:

$$\begin{aligned} w(T_2) &\geq w(\mathcal{C}) - \frac{w(C_0) - a}{2} - \frac{w(\mathcal{C}) - w(C_0)}{2} \\ &= \frac{w(\mathcal{C}) + a}{2} \geq \frac{w(T^*_{n+1}) + a}{2}. \end{aligned}$$

Combining the two bounds for T_1 and T_2, we get a solution which is $(2/3) \cdot (1 - 1/n)$-approximate. □

The above upper bound can be improved to 0.8 when the distance matrix is known to be symmetric [5].

4.4.2.2. *Maximum Metric TSP*

The usual Maximum TSP problem does not admit a polynomial-time approximation scheme, that is, there exists a constant c such that it is NP-hard to approximate the problem within a factor better than c. This result extends also to the Maximum Metric TSP [29]. The best known approximation for the Maximum Metric TSP is a randomized algorithm with an approximation guarantee of 7/8 [21].

By contrast, in the reoptimization of Max MTSP under insertion of a vertex, the Best Insertion algorithm turns out to be a very good strategy: It is asymptotically optimum. In particular, the following holds:

Theorem 4.16 ([5]). Max MTSP+ *is approximable within ratio* $1 - O(n^{-1/2})$.

Using the above result one can easily prove that Max MTSP+ admits a polynomial-time approximation scheme: If the desired approximation guarantee is $1 - \epsilon$, for some $\epsilon > 0$, just solve by enumeration the instances with $O(1/\epsilon^2)$ nodes, and use the result above for the other instances.

4.4.3. *The Minimum Latency Problem*

Although superficially similar to the Minimum Metric TSP, the Minimum Latency Problem appears to be more difficult to solve. For example, in the special case when the metric is induced by a weighted tree, the MLP is NP-hard [35] while the Metric TSP is trivial. One of the difficulties in the MLP is that local changes in the input can influence the global shape of the optimum solution. Thus, it is interesting to notice that despite this fact, reoptimization still helps. In fact, the best known approximation so far for the static version of the MLP gives a factor of 3.59 and is achieved via a sophisticated algorithm due to Chaudhuri et al. [10], while it is possible to give a very simple 3-approximation for MLP+, as we show in the next theorem.

Theorem 4.17. MLP+ *is approximable within ratio* 3.

Proof. We consider the *Insert Last* algorithm that inserts the new node $n+1$ at the "end" of the tour, that is, just before node 1. Without loss of generality, let $T_n^* = \{(1,2),(2,3),\dots,(n-1,n)\}$ be the optimal tour for the initial instance I_n (that is, the kth node to be visited is k). Let T_{n+1}^* be the optimal tour for the modified instance I_{n+1}. Clearly $\ell(T_{n+1}^*) \geq \ell(T_n^*)$ since relaxing the condition that node $n+1$ must be visited cannot raise the overall latency.

The quantity $\ell(T_n^*)$ can be expressed as $\sum_{i=1}^{n} t_i$, where for $i = 1,\dots,n$, $t_i = \sum_{j=1}^{i-1} d(j,j+1)$ can be interpreted as the "time" at which node i is first visited in the tour T_n^*.

In the solution constructed by Insert Last, the time at which each node $i \neq n+1$ is visited is the same as in the original tour (t_i), while $t_{n+1} = t_n + d(n,n+1)$. The latency of the solution is thus $\sum_{i=1}^{n+1} t_i = \sum_{i=1}^{n} t_i + t_n + d(n,n+1) \leq 2\ell(T_n^*) + \ell(T_{n+1}^*) \leq 3\ell(T_{n+1}^*)$, where we have used $\ell(T_{n+1}^*) \geq d(n,n+1)$ (any feasible tour must include a subpath from n to $n+1$ or vice versa). □

4.5. Concluding Remarks

In this chapter we have seen how the reoptimization model can often be applied to NP-hard combinatorial problems in order to obtain algorithms with approximation guarantees that improve upon the trivial approach of computing an approximate solution from scratch.

