Algorithmica (2018) 80:576–607
https://doi.org/10.1007/s00453-017-0274-8

A Theory and Algorithms for Combinatorial Reoptimization

Baruch Schieber[1] · **Hadas Shachnai**[2] · **Gal Tamir**[2] · **Tami Tamir**[3]

Received: 21 December 2015 / Accepted: 4 January 2017 / Published online: 27 January 2017

Abstract Many real-life applications involve systems that change dynamically over time. Thus, throughout the continuous operation of such a system, it is required to compute solutions for new problem instances, derived from previous instances. Since the transition from one solution to another incurs some cost, a natural goal is to have the solution for the new instance close to the original one (under a certain distance measure). In this paper we develop a general framework for combinatorial repotimization, encompassing classical objective functions as well as the goal of minimizing the transition cost from one solution to the other. Formally, we say that $\mathcal{A}$ is an (r, ρ)-reapproximation algorithm if it achieves a ρ-approximation for the optimization problem, while paying a transition cost that is at most r times the minimum required for solving the problem optimally. Using our model we derive reoptimization and

A preliminary version of this paper appeared in the Proceedings of the 10th Latin American Symposium on Theoretical Informatics (LATIN), Arequipa, April 2012. Research supported by the Israel Science Foundation (Grant Number 1574/10), and by the Ministry of Trade and Industry MAGNET program through the NEGEV Consortium.

✉ Hadas Shachnai
hadas@cs.technion.ac.il

Baruch Schieber
sbar@us.ibm.com

Gal Tamir
galtamir@cs.technion.ac.il

Tami Tamir
tami@idc.ac.il

1 IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598, USA

2 Department of Computer Science, Technion, 3200003 Haifa, Israel

3 School of Computer Science, The Interdisciplinary Center, Herzliya, Israel

reapproximation algorithms for several classes of combinatorial reoptimization problems. This includes a fully polynomial time $(1+\varepsilon_1, 1+\varepsilon_2)$-reapproximation scheme for the wide class of DP-benevolent problems introduced by Woeginger (Proceedings of Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 1999), a $(1, 3)$-reapproximation algorithm for the metric k-Center problem, and $(1, 1)$-reoptimization algorithm for polynomially solvable subset-selection problems. Thus, we distinguish here for the first time between classes of reoptimization problems by their hardness status with respect to the objective of minimizing transition costs, while guaranteeing a good approximation for the underlying optimization problem.

Keywords Combinatorial reoptimization · Reapproximation · k-Center · Subset selection · Approximation schemes

1 Introduction

Traditional combinatorial optimization problems require finding solutions for a single instance. However, many of the real-life scenarios motivating these problems involve systems that change dynamically over time. Thus, throughout the continuous operation of such a system, it is required to compute solutions for new problem instances, derived from previous instances. Moreover, since there is some cost associated with the transition from one solution to another, a natural goal is to have the solution for the new instance *close* to the original one (under certain distance measure).

For example, in a *video-on-demand* (*VoD*) system, such as Hulu [33] or Netflix [48], movie popularities tend to change frequently. In order to satisfy the new client requests, the content of the storage system needs to be modified. The new storage allocation needs to reflect the current demand; also, due to the cost of file migrations, this should be achieved by using a minimum number of reassignments of file copies to servers. In communication networks, such as Multiprotocol Label Switching (MPLS) [12] or Asynchronous Transfer Mode (ATM) [47], the set of demands to connect sources to destinations changes over time. Rerouting incurs the cost of acquiring additional bandwidth for some links that were not used in the previous routing. The goal is to optimally handle new demands while minimizing the total cost incurred due to these routing changes. In production planning, due to unanticipated changes in the timetables for task processing or out-of-order machines, the production schedule needs to be modified. Rescheduling tasks is costly (due to relocation overhead and machine setup times). The goal is to find a new feasible schedule, which is as close as possible to the previous one. In crew scheduling, due to unexpected changes in the timetables of crew members, the crew assignment needs to be updated. Since this may require other members to be scheduled at undesirable times (incurring some compensation for these members), the goal is to find a new feasible schedule, which is as close as possible to the previous one.

Thus, solving a *reoptimization* problem involves two challenges:

(i) *Computing* an optimal (or close to the optimal) solution for the new instance.
(ii) Efficiently *converting* the current solution to the new one.

Each of these challenges, even when considered alone, gives rise to many theoretical and practical questions. Obviously, combining the two challenges is an important goal, which naturally shows up in numerous applications (see Sect. 1.1).

In this paper we develop a general framework for combinatorial repotimization, encompassing objective functions that combine the two above challenges. Our study differs from previous work in two aspects. One aspect is in the generality of our approach. To the best of our knowledge, previous studies consider specific reoptimization problems. Consequently, known algorithms rely on techniques tailored for these problems (see Sect. 1.2). We are not aware of general theoretical results or algorithmic techniques developed for certain *classes* of combinatorial reoptimization problems. This is the focus of our work. The other aspect is our performance measure, which combines two objective functions.[1] The vast majority of previous research refers to the computational complexity of solving an optimization problem once an initial input has been modified, i.e., the first of the above-mentioned challenges (see, e.g., the results for reoptimization of the *traveling salesman problem* (*TSP*) [6,15]).

One consequence of these differences between our study and previous work is in the spirit of our results. Indeed, in solving a reoptimization problem, we usually expect that starting off with a solution for an initial instance of a problem should help us obtain a solution at least as good (in terms of approximation ratio) for a modified instance, with better running time. Yet, our results show that reoptimization with transition costs may be harder than solving the underlying optimization problem. This is inherent in the reoptimization problems motivating our study, rather than the model we use to tackle them. Indeed, due to the transition costs, we seek for the modified instance an efficient solution that can be reached at low cost. In that sense, the given initial solution plays a restrictive role, rather than serve as guidance to the algorithm.[2]

1.1 Applications

In this section we describe a few of the many applications where reoptimization problems naturally show up.

1.1.1 Storage Systems in Video on Demand Services

In a VoD system, such as Hulu [33] or Netflix [48], clients are connected through a network to a set of servers that hold a large library of media objects (movies). The transmission of a movie to a client requires the allocation of unit load capacity (or, a *data stream*) on a server that holds a copy of the movie. The *reconfiguration* problem described below is a reoptimization problem arising due to the frequent changes in movie popularity.

The system consists of M movies and N servers. Each server, j, is characterized by (i) its storage capacity, S_j, the number of files that can reside on it, and (ii) its load capacity, L_j, the number of data streams that can be read simultaneously

[1] As discussed in Sect. 1.2, this is different than multi-objective optimization.

[2] This is similar in nature, e.g., to *incremental optimization* studied in [43].

Table 1 The assignment and broadcast matrices of the placement before (left) and after (right) the popularity change

A_{I_0}	1	2
1	1	0
2	1	1
3	1	0
4	0	1
5	0	1
6	0	1

B_{I_0}	1	2
1	1	0
2	8	4
3	1	0
4	0	3
5	0	1
6	0	2

A_I	1	2
1	1	0
2	0	1
3	1	0
4	0	1
5	1	1
6	0	1

B_I	1	2
1	2	0
2	0	3
3	1	0
4	0	3
5	7	2
6	0	2

from that server. Each movie i is associated with *broadcast demand* D_i^0, based on its popularity, where $\sum_{i=1}^{M} D_i^0 = \sum_{j=1}^{N} L_j$ is the total load capacity of the system. The *data placement problem* is to determine a placement of movie file copies on the servers (the placement matrix, A), and the amount of load capacity assigned to each file copy (the broadcast matrix, B), so as to maximize the total amount of broadcast demand satisfied by the system. Under certain conditions, it is known that a *perfect placement*, where each movie is allocated exactly D_i^0 broadcasts, always exists [52].

In the *reconfiguration problem*, the demand vector is changing. The goal is to modify the data placement to a perfect placement for the new demand. This involves replications and deletions of files. File replications incur significant cost as they require bandwidth and storage capacity, at the source, as well as the destination server. We denote by $s_{i,j}$ the cost of adding a copy of movie i on server j. Using our general notation for a reoptimization problem, the storage allocation reoptimization problem is defined as follows.

Let I_0 be an input for the problem, given by the number and capacities of servers, and the number and broadcast demand of movies. Let C_{I_0} be a perfect assignment for I_0. That is, C_{I_0} consists of two (assignment and broadcast) matrices, A_{I_0}, B_{I_0}, such that each movie is allocated broadcasts that perfectly fit its demand. Let I be an input derived from I_0 by changing the broadcast demand of the movies. Let C_I be a solution for I, let A_I, B_I be the corresponding assignment and broadcast matrices. The transition cost is the total cost of file replications required when moving from assignment A_{I_0} to assignment A_I. Formally, $cost(C_{I_0}, C_I) = \sum_{i,j} s_{i,j} \cdot max(A_I[i, j] - A_{I_0}[i, j], 0)$. The objective of the reoptimization problem is to find $C_I \in \mathcal{C}_{\mathcal{I}}$ such that $val(C_I)$ is maximal (i.e., a perfect placement for I), and $cost(C_{I_0}, C_I))$ is minimal.

Example: Consider a system of two servers that hold 6 movies. Both servers have the same load capacity $L_1 = L_2 = 10$, while the storage capacities are $S_1 = 3, S_2 = 4$. The demand vector is $D^0 = \langle 1, 12, 1, 3, 1, 2\rangle$. Assume that the demand vector is changed to $D = \langle 2, 3, 1, 3, 9, 2\rangle$. It is possible to satisfy the new demands by adding (and deleting) a copy of one file. The new placement is perfect for the new demand vector. The corresponding assignment and broadcast matrices are given in Table 1. The reconfiguration cost is $s_{5,1}$.

1.1.2 Communication Services and Other Network Problems

In communication networks such as Multiprotocol Label Switching (MPLS) [12] or Asynchronous Transfer Mode (ATM) [47], data is directed and carried along high-performance communication links. A network is characterized by its topology (N routers and M links). Each link e has a given capacity $c(e)$ specifying the total bandwidth of packets that can be carried on e. The network receives bandwidth demands $d_{i,j}$ between n source-sink router pairs (i, j) for which it generates a routing scheme, using different performance measures (see, e.g., [12]). Due to the dynamic nature of the network operation, the end-to-end demand is changing over time. The reoptimization problem is to compute a new routing scheme achieving good performance relative to the new demands. There is a cost associated with bandwidth upgrades and packets rerouting. Thus, it is desirable to minimize the transition cost between the two routing schemes.

1.1.3 Production Planning

Tasks are processed by machines. The production schedule needs to be modified due to unanticipated changes in the timetables for task processing, out-of-order machines, etc. Rescheduling tasks is costly (due to relocation overhead and machine setup times). The goal is to find a new feasible schedule, which is as close as possible to the previous one.

1.1.4 Vehicle Routing

Customers are serviced using a fleet of vehicles The goal is to find an updated service schedule which is as close as possible to the original. Changes include: reschedules of pick-up and delivery times for some customers, a vehicle breaks down, or changes in customer demands.

1.1.5 Crew Scheduling

Crews are assigned to operate transportation systems, such as trains or aircrafts. Due to unexpected changes in the timetables of crew members, the crew assignment needs to be updated. Since this may require other members to be scheduled at undesirable times (incurring some compensation for these members), the goal is to find a new feasible schedule, which is as close as possible to the previous one.

1.1.6 Facility Location

Facilities need to be opened in order to minimize transportation costs [20]. Possible changes include: addition of a new facility, increase or decrease in facility capacity or in client demands.

1.2 Related Work

The study of reoptimization problems started with the analysis of *dynamic graph* problems (see e.g. [22,55] and a survey in [18]). These works focus on developing data structures which support updates and query operations on graphs. Reoptimization algorithms were developed also for some classic problems on graphs, such as shortest-paths [46,49] and the minimum spanning tree [2]. Since all of these problems can be solved in polynomial time, even with no initial solution, the goal is to compute an optimal solution efficiently, based on the local nature of the updates and on properties of optimal solutions.

A different line of research deals with the computation of a good solution for an NP-hard problem, given an optimal solution for a close instance. In general, NP-hardness of a problem implies that a solution for a locally modified instance cannot be found in polynomial time. However, it is an advantage to have a solution for a close instance, compared to not knowing it. In particular, for some problems, it is possible to develop algorithms guaranteeing better approximation ratio for the reoptimization version than for the original problem. Among the problems studied in this setting are (i) The Traveling Salesman problem (TSP), in which the modification is a change in the cost of exactly one edge [4,6,15], (ii) The Steiner Tree problem in weighted graphs, where the modifications considered are insertion/deletion of a vertex, or increase/decrease in the weight of a single edge [16,23,38]. (iii) The Knapsack problem, in which the modification is an addition of a single element [5], and (iv) Pattern Matching problems: for example, reoptimization of the shortest common superstring problem [13], in which a single string is added to the input.

A survey of other research in this direction is given in [7]. It is important to note that, unlike the present paper, in all of the above works, the goal is to compute an optimal (or approximate) solution for the modified instance. The resulting solution may be significantly different from the original one, since there is no cost associated with the transition among solutions. Reoptimization is also used as a technique in local-search algorithms. For example, in [58], reoptimization is used for efficient multiple sequence alignment—a fundamental problem in bioinformatics and computational biology. In [54], reoptimization in used to improve the performance of a branch-and-bound algorithm for the Knapsack problem.

Other related works consider *multi-objective* optimization problems. In these problems, there are several weight functions associated with the input elements. The goal is to find a solution whose quality is measured with respect to a combination of these weights (see e.g., [14,29,50]). Indeed, in alternative formulation of these problems, we can view one of the weight functions as the transition cost from one solution to another. Thus, known results for multi-objective optimization carry over to *budgeted* reoptimization. Such a model was studied, e.g., in [41] (see also [42] and the references therein). However, in this paper we focus on minimizing the total transition cost required for achieving a good solution for the underlying optimization problem, rather than efficiently using a given budget. Indeed, in solving our reoptimization problems, it is natural to consider applying binary search, to find the reoptimization cost (i.e., the *budget*), and then use a multi-objective optimization algorithm as a black-box. However (as we show in Theorem 1), this cost cannot be found in polynomial time,

unless $P = NP$. This leads us to use a different approach (and alternative measures) for obtaining reapproximation algorithms.

The application of storage management has been extensively studied (see, e.g., [27,35,51,57] and a comprehensive survey in [36]). The dynamic problem considers the more realistic model, where file popularities change over time. Several variants of reoptimization problems with different settings were analyzed. In some works, the final configuration is given as part of the input, and the only goal is to minimize the transition costs [52]. In other works, the reconfiguration should be performed within limited time, and by using limited bandwidth [37]. There has been some other work on reconfiguration of data placement, in which heuristic solutions were investigated through experimental studies (e.g., [17]). Communication networks define a wide research area (see survey in [40]). In particular, routing problems were extensively studied even in a static environment [26]. Rerouting problems were considered in both centralized and distributed systems (e.g., [8,12]).

The k-Center problem has been widely studied (see, e.g., [3,21,31,34] and the references therein). The problem is hard to approximate within ratio better than 2, in any metric space, unless $P = NP$ (see, e.g., [30]). The best possible ratio of 2 was obtained in [21,31]. The k-Supplier problem is a generalization of the k-Center problem. On general metrics k-Supplier admits a 3-approximation, which was shown to be tight [32] (see also [24,45]). To the best of our knowledge, the reoptimization versions of these problems are studied here for the first time.

Since its introduction as an extended abstract [53], our framework, of analyzing a reoptimization scenario as a combination of computation of and a transition to a new configuration, was adopted for rescheduling [9,10] and storage reallocation [11]. A different approach in which the reoptimization cost is studied as a parameterized complexity optimization problem is suggested in [1].

1.3 Our Results

We develop (in Sect. 2) a general model for combinatorial reoptimization that captures many real-life scenarios. Using our model, we derive reoptimization and reapproximation algorithms for several important classes of optimization problems. In particular, we consider (in Sect. 3) the class of DP-benevolent problems introduced by Woeginger [56]. The paper [56] gives an elaborate characterization of these problems, which is used to show that any problem in this class admits a *fully polynomial time approximation scheme* (*FPTAS*).[3]

We introduce (in Definition 5) the notion of *fully polynomial time reapproximation scheme (FPTRS)*. Informally, given an optimization problem Π, such a scheme takes as input parameters $\varepsilon_1, \varepsilon_2 > 0$ and outputs a solution that approximates simultaneously the minimum reoptimization cost (within factor $1 + \varepsilon_1$) and the objective function for Π (within factor $1 + \varepsilon_2$), in time that is polynomial in the input size and in $1/\varepsilon_1$, $1/\varepsilon_2$. We show that the reoptimization variants of a non-trivial subclass of DP-benevolent

[3] A key property is that each problem in the class can be formulated via a dynamic program of certain structure, and the involved costs and transition functions satisfy certain arithmetic and structural conditions.

problems admit fully polynomial time $(1+\varepsilon_1, 1+\varepsilon_2)$-reapproximation schemes, for any $\varepsilon_1, \varepsilon_2 > 0$. We note that this is the best possible, unless $P = NP$.

In Sect. 4 we give a $(1, 3)$-reapproximation algorithm for reoptimization of the metric k-Center problem. The same ratio holds for reoptimization of the metric k-Supplier problem. In Sect. 5, we show that for any subset-selection problem Π over n elements, which can be optimally solved in time $T(n)$, there is a $(1, 1)$-reoptimization algorithm for the reoptimization version of Π, whose running time is $T(n')$, where n' is the size of the modified input. This yields a polynomial time $(1, 1)$-reoptimization algorithm for a large set of polynomially solvable problems, as well as for problems that are fixed parameter tractable.[4]

Thus, we distinguish here for the first time between classes of reoptimization problems by their hardness status with respect to the objective of minimizing transition costs, while guaranteeing a good approximation for the underlying optimization problem.

We conclude (in Sect. 6) with a discussion of possible directions for future work.

2 Combinatorial Reoptimization: Definitions and Notation

In the following we formally define our model for combinatorial reoptimization. Let Π be an optimization problem and I_0 an input for Π; also, let $\mathcal{C}_{I_0} = \{C^1_{I_0}, C^2_{I_0}, \ldots\}$ be the set of configurations corresponding to the solution space of Π for the input I_0. Each configuration $C^j_{I_0} \in \mathcal{C}_{I_0}$ has some value $val(C^j_{I_0})$. In the reoptimization problem, $R(\Pi)$, we are given a configuration $C^j_{I_0} \in \mathcal{C}_{I_0}$ of an initial instance I_0, and a new instance I derived from I_0 by admissible operations, e.g., addition or removal of elements, changes in element parameters etc. For any element $i \in I$ and configuration $C^k_I \in \mathcal{C}_{\mathcal{I}}$, we are given the *transition cost* of i when moving from the initial configuration $C^j_{I_0}$ to the feasible configuration C^k_I of the new instance. We denote this transition cost by $\delta(i, C^j_{I_0}, C^k_I)$. Practically, the transition cost of i is not given as a function of two configurations, but as a function of i's state in the initial configuration and its possible states in any new configuration. This representation keeps the input description more compact. The primary goal is to find an optimal solution for I. Among all configurations with an optimal $val(C^k_I)$ value, we are looking for a configuration C^*_I for which the total transition cost, given by $\sum_{i\in I} \delta(i, C^j_{I_0}, C^*_I)$ is minimized.

For example, assume that Π is the *minimum spanning tree (MST)* problem. Let $G_0 = (V_0, E_0)$ be a weighted graph, and let $T_0 = (V_0, E_{T_0})$ be an MST of G_0. Let $G = (V, E)$ be a graph derived from G_0 by adding or removing vertices and/or edges, and by changing the weights of edges. Let $T = (V, E_T)$ be an MST of G. For every edge $e \in E_T \backslash E_{T_0}$, we are given the cost $\delta_{add}(e)$ of including e in the new solution, and for every edge $e \in E \cap (E_{T_0} \backslash E_T)$ we are given the cost $\delta_{rem}(e)$ of removing e from the solution. The goal in the reoptimization problem $R(MST)$ is to find an MST of G with minimal total transition cost. As we show in Sect. 5, $R(MST)$ belongs to a class of subset-selection problems that are polynomially solvable.

[4] see, e.g., [19] for the theory of fixed-parameter algorithms and parameterized complexity.

The input for the reoptimization problem, I_R, contains both the new instance I and the transition costs δ (that may be encoded in various ways). Note that I_R does not include the initial configuration—since apart from determining the transition costs, it has no effect on the reoptimization problem. Thus, our results hold for *any* configuration $C^j_{I_0} \in \mathcal{C}_{I_0}$, which may not be a 'good' solution for I_0. Moreover, the new instance I can result from any admissible changes in I_0.[5]

2.1 Approximate Reoptimization

When the problem Π is NP-hard, or when the reoptimization problem $R(\Pi)$ is NP-hard,[6] we consider approximate solutions. The goal is to find a good solution for the new instance, while keeping a low transition cost from the initial configuration to the new one. Formally, denote by $\mathcal{O}(I)$ the optimal solution value of $\Pi(I)$. A configuration $C^k_I \in \mathcal{C}_\mathcal{I}$ yields a ρ-approximation for $\Pi(I)$, for some $\rho \geq 1$, if its value is within ratio ρ from $\mathcal{O}(I)$. That is, if Π is a minimization problem then $val(C^k_I) \leq \rho\mathcal{O}(I)$; if Π is a maximization problem then $val(C^k_I) \geq \frac{1}{\rho}\mathcal{O}(I)$. Given a reoptimization instance I_R, for any $\rho \geq 1$, denote by $\mathcal{O}_R(I_R, \rho)$ the minimal possible transition cost to a configuration $C^k_I \in \mathcal{C}_\mathcal{I}$ that yields a ρ-approximation for $\mathcal{O}(I)$, and by $\mathcal{O}_R(I_R)$ the minimal transition cost to an optimal configuration of I.

Ideally, in solving a reoptimization problem, we would like to find a solution whose total transition cost is close to the best possible, among all solutions with a given approximation guarantee, $\rho \geq 1$, for the underlying optimization problem. Formally,

Definition 1 An algorithm $\mathcal{A}$ yields a *strong* (r, ρ)*-reapproximation* for $R(\Pi)$, for $\rho, r \geq 1$, if, for any reoptimization input I_R, $\mathcal{A}$ achieves a ρ-approximation for $\mathcal{O}(I)$, with transition cost at most $r \cdot \mathcal{O}_R(I_R, \rho)$.

Unfortunately, for many NP-hard optimization problems, finding a strong (r, ρ)-reapproximation is NP-hard, for any $r, \rho \geq 1$. This follows from the fact that it is NP-hard to determine whether the initial configuration is a ρ-approximation for the optimal one (in which case, the transition cost to a ρ-approximate solution is equal to zero). We demonstrate this hardness for the Knapsack problem.

Theorem 1 *For any* $r, \rho \geq 1$*, obtaining a strong* (r, ρ)*-reapproximation for Knapsack is NP-hard.*

Proof We show a reduction from the decision version of Knapsack:
Instance: A set of items A, each having a size and a profit, a knapsack of size B, and a value P.
Question: Is it possible to pack a subset having total value larger than P?

Given an instance of the above decision problem, assume there exists an (r, ρ)-reapproximation algorithm for Knapsack, for some $\rho \geq 1$, and define the following instance for the reoptimization problem.

[5] Some earlier results rely on the assumption that I results from a slight modification of I_0; see Sect. 1.2).

[6] As we show below, it may be that none, both, or only $R(\Pi)$ are NP-hard.

(i) The set of items consists of A and an additional item $\hat{i}$ having size B and value P/ρ. All items have transition cost 1—charged if added to or removed from the knapsack.
(ii) In the initial configuration, item $\hat{i}$ is in the knapsack.

□

Claim 2 *The (r, ρ)-reapproximation algorithm leaves the item $\hat{i}$ in the knapsack if and only if the answer to the decision problem is 'NO'.*

Proof If 'NO', then it is impossible to pack a subset of A having total value larger than P. Therefore, packing item $\hat{i}$ is a ρ-approximation. Thus, there exists a ρ-approximate solution for the reoptimization problem, whose transition cost is 0. For any r, the (r, ρ)-reapproximation algorithm must incur transition cost 0. The only possible packing of profit at least P/ρ and transition cost 0 is a packing of item $\hat{i}$.

If 'YES', then there exists a subset having a total value larger than P. Thus, no ρ-approximate solution packs item $\hat{i}$ – as it leaves no space for additional items. In particular, no (r, ρ)-reapproximation algorithm packs the item $\hat{i}$. □

Thus, for such problems, we use an alternative measure, which compares the total transition cost of the algorithm to the best possible, when the underlying optimization problem is solved optimally. Formally,

Definition 3 An algorithm $\mathcal{A}$ yields an (r, ρ)-reapproximation for $R(\Pi)$, for $\rho, r \geq 1$, if, for any reoptimization input I_R, $\mathcal{A}$ achieves a ρ-approximation for $\mathcal{O}(I)$, with transition cost at most $r \cdot \mathcal{O}_R(I_R)$.

Note that in both definitions, the quality of the solution is within factor ρ from the optimal solution. In a weak reapproximation, the transition cost is compared to the minimal cost required to reach an optimal solution, while in a strong reapproximation, the transition cost is compared to the minimal cost required to reach any ρ-approximate solution. Clearly, any strong (r, ρ)-reapproximation yields also an (r, ρ)-reapproximation with respect to Definition 3.

For $\rho = 1$, an $(r, 1)$-reapproximation algorithm is also called an $(r, 1)$-reoptimization algorithm (as it yields an optimal solution). In this case, Definitions 1 and 3 coincide.

Example: We demonstrate the usage of the above definitions by presenting a simple strong reapproximation algorithm for the minimum makespan problem, where the possible modification is removal of machines. Let Π be the minimum makespan scheduling problem [28] (denoted in standard scheduling notation $P||C_{max}$). An instance of Π consists of a set of n jobs and m parallel identical machines. The goal is to find an assignment of the jobs to the machines so as to minimize the latest completion time. Let I_0 be an input for the problem, and let $C^{\ell}_{I_0} = \left(C^{\ell}_{I_0}(1), \ldots, C^{\ell}_{I_0}(m)\right)$ be a solution of Π for I_0. That is, $C^{\ell}_{I_0}(i)$ specifies the set of jobs assigned to machine i, $1 \leq i \leq m$. Π is a minimization problem, and $val\left(C^{\ell}_{I_0}\right)$ is the makespan achieved under the assignment $C^{\ell}_{I_0}$. Assume further that $C^{\ell}_{I_0}$ is a *reasonable* schedule in a sense

that it is not possible to reduce the makespan by migrating the last completing job. Let I be an input derived from I_0 by removing $m_{rem} < m$ machines. For a configuration C_I^k, the transition cost of a job $j \in I$ from $C_{I_0}^\ell$ to C_I^k is equal to 0 if j remains on the same machine, and to 1 if j is assigned to different machines in $C_{I_0}^\ell$ and C_I^k.

Let S denote the set of jobs previously assigned to the m_{rem} machines that are removed from the instance. Consider a reoptimization algorithm that assigns the jobs of S subsequent to the jobs scheduled on the $m' = m - m_{rem}$ remaining machines, by *list-scheduling* (i.e., each job is assigned in turn to the least loaded machine). It is easy to verify that the resulting schedule is a possible output of list-scheduling applied to the whole instance. The transition cost of this algorithm is exactly $|S|$, which is the minimal possible transition cost from $C_{I_0}^\ell$ to any feasible configuration in $\mathcal{C}_\mathcal{I}$ (clearly, at least the set S of jobs must be reassigned). Since list-scheduling yields a $\left(2 - \frac{1}{m}\right)$-approximation for the minimum makespan problem [28] for an instance of m machines, we have that this algorithm yields a strong $\left(1, 2 - \frac{1}{m'}\right)$-reapproximation for $R(\Pi)$.

The complexity class PTAS includes all the approximation problems that admit a *polynomial time approximation scheme (PTAS)*. Specifically, $\Pi \in PTAS$ if there exists an algorithm that yields a $(1 + \varepsilon)$ approximation for Π in polynomial time (possibly exponential in $1/\varepsilon$). Generalizing this notion for the reoptimization version of the problems involves two error parameters, $\varepsilon_1, \varepsilon_2$. This leads to the following extension for the classic definition of PTAS.

Definition 4 A polynomial time reapproximation scheme (PTRS) for $R(\Pi)$ is an algorithm that gets an input for $R(\Pi)$ and the parameters $\varepsilon_1, \varepsilon_2 > 0$, and yields a $(1 + \varepsilon_1, 1 + \varepsilon_2)$-reapproximation for $R(\Pi)$, in time polynomial in $|I_R|$ (possibly exponential in $1/\varepsilon_1$ and $1/\varepsilon_2$).

Our study encompasses a non-trivial class of optimization problems that admit *fully polynomial time approximation schemes (FPTAS)*. We extend as follows the classic definition of FPTAS.

Definition 5 A fully polynomial time reapproximation scheme (FPTRS) for $R(\Pi)$ is an algorithm that gets an input for $R(\Pi)$ and the parameters $\varepsilon_1, \varepsilon_2 > 0$, and yields a $(1 + \varepsilon_1, 1 + \varepsilon_2)$-reapproximation for $R(\Pi)$, in time polynomial in $|I_R|$, $1/\varepsilon_1$ and $1/\varepsilon_2$.

2.1.1 Budgeted Reoptimization

The budgeted reoptimization problem is a single objective version of $R(\Pi)$, in which we have the constraint that the transition cost is at most m, for some budget $m \geq 0$. Thus, rather than optimizing on the transition cost, we seek the best possible solution for Π, among those of total transition cost at most m. The problem is denoted $R(\Pi, m)$, and its optimal solution for the input I_R is denoted $\mathcal{O}(I_R, m)$. Note that $\mathcal{O}(I_R, m)$ is the value of the best configuration that can be reached from the initial configuration with transition cost at most m.

Definition 6 An algorithm $\mathcal{A}$ yields a ρ-approximation for $R(\Pi, m)$ if, for any reoptimization input I_R, $\mathcal{A}$ yields a ρ-approximation for $\mathcal{O}(I_R, m)$, with transition cost at most m.

Note that the optimal value of $\mathcal{O}(I_R, m)$ may be far from $\mathcal{O}(I)$; thus, it is reasonable to evaluate algorithms for $R(\Pi, m)$ by comparison to $\mathcal{O}(I_R, m)$ and not to $\mathcal{O}(I)$.

In Sect. 3.2, we show how we can use efficient algorithms for $R(\Pi, m)$, for $m \in \mathbb{N}$, to obtain a reapproximation algorithm for $R(\Pi)$.

3 Reoptimization of DP-Benevolent Problems

In this Section we show that a wide class of optimization problems admits fully polynomial time $(1 + \varepsilon_1, 1 + \varepsilon_2)$-reapproximation schemes (FPTRS). The class of DP-benevolent problems (for short, $DP\text{-}B$) was introduced in [56]. In Sect. 3.1 we give some definitions describing the problems in this class (more details can be found in Appendix A). The reader may find it useful to refer to Sect. 3.4 that illustrates the definitions for the Knapsack problem. In Sects. 3.2 and 3.3 we show the existence of FPTRS for a problem $R(\Pi)$, where $\Pi \in DP\text{-}B$. This is done by showing the claim first for instances of $R(\Pi)$ with polynomially-bounded transition costs (in Sect. 3.2) and then generalizing to instances with arbitrary transition costs (Sect. 3.3). To simplify the presentation, we assume throughout the discussion that Π is a maximization problem. Our results can be extended to apply also for minimization problems, by using similar arguments.

3.1 The Class of DP-Benevolent Problems

Let Π be a problem in $DP\text{-}B$. For completeness, we include below the main definitions relating to problems in $DP\text{-}B$ and their solutions via dynamic programming, as given in [56]. The following formally defines the structure of the input for a problem Π in this class.

Definition 7 Let $\alpha \geq 1$ be a fixed integral constant. In any instance I of Π, the input is given by n vectors $\{\bar{X}_i \in \mathbb{N}^\alpha \mid 1 \leq i \leq n\}$. Each vector $\bar{X}_i$ consists of α non-negative integers $(x_{1,i}, \ldots, x_{\alpha,i})$. All entries of the vectors $\bar{X}_i$, $1 \leq i \leq n$, are encoded in binary.

Intuitively, α is the dimension of the input vectors, the number of different parameters associated with any element in the input. For example, in the 0/1 *Knapsack* problem (see Sect. 3.4) $\alpha = 2$, since each item i is associated with the vector (p_i, w_i), where $p_i > 0$ is its profit, and $w_i > 0$ gives its weight.

We use ';' to denote the concatenation of two vectors. Formally, for any two vectors $\bar{X} = (x_1, \ldots, x_a) \in \mathbb{N}^a$ and $\bar{Y} = (y_1, \ldots, y_b) \in \mathbb{N}^b$, $\bar{X}; \bar{Y} \in \mathbb{N}^{a+b}$, and $\bar{X}; \bar{Y} = (x_1, \ldots, x_a, y_1, \ldots, y_b)$. We use ';' to distinguish between the parameters of Π and the transition costs.

Definition 8 Structure of the dynamic program DP.
Given an input I of n elements, the dynamic program DP for the problem Π goes through n phases. The k-th phase processes the input piece $\bar{X}_k$ and produces a set $\mathcal{S}_k$ of states. Any state in the state space $\mathcal{S}_k$ is a vector $\bar{S} = (s_1, \ldots, s_\beta) \in \mathbb{N}^\beta$. The number β is a positive integer whose value depends on Π, but does not depend on any specific instance of Π.

Put plainly, β is the length of the vectors describing the states, indicating the number of factors involved in the calculation of the dynamic program.

The following definition refers to the set of functions that handle the transitions among states in the dynamic program.

Definition 9 Iterative computation of the state space in DP.
The set $\mathcal{F}$ is a finite set of mappings $\mathbb{N}^\alpha \times \mathbb{N}^\beta \to \mathbb{N}^\beta$. The set $\mathcal{H}$ is a finite set of mappings $\mathbb{N}^\alpha \times \mathbb{N}^\beta \to \mathbb{R}$. For every mapping $F \in \mathcal{F}$, there is a corresponding mapping $H_F \in \mathcal{H}$. In the initialization phase of DP, the state space $\mathcal{S}_0$ is initialized by a finite subset of $\mathbb{N}^\beta$. In the k-th phase of DP, $1 \le k \le n$, the state space $\mathcal{S}_k$ is obtained from the state space $\mathcal{S}_{k-1}$ via $\mathcal{S}_k = \left\{F\left(\bar{X}_k, \bar{S}\right) : \ F \in \mathcal{F}, \ \bar{S} \in \mathcal{S}_{k-1}, \ H_F\left(\bar{X}_k, \bar{S}\right) \le 0\right\}$.

While the transition mappings $\mathcal{F}$ determine the new state after any transition, the set of mappings $\mathcal{H}$ returns for each such transition an indication for the feasibility of the new state. Specifically, the new state $F\left(\bar{X}_k, \bar{S}\right)$ is feasible if $H_F\left(\bar{X}_k, \bar{S}\right) \le 0$.

The next definition refers to the objective value for the dynamic program which solves a maximization problem Π.

Definition 10 Objective value in DP.
The function $G : \mathbb{N}^\beta \to \mathbb{N}$ is a non-negative function. The optimal objective value of an instance I of Π is $\mathcal{O}(I) = \max\left\{G(\bar{S}) : \bar{S} \in \mathcal{S}_n\right\}$.

Theorem 2 [56] *If an optimization problem Π is DP-benevolent then it has an $FPTAS$.*

3.2 Polynomially Bounded Transition Costs

We first consider instances with transition costs polynomially bounded in the size of the instance I of Π. Our first main result is the following.

Theorem 3 *Let $R(\Pi)$ be the reoptimization version of a problem $\Pi \in DP$-B, with polynomial transition costs. Then there exists a* fully polynomial time $(1, 1+\varepsilon)$-reapproximation scheme (FPTRS) *for $R(\Pi)$.*

For simplicity, we assume that all transition costs are in $\{0, 1\}$. The same results hold for any integral transition costs that are bounded by some polynomial in the input size.

The key to proving Theorem 3 is to show for a given problem $\Pi \in DP$-B that a 'budgeted' version of $R(\Pi)$ is also in DP-B. In this version we seek the best approximation ratio for Π, subject to the constraint that the total transition cost does not exceed a given budget. Recall that $R(\Pi, m)$ is such a budgeted version of $R(\Pi)$,

in which the total transition cost is at most m, for some integer $m \geq 0$; also, $\mathcal{O}(I_R, m)$ is the optimal value of $R(\Pi, m)$ for the input I_R.

By the definitions and conditions satisfied by problems in DP-B, the dynamic program for Π, denoted by DP, can be described using a polynomial size set of states, which enable finding a close to the optimal solution. Let DP' denote the dynamic program for $R(\Pi, m)$. To deal with transition costs when solving $R(\Pi, m)$, we add a coordinate to the vector describing each state in DP, indicating the transition cost to this state. Thus, we duplicate each of the states in DP and attach to each copy the corresponding transition cost. We show that the consequences can be easily propagated to other elements of the dynamic programming formulation for Π (such as the domination relation between states; see Appendix B). This increases the size of the table for DP' within a polynomial factor, as long as the transition costs are polynomially bounded in the size of I.

Formally, we show that $R(\Pi, m)$ is DP-benevolent in two steps. First, we show that $R(\Pi, m)$ is DP-simple, i.e., it can be expressed via a simple dynamic programming formulation as specified in Definitions 8–10. In the second step, we show that $R(\Pi, m)$ satisfies Conditions A1–A4.

Lemma 1 *For any $\Pi \in DP$-B and $m \in \mathbb{N}$, the problem $R(\Pi, m)$ is DP-simple.*

Proof We reformulate Definitions 7–10 and show that they hold for $R(\Pi, m)$. Using Definitions 7 and 8, let α be the number of different parameters associated with any element in the input I; β is the length of the vectors describing the states in the dynamic program DP, and $\mathcal{F}$ is the set of mappings for the state vectors of Π. Also, the cost of the transition F for element i is given by $r_{F,i}$. Let $\bar{R}_i = \left(r_{F_1,i}, r_{F_2,i}, \ldots\right)$ be the cost vector for element i, associated with the input vector $\bar{X}_i$. For an arbitrary element in the input, we omit the index and denote the input vector by $\bar{X}$ and the cost vector by $\bar{R}$. □

Definition 11 Structure of the input for $R(\Pi, m)$ (reformulation of Definition 7). In any instance I_R of $R(\Pi, m)$, the input is structured into n vectors

$$\left\{\bar{Y}_i \in \mathbb{N}^{\alpha} \times \{0,1\}^{|\mathcal{F}|} \mid \bar{Y}_i = \left(\bar{X}_i; \bar{R}_i\right), \text{ and } \bar{X}_i \in I, 1 \leq i \leq n\right\}.$$

That is, any vector $\bar{Y}_i$ contains in the first α coordinates the input vector $\bar{X}_i$ of I. All of the $\alpha' = \alpha + |\mathcal{F}|$ components of the vectors $\bar{Y}_i$ are encoded in binary; therefore, the structure of $R(\Pi, m)$ is the same as given in Definition 7.

Definition 12 Structure of the dynamic program DP' (reformulation of Definition 8). The dynamic program DP' for problem $R(\Pi, m)$ goes through n phases. The k-th phase processes the input piece $\bar{Y}_k$ and produces a set $\mathcal{S}'_k$ of states. A state is a vector $\bar{\Upsilon} = \left(\bar{S}; c\right)$, where $\bar{S} \in \mathcal{S}_k \subseteq \mathbb{N}^{\beta}$ and $0 \leq c \leq m$.

Let β' denote the length of a state vector for $R(\Pi, m)$, then the structure of DP' can be specified using Definition 8, with $\beta' = \beta + 1$.

Definition 13 Iterative computation of the state space in DP' (reformulation of Definition 9). The set $\mathcal{F}'$ is a set of mappings $\mathbb{N}^{\alpha} \times \{0,1\}^{|\mathcal{F}|} \times \mathbb{N}^{\beta'} \rightarrow \mathbb{N}^{\beta'}$. For any $F \in \mathcal{F}$, there is a corresponding mapping $F' \in \mathcal{F}'$, where

$$F'\left(\bar{X};\bar{R},\bar{S};c\right) \equiv F\left(\bar{X},\bar{S}\right);(c+r_F).$$

The set of *validity functions* $\mathcal{H}'$ is a set of mappings $\mathbb{N}^{\alpha} \times \{0,1\}^{|\mathcal{F}|} \times \mathbb{N}^{\beta'} \rightarrow \mathbb{R}$. For any $H_F \in \mathcal{H}$ there is a corresponding mapping $H'_F \in \mathcal{H}'$, where

$$H'_F\left(\bar{X};\bar{R},\bar{S};c\right) = \begin{cases} min\left(H_F\left(\bar{X},\bar{S}\right),1\right) & \text{if } c+r_F \leq m \\ 1 & \text{otherwise} \end{cases}$$

Let $\mathcal{S}_0$ denote the set of initial states, where there is no input and the cost is 0. Then the corresponding state space for $R(\Pi,m)$ is $\mathcal{S}'_0 = \left\{\bar{S}' \mid \bar{S}' = \left(\bar{S};0\right),\ \bar{S} \in \mathcal{S}_0\right\}$. The state space for phase k of DP' is given by

$$\mathcal{S}'_k = \left\{F\left(\bar{Y}_k,\bar{\Upsilon}\right) \mid F \in \mathcal{F}',\ \bar{\Upsilon} \in \mathcal{S}'_{k-1} \text{ and } H'_F\left(\bar{Y}_k,\bar{\Upsilon}\right) \leq 0\right\}.$$

We note that, given the set of mappings $\mathcal{F}$, the set of validity functions $\mathcal{H}$, and the set of initial states $\mathcal{S}_0$ for Π, we have that $|\mathcal{F}'| = |\mathcal{F}|$, $|\mathcal{H}'| = |\mathcal{H}|$ and $|\mathcal{S}'_0| = |\mathcal{S}_0|$. Thus, the iterative computation of the state space for $R(\Pi,m)$ is as given in Definition 9.

Definition 14 Objective value in DP' (reformulation of Definition 10). For any $\bar{\Upsilon} = \left(\bar{S};c\right) \in \mathcal{S}'$, the objective function $G' : \mathbb{N}^{\beta'} \rightarrow \mathbb{N}$ is defined by $G'\left(\bar{\Upsilon}\right) = G\left(\bar{S}\right)$.

Thus, the objective value for DP' is as given in Definition 10. This completes the proof of the lemma. □

We show (in the appendix) that $R(\Pi,m)$ satisfies Conditions A1–A4, which define a DP-benevolent problem. Hence, we have

Theorem 4 *For any $\Pi \in DP\text{-}B$ and $m \in \mathbb{N}$, $R(\Pi,m) \in DP\text{-}B$.*

Intuitively, budgeted Π is also in DP$-B$, since the budget induces a new 'knapsack-like constraint' for Π. We give the formal proof of Theorem 4 in "Appendix B".

Let $\mathcal{O}_R(I_R) \equiv \mathcal{O}_R(I_R,1)$ be the minimal transition cost required to obtain an optimal solution for I. The next lemmas will be used in the proof of Theorem 3.

Lemma 2 *For any $\mu \geq 0$, if*

$$\mathcal{O}_R(I_R) \leq \mu \tag{1}$$

then $\mathcal{O}(I_R,\mu) = \mathcal{O}(I)$.

Proof If (1) holds then μ does not impose a restriction on the reoptimization cost. Formally, there exists a solution for the input I_R of $R(\Pi)$ that achieves the optimum value for I, and whose transition cost is at most μ. This solution is feasible also for the input I_R of $R(\Pi,m)$, for any $m \geq \mu$. □

Given a problem $\Pi \in$ DP-B, let $T(R(\Pi,m),I_R,\varepsilon)$ be the best objective value that can be obtained for the input I_R of $R(\Pi,m)$, by using DP' with ε as a parameter.

Lemma 3 *For any integer $m \geq 0$, if $\mathcal{O}_R(I_R) \leq m$ then $T(R(\Pi, m), I_R, \varepsilon) \geq T(\Pi, I, \varepsilon)/(1+\varepsilon)$.*

Proof Since $R(\Pi, m) \in DP\text{-}B$, by Theorem 2, for any $m \in \mathbb{N}$,

$$T(R(\Pi, m), I_R, \varepsilon) \geq \frac{\mathcal{O}(I_R, m)}{1+\varepsilon} = \frac{\mathcal{O}(I)}{1+\varepsilon} \geq \frac{T(\Pi, I, \varepsilon)}{1+\varepsilon}.$$

The equality follows from Lemma 2, and the second inequality holds since Π is a maximization problem. □

To find a $(1, 1+\varepsilon)$-reapproximation for $R(\Pi)$, we need to search over a set of positive integer values m for the problem $R(\Pi, m)$. We select in this set the minimal value of μ satisfying (1). Formally,

Definition 15 For any maximization problem Π and $\varepsilon \geq 0$,

$$\ell(R(\Pi), I_R, \varepsilon) \equiv \min \left\{ m \in \mathbb{N} \mid T(R(\Pi, m), I_R, \varepsilon) \geq \frac{T(\Pi, I, \varepsilon)}{1+\varepsilon} \right\} \tag{2}$$

For short, let $\ell^* = \ell(R(\Pi), I_R, \varepsilon)$ be the minimal value found in (2). The next two lemmas show that $T(R(\Pi, \ell^*), I_R, \varepsilon)$ satisfies two properties: (i) The transition cost is at most $\mathcal{O}_R(I_R)$, and (ii) the resulting solution yields a $(1+2\varepsilon)$-approximation for the maximization problem Π.

Lemma 4 *For any $\varepsilon \geq 0$, $\ell^* \leq \mathcal{O}_R(I_R)$.*

Proof By Lemma 3, the inequality in (2) holds for any $m \geq \mathcal{O}_R(I_R)$. Since we choose the minimum value of m satisfying (2), this gives the claim. □

Lemma 5 *For any $\varepsilon \geq 0$, $T(R(\Pi, \ell^*), I_R, \varepsilon) \geq \frac{\mathcal{O}(I)}{1+2\varepsilon}$.*

Proof By the definition of ℓ^*, it holds that

$$T(R(\Pi, \ell^*), I_R, \varepsilon) \geq \frac{T(\Pi, I, \varepsilon)}{1+\varepsilon} \geq \frac{\mathcal{O}(I)}{(1+\varepsilon)^2} \geq \frac{\mathcal{O}(I)}{1+2\varepsilon}.$$

The second inequality follows from the fact that $\Pi \in DP\text{-}B$. □

Proof of Theorem 3: We need to show that, for any input I_R, $R(\Pi)$ can be reapproximated within factor $1+\varepsilon$ using the optimal reoptimization cost. We have the claim by combining Lemmas 4 and 5.

Next, we show that the resulting reapproximation scheme is polynomial in n and in $1/\varepsilon$. This holds since, by Theorem 4, $R(\Pi, m) \in$ DP-B for any $m \in \mathbb{N}$. Also, ℓ^* can be found in polynomial time in the input size. □

3.3 Arbitrary Transition Costs

Recall that any vector $\bar{Y}_i \in I_R$ is defined as $\bar{Y}_i = \left(\bar{X}_i; \bar{R}_i\right)$, where $\bar{R}_i$ is the transition cost vector associated with $\bar{X}_i$ in I_R. To obtain approximate solutions for instances with arbitrary transition costs, we first apply a transformation on the cost vectors $\bar{R}_i$ of the elements. Let n_i denote the number of entries in $\bar{R}_i$.

Definition 16 Given an input I_R for $R(\Pi)$, let γ be a function that accepts as input the cost vector $\bar{R}_i$, $1 \leq i \leq n$, and the parameters $d, n \in \mathbb{N}$ and $\varepsilon \in (0, 1)$, then $\gamma\left(\bar{R}_i, d, n, \varepsilon\right) = \left(\gamma\left(r_{F_1,i}\right), \gamma\left(r_{F_2,i}\right), \ldots\right)$, where

$$\gamma\left(r_{F_j,i}\right) = \begin{cases} 0 & \text{if } 0 \leq r_{F_j,i} < \dfrac{\varepsilon d}{n} \\ (1+\varepsilon)^h & \text{if } r_{F_j,i} \in \left[\dfrac{\varepsilon d}{n}(1+\varepsilon)^h, \dfrac{\varepsilon d}{n}(1+\varepsilon)^{h+1}\right) \\ & \text{for an integer } 0 \leq h \leq \lceil \log_{1+\varepsilon}\left(\dfrac{n}{\varepsilon}\right) \rceil \end{cases}$$

Applying γ on I_R, each element $\bar{Y}_i = \left(\bar{X}_i; \bar{R}_i\right) \in I_R$ is modified to $\bar{Y}'_i = \left(\bar{X}_i; \gamma\left(\bar{R}_i, d, n, \varepsilon\right)\right)$. The resulting instance is denoted $\hat{I}_{R,\varepsilon}$. We give below the pseudocode for $\mathcal{A}_{DP-B}$, our algorithm for finding a $(1+\varepsilon_1, 1+\varepsilon_2)$-reapproximation for $R(\Pi)$, where $\Pi \in DP\text{-}B$.

Algorithm 1 $\mathcal{A}_{DP\text{-}B}$

1: Given an input I_R for $R(\Pi)$ and $\varepsilon_1, \varepsilon_2 > 0$, guess the maximum transition cost, $0 \leq C_{max} \leq \max_{1\leq i\leq n}\left\{\max_{1\leq j\leq n_i} r_{F_j,i}\right\}$, used in some solution of minimum total transition cost.
2: Use a binary search to find the minimum total transition cost $m \in [C_{max}, nC_{max}]$, which enables to approximate the optimization problem, Π, within factor $(1+\varepsilon_2)$, using the rounded instance $\hat{I}_{R,\varepsilon_1}$.
3: Output a $(1+\varepsilon_1, 1+\varepsilon_2)$-reapproximate solution for $R(\Pi)$, i.e., a solution that is $(1+\varepsilon_2)$-approximation for Π on the input I_R, whose total transition cost is at most $m(1+\varepsilon_1)$.

Theorem 5 *Let $R(\Pi)$ be the reoptimization version of a problem $\Pi \in DP\text{-}B$, then for any transition costs, there exists a* fully polynomial time $(1+\varepsilon_1, 1+\varepsilon_2)$-reapproximation scheme (FPTRS) *for $R(\Pi)$.*

Proof Given an input I_R for $R(\Pi)$ and $\varepsilon_1 > 0$, let m be the optimal transition cost that we have guessed correctly.