Apart from designing algorithms with good approximation guarantees for reoptimization problems – and from obtaining sharper negative results – there are some general open directions in the area. One is to investigate the more general issue of maintaining an approximate solution under input modifications. In our model we assumed that an optimal solution was available for the instance prior to the modification, but it is natural to relax this constraint by assuming only an approximate solution instead. In some cases the analysis of the reoptimization algorithm can be carried out in a similar way even with such a relaxed assumption, but this needs not be always true.

Another general question is that of studying the interplay between running time, approximation guarantee, and amount of data perturbation. If we devote enough running time (for example, exponential time for problems in NPO) to the solution of an instance, we can find an optimal solution independently of the amount of perturbation. On the other hand we saw that for many problems it is possible to find in polynomial time an almost optimal solution for any slightly perturbed instance. One could expect that there might be a general trade-off between the amount of data perturbation and the running time needed the reconstruct a solution of a given quality. It would be interesting to identify problems for which such trade-offs are possible.

References

[1] C. Archetti, L. Bertazzi, and M. G. Speranza, Reoptimizing the traveling salesman problem, *Networks.* **42**(3), 154–159, (2003).

[2] C. Archetti, L. Bertazzi, and M. G. Speranza. Reoptimizing the 0–1 knapsack problem. Technical Report 267, Department of Quantitative Methods, University of Brescia, (2006).

[3] T. Asano, K. Hori, T. Ono, and T. Hirata. A theoretical framework of hybrid approaches to MAX SAT. In *Proc. 8th Int. Symp. on Algorithms and Computation*, pp. 153–162, (1997).

[4] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and approximation – Combinatorial optimization problems and their approximability properties*, Springer, Berlin, (1999).

[5] G. Ausiello, B. Escoffier, J. Monnot, and V. T. Paschos. Reoptimization of minimum and maximum traveling salesman's tours. In *Proc. 10th Scandinavian Workshop on Algorithm Theory*, pp. 196–207, (2006).

[6] M. Bartusch, R. Möhring, and F. J. Radermacher, Scheduling project networks with resource constraints and time windows, *Ann. Oper. Res..* **16**, 201–240, (1988).

[7] M. Bartusch, R. Möhring, and F. J. Radermacher, A conceptional outline of a DSS for scheduling problems in the building industry, *Decision Support Systems.* **5**, 321–344, (1989).

[8] H.-J. Böckenhauer, L. Forlizzi, J. Hromkovic, J. Kneis, J. Kupke, G. Proietti, and P. Widmayer. Reusing optimal TSP solutions for locally modified input instances. In *Proc. 4th IFIP Int. Conf. on Theoretical Computer Science*, pp. 251–270, (2006).

[9] H.-J. Böckenhauer, J. Hromkovic, T. Mömke, and P. Widmayer. On the hardness of reoptimization. In *Proc. 34th Conf. on Current Trends in Theory and Practice of Computer Science*, pp. 50–65, (2008).

[10] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. In *Proc. 44th Symp. on Foundations of Computer Sci-*

ence, pp. 36–45, (2003).

[11] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, (1976).

[12] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig, Sparsification-a technique for speeding up dynamic graph algorithms, *J. ACM.* **44**(5), 669–696, (1997).

[13] B. Escoffier, M. Milanic, and V. T. Paschos, *Simple and fast reoptimizations for the Steiner tree problem*, Cahier du LAMSADE 245, LAMSADE, Université Paris-Dauphine, (2007).

[14] S. Even and H. Gazit, Updating distances in dynamic graphs, *Methods Oper. Res.* **49**, 371–387, (1985).

[15] U. Feige and M. Singh. Improved approximation ratios for traveling salesperson tours and paths in directed graphs. In *Proc. 10th Int. Workshop on Approximation, Randomization, and Combinatorial Optimization*, pp. 104–118, (2007).