Since the transition costs in $\hat{I}_{R,\varepsilon_1}$, generated by taking $\gamma\left(\bar{R}_i, m, n, \varepsilon_1\right)$ for each cost vector $\bar{R}_i$, $1 \leq i \leq n$, are rounded down, it holds that $\mathcal{O}_R(\hat{I}_{R,\varepsilon_1}) \leq \mathcal{O}_R(I_R)$. Also, the transition costs in $\hat{I}_{R,\varepsilon_1}$ are polynomial. Thus, by Theorem 3, for any fixed $\varepsilon_2 > 0$, there exists an FPTRS that yields a $(1, 1+\varepsilon_2)$-reapproximation for the input $\hat{I}_{R,\varepsilon_1}$ of $R(\Pi)$. Returning to the original instance, I_R, we have that the total cost increases at most by a factor $(1+\varepsilon_1)$. This is due to the fact that each element having 'large' transition cost in the optimal solution for $\hat{I}_{R,\varepsilon_1}$, i.e., $(1+\varepsilon)^h$, for some

$0 \leq h \leq \lceil \log_{1+\varepsilon}(\frac{n}{\varepsilon}) \rceil$, contributes to $\mathcal{O}_R(I_R)$ at most $(1+\varepsilon)^{h+1}$. Also, the total contribution of transitions with 'small' cost (i.e., each such transition of an element $1 \leq i \leq n$ has cost $0 \leq r_{F_j,i} < \frac{\varepsilon m}{n}$) is at most εm, since there are at most n such transitions.

The running time of $\mathcal{A}_{DP\text{-}B}$ is polynomial. This follows from the fact that to guess C_{max}, we can simply go over all the cost vectors. Recall that we have such a cost vector for each element in the input. This vector, $\bar{R}_i$, gives all the possible transition costs for element i, for $1 \leq i \leq n$, depending on the state to which element i moves in the solution. Also, guessing m, once we have guessed C_{max}, is done in polynomial time. □

We note that the result in Theorem 5 is the best possible, unless $P = NP$. Indeed, there exist optimization problems Π that can be reduced to their reoptimization version, $R(\Pi)$. This includes, e.g., the subclass of minimum subset selection problems, in which we can use the costs in a given instance I as transition costs and assign to all elements initial cost 0. Thus, solving Π for I is equivalent to solving $R(\Pi)$ for I_R.

The next result follows from applying Theorem 5 to the problems in DP-B.

Corollary 1 *Each of the problems listed in Appendix A under "Some Problems in DP-B" admits an FPTRS.*

3.4 Example: Reoptimization of the Knapsack Problem

The input for the Knapsack problem consists of n pairs of positive integers (p_k, w_k) and a positive integer W. Each pair (p_k, w_k) corresponds to an item having profit p_k and weight w_k. The parameter W is the weight bound of the knapsack. The goal is to select a subset of items $K \subseteq \{x \in \mathbb{N} | 1 \leq x \leq n\}$ such that $\sum_{k \in K} w_k \leq W$ and the total profit $\sum_{k \in K} p_k$ is maximized. Knapsack is in $DP\text{-}B$ with $\alpha = 2$ and $\beta = 2$. For $1 \leq k \leq n$ define the input vector $\bar{X}_k = [p_k, w_k]$. A state $\bar{S} = [s_1, s_2]$ in $\mathcal{S}_k$ encodes a partial solution for the first k indices (items): s_1 gives the total weight, and s_2 gives the total profit of the partial solution. The set $\mathcal{F}$ consists of two functions F_1 and F_2.

$$F_1(w_k, p_k, s_1, s_2) = [s_1 + w_k, s_2 + p_k]$$
$$F_2(w_k, p_k, s_1, s_2) = [s_1, s_2]$$

In words, the function F_1 adds the k-th item to the partial solution, and F_2 does not add it. In the set $\mathcal{H}$, there is a function $H_1(w_k, p_k, s_1, s_2) = s_1 + w_k - W$ corresponding to F_1 and a function $H_2(w_k, p_k, s_1, s_2) = 0$ corresponding to F_2. Finally, for the objective function, let $G(s_1, s_2) = s_2$ to extract the total profit from a solution. The initial state space is defined to be $\mathcal{S}_0 = \{[0, 0]\}$.

The budgeted reoptimization problem $R(\text{Knapsack}, m)$ accepts as input n vectors $\bar{Y}_k = \left(\bar{X}_k; \bar{R}_k\right) \in \mathbb{N}^2 \times \{0, 1\}^2$, where $\bar{X}_k$ is the original input vector, and $\bar{R}_k$ is the transition-cost vector for the k-th item. Specifically, $\bar{Y}_k = (p_k, w_k, c(F_1, k), c(F_2, k))$, where $c(F_1, k)$ is the cost for placing the k-th item in the knapsack, and $c(F_2, k)$ is the cost of not placing it. Let K_0 be the set of items packed in the initial configuration,

then $c(F_1, k) = 1$ if $k \notin K_0$, $c(F_1, k) = 0$ if $k \in K_0$, $c(F_2, k) = 0$ if $k \notin K_0$ and $c(F_2, k) = 1$ if $k \in K_0$.

The goal is to select an index set $K \subseteq \{x \in \mathbb{N} | 1 \leq x \leq n\}$ such that (i) $\sum_{k \in K} w_k \leq W$, (ii) the total transition cost is at most the budget, that is, $\sum_{k \in K} c(F_1, k) + \sum_{k \in K} c(F_2, k) \leq m$, and (iii) the profit $\sum_{k \in K} p_k$ is maximized.

$R(\text{Knapsack}, m)$ is in $DP\text{-}B$ with $\alpha' = 2 + 2$ and $\beta' = 2 + 1$. For $1 \leq k \leq n$ define the input vector $\bar{X}_k = [p_k, w_k, c(F_1, k), c(F_2, k)]$. A state $\bar{S} = [s_1, s_2, s_3]$ in $\mathcal{S}'_k$ encodes a partial solution for the first k indices. The value of s_3 gives the total transition cost of the partial solution. The set $\mathcal{F}'$ consists of two functions F'_1 and F'_2.

$$F'_1(w_k, p_k, c(F_1, k), c(F_2, k), s_1, s_2, s_3) = [s_1 + w_k, s_2 + p_k, s_3 + c(F_1, k)]$$
$$F'_2(w_k, p_k, c(F_1, k), c(F_2, k), s_1, s_2, s_3) = [s_1, s_2, s_3 + c(F_2, k)]$$

Intuitively, the function F_1 adds the k-th item to the partial solution, and F_2 does not add it. In the set $\mathcal{H}'$ there is a function

$$H'_1(w_k, p_k, c(F_1, k), c(F_2, k), s_1, s_2, s_3) = \begin{cases} s_1 + w_k - W & \text{if } s_3 + c(F_1, k) \leq m \\ 1 & \text{otherwise} \end{cases}$$

corresponding to F'_1 and a function

$$H'_2(w_k, p_k, c(F_1, k), c(F_2, k), s_1, s_2, s_3) = \begin{cases} 0 & \text{if } s_3 + c(F_2, k) \leq m \\ 1 & \text{otherwise} \end{cases}$$

corresponding to F'_2. Finally, let $G(s_1, s_2, s_3) = s_2$ to extract the total profit from a solution. The initial state space is defined to be $\mathcal{S}'_0 = \{[0, 0, 0]\}$.

4 Reoptimization of the k-Center Problem

In this section we show that the reoptimization version of the classic k-Center problem can be solved with approximation ratio close to the best known of 2 (see [21,31]) and the minimum reoptimization cost. The input for the problem is a set of n clients numbered by $1, \ldots, n$, in a metric space. The goal is to open centers at k clients so that the maximum distance from a client to its nearest center is minimized. Let $\Pi(I_0)$ be the k-Center problem over an instance I_0. In the reoptimization version the instance I_0 is modified. The change can involve insertion or deletion of clients, as well as changes in the distance function. Denote by I the modified instance. Given an approximate solution S_0 for $\Pi(I_0)$, the goal is to find an approximate solution S for $\Pi(I)$, such that S has the minimal possible transition cost from S_0. Specifically, the opening cost of any center $i \in S_0$ is equal to zero, while there is an arbitrary positive cost associated with opening a center at any other location. Denote by $c(j) \geq 0$ the cost of opening a center at client j, $1 \leq j \leq n$. The transition cost from S_0 to S is the sum of the costs of the centers in $S \backslash S_0$, which is equal to $c(S) = \sum_{\ell \in S} c(\ell)$ (since $c(\ell) = 0$ for $\ell \in S_0$). For a set of centers $S \subseteq \{1, \ldots, n\}$, let $d(j, S)$ be the distance from client j to the

closest center in S. We give below the pseudocode of $\mathcal{A}_{R(k\text{-}Center)}$, a reapproximation algorithm for $R(k\text{-}Center)$. The transition cost incurred by the algorithm is measured relative to the optimum, as given in Definition 3.

Algorithm 2 $\mathcal{A}_{R(k\text{-}Center)}$

Let $P = \{1, \ldots, n\}$ be the set of clients.
Guess D_{OPT}, the maximum distance from a client to the nearest center in an optimal solution, i.e., $D_{OPT} = \min_{\{S \subseteq P : |S| \leq k\}} \max_{j \in P} d(j, S)$.
$\mathcal{B} = \emptyset$.
for $1 \leq j \leq n$ **do**
  Define a ball B_j of radius D_{OPT}, whose center point is client j.
  $\mathcal{B} = \mathcal{B} \cup \{j\}$.
end for
Let $S = \emptyset$ and $c(S) = 0$.
while $|S| < k$ **do**
  Find an active ball B_i, $i \in \mathcal{B}$, that contains a client j at which the cost of opening a center in minimized, i.e., $c(j) = \min_{\{\ell \in P \setminus S, \ell \in \cup_{r \in \mathcal{B}} B_r\}} c(\ell)$.
  $S = S \cup \{j\}$
  $c(S) = c(S) + c(j)$
  Omit the balls intersecting B_j, i.e., $\mathcal{B} = \mathcal{B} \setminus \{\{r\} : B_r \cap B_j \neq \emptyset\}$
end while
Return S, $c(S)$

Theorem 6 *$\mathcal{A}_{R(k\text{-}Center)}$ is a $(1, 3)$-reapproximation algorithm for $R(k\text{-}Center)$.*

Proof Let $c(S_{OPT})$ be the minimum transition cost to an optimal set of centers, S_{OPT}, that achieves the radius D_{OPT}. Suppose that we have guessed correctly D_{OPT}. Define the *cost lower bound (CLB)* of a ball B_j of radius D_{OPT} to be the cost of the cheapest center that can be opened inside B_j. Clearly, the cost of any center that serves client j is at least $CLB(B_j)$. Moreover, we say that two clients i and j are *independent* if their respective balls do not intersect. Consider any independent set of clients. Clearly, the sum of the cost lower bounds of the corresponding balls is a lower bound on the cost of covering these clients by OPT, an optimal solution for the input I. This is due to the fact that any optimal solution has to open a different center for each such client. It follows also that there are at most k independent clients.

We first argue that $\mathcal{A}_{R(k\text{-}Center)}$ incurs at most the optimal opening cost, i.e., the set S of centers output by the algorithm satisfies $c(S) \leq c(S_{OPT})$, where S_{OPT} is the set of centers opened by OPT. Note that $\mathcal{A}_{R(k\text{-}Center)}$ defines an independent set of clients (the set of clients whose balls are considered), and $c(S)$, the cost of $\mathcal{A}_{R(k\text{-}Center)}$, is exactly the sum of the cost lower bounds of these balls. Hence, $c(S) \leq c(S_{OPT})$.

Now, for the maximum distance from a client to the closest center, we note that each client in the independent set is at distance at most D_{OPT} from a center opened by $\mathcal{A}_{R(k\text{-}Center)}$. Furthermore, as we have guessed correctly D_{OPT}, any maximal independent set of clients is of size at most k. Since the independent set defined by $\mathcal{A}_{R(k\text{-}Center)}$ is maximal, after at most k iterations $\mathcal{B}$ is empty. Therefore, any client not in the independent set is at distance at most $2D_{OPT}$ from a client in the independent set, and thus at distance at most $3D_{OPT}$ from a center.

Finally, $\mathcal{A}_{R(k\text{-}Center)}$ has polynomial running time, since the optimal radius, D_{OPT}, can be guessed by exhaustive search over the distances between all pairs of clients. Other steps can also be implemented in $O\left(n^2\right)$ time. □

The above approach can be applied also to derive a $(1, 3)$-reapproximation algorithm for the k-Supplier problem considered in [32]. The input for the k-Supplier problem is a set P of n points in a metric space, partitioned to two disjoint sets: a set P_C of clients and a set P_S of suppliers. The goal is to open k suppliers (from P_S) so that the maximum distance from a client (in P_C) to its nearest open supplier is minimized. The algorithm for the k-Supplier problem follows $\mathcal{A}_{R(k\text{-}Center)}$. We guess D_{OPT} and then define balls of radius D_{OPT} around every client in P_C. For a correct guess, each such ball must include a supplier, and any maximal independent set of clients is of size at most k. We find such a maximal independent set and open the cheapest supplier in each ball of the independent set. As before, the opening cost is the sum of the cost lower bounds for these balls, and any client not in the independent set is at distance at most $2D_{OPT}$ from a client in the independent set, and thus at distance at most $3D_{OPT}$ from an open supplier. This implies a $(1, 3)$-reapproximation. Note that for this problem it was shown in [32] that 3-approximation is the best possible unless P = NP, thus $(1, 3)$-reapproximation is also best possible.

5 Optimal Reoptimization of Weighted Subset-Selection

In a weighted subset-selection problem, we are given a set I of weighted elements, and the goal is to select a subset $S \subseteq I$ satisfying various constraints, such that $\sum_{i \in S} w_i$ is maximal. In this section we show that for any subset-selection problem Π over n elements that can be solved optimally in $T(n)$ time, there is a $(1, 1)$-reoptimization algorithm for the reoptimization version of Π, whose running time is $T(n')$, where n' is the size of the modified input. In particular, if Π is solvable in polynomial time, then so is its reoptimization variant. This includes the minimum spanning tree problem, shortest path problems, maximum matching, maximum weighted independent set in interval graphs, and more. Similarly, if Π is fixed parameter tractable, then so is $R(\Pi)$.

We describe the framework for maximization problems. With slight changes it fits also for minimization problems.

Let Π be a polynomially solvable subset-selection maximization problem over an instance I_0. The weight of an element $i \in I_0$ is given by an integer $w_i \geq 0$. The goal is to select a subset $S_0 \subseteq I_0$ satisfying various constraints, such that the total weight of the elements in S is maximized. In the reoptimization problem, the instance I_0 is modified. The change can involve insertion or deletion of elements, as well as changes in element weights. For example, in the maximum matching problem, possible changes are addition or deletion of vertices and edges, as well as changes in edge weights. Denote by I the modified instance. Let w'_i denote the modified weight of element i. For a given optimal solution S_0 of $\Pi(I_0)$, the goal in the reoptimization problem is to find an optimal solution S of $\Pi(I)$ with respect to the modified weights, such that S has the minimal possible transition cost from S_0. Specifically, every element $i \in I$, is associated with a removal-cost $\delta_{rem}(i)$ to be charged if $i \in S_0 \backslash S$, and an addition-cost $\delta_{add}(i)$ to be charged if $i \in S \backslash S_0$. The transition cost from S_0 to S is

defined as the sum of the corresponding removal- and addition-costs. The following is a (1, 1)-reoptimization algorithm for $R(\Pi)$.

Algorithm 3 $\mathcal{A}_{Subset-Select}$: A (1,1)-reoptimization algorithm for $R(\Pi)$

Let $\Delta = \max\{\max_{i \in S_0 \cap I} \delta_{rem}(i), \max_{i \in I \setminus S_0} \delta_{add}(i)\}$.
Let $\lambda = 2|I|\Delta + 1$.
for all $i \in I \cap S_0$ **do**
    $\hat{w}_i = \lambda w'_i + \delta_{rem}(i)$.
end for
for all $i \in I \setminus S_0$ **do**
    $\hat{w}_i = \lambda w'_i - \delta_{add}(i)$
end for
Solve $\Pi(I)$ with weights $\hat{w}$.

Theorem 7 *An optimal solution for $\Pi(I)$ with weights $\hat{w}$ is an optimal solution for $\Pi(I)$ with weights w', and a minimal transition cost, i.e., it is a* (1, 1)*-reoptimization for $R(\Pi)$.*

Proof The proof combines two claims, regarding the optimality of the solution for $\Pi(I)$ and regarding the minimization of the transition costs. Being a subset-selection problem, the solution for Π is given by a binary vector $X = (x_1, \ldots, x_n)$, where x_i indicates if i is in the solution. Let $X_S = (x_1, \ldots, x_n)$ be the vector representing the solution provided by the algorithm, and let S_X denote the associated set of elements in the solution. □

Claim 17 *An optimal solution for $\Pi(I)$ with weights $\hat{w}$ is an optimal solution for $\Pi(I)$ with weights w'.*

Proof Assume by way of contradiction that X_S is not optimal for $\Pi(I)$ with weights w', and that $Y = (y_1, \ldots, y_n)$ is a vector representing a better solution.

Therefore,

$$1 + \sum_{i=1}^{n} w'_i \cdot x_i \leq \sum_{i=1}^{n} w'_i \cdot y_i. \tag{3}$$

Let S_Y denote the set of elements in the solution Y. The profit of Y on $\Pi(I)$ with weights $\hat{w}$ is

$$\sum_{i=1}^{n} \hat{w}_i \cdot y_i = \lambda \sum_{i=1}^{n} w'_i \cdot y_i + \