[16] G. N. Frederickson, Data structures for on-line updating of minimum spanning trees, with applications, *SIAM J. Comput..* **14**(4), 781–798, (1985).

[17] A. M. Frieze, G. Galbiati, and F. Maffioli, On the worst-case performance of some algorithms for the asymmetric traveling salesman problem, *Networks.* **12**(1), 23–39, (1982).

[18] C. Gröpl, S. Hougardy, T. Nierhof, and H. Prömel. Approximation algorithms for the Steiner tree problem in graphs. In eds. D.-Z. Du and X. Cheng, *Steiner Trees in Industry*, pp. 235–279. Kluwer Academic Publishers, Dordrecht, (2000).

[19] G. Gutin and A. P. Punnen, Eds., *The Traveling Salesman Problem and its Variations*. Kluwer, Dordrecht, (2002).

[20] N. G. Hall and M. E. Posner, Sensitivity analysis for scheduling problems, *J. Sched..* **7**(1), 49–83, (2004).

[21] R. Hassin and S. Rubinstein, A 7/8-approximation algorithm for metric Max TSP, *Inform. Process. Lett..* **81**(5), 247–251, (2002).

[22] M. R. Henzinger and V. King, Maintaining minimum spanning forests in dynamic graphs, *SIAM J. Comput..* **31**(2), 367–374, (2001).

[23] O. H. Ibarra and C. E. Kim, Fast approximation algorithms for the knapsack and sum of subset problems, *J. ACM.* **22**(4), 463–468, (1975).

[24] D. S. Johnson, Approximation algorithms for combinatorial problems, *J. Comput. Systems Sci..* **9**, 256–278, (1974).

[25] S. R. Kosaraju, J. K. Park, and C. Stein. Long tours and short superstrings. In *Proc. 35th Symp. on Foundations of Computer Science*, pp. 166–177, (1994).

[26] E. L. Lawler, J. K. Lenstra, A. Rinnoy Kan, and D. B. Shymois, Eds., *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* Wiley, Chichester, (1985).

[27] J. M. Lewis and M. Yannakakis, The node-deletion problem for hereditary properties is NP-complete, *J. Comput. Systems Sci..* **20**(2), 219–230, (1980).

[28] C. H. Papadimitriou and K. Steiglitz, On the complexity of local search for

the traveling salesman problem, *SIAM J. Comput.*. **6**(1), 76–83, (1977).

[29] C. H. Papadimitriou and M. Yannakakis, The traveling salesman problem with distances one and two, *Math.Oper. Res.*. **18**(1), 1–11, (1993).

[30] A. Paz and S. Moran, Non-deterministic polynomial optimization problems and their approximations, *Theoret. Comput. Sci.*. **15**, 251–277, (1981).

[31] G. Robins and A. Zelikovsky, Tighter bounds for graph Steiner tree approximation, *SIAM J. Discrete Math.*. **19**(1), 122–134, (2005).

[32] H. Rohnert. A dynamization of the all-pairs least cost problem. In *Proc. 2nd Symp. on Theoretical Aspects of Computer Science*, pp. 279–286, (1985).

[33] S. Sahni and T. F. Gonzalez, P-complete approximation problems, *J. ACM*. **23**(3), 555–565, (1976).

[34] M. W. Schäffter, Scheduling with forbidden sets, *Discrete Appl. Math.*. **72** (1-2), 155–166, (1997).

[35] R. Sitters. The minimum latency problem is NP-hard for weighted trees. In *Proc. 9th Integer Programming and Combinatorial Optimization Conf.*, pp. 230–239, (2002).

[36] V. V. Vazirani, *Approximation Algorithms*. Springer, Berlin, (2001).

[37] D. Zuckerman, Linear degree extractors and the inapproximability of max clique and chromatic number, *Theory Comput.*. **3**(1), 103–128, (2007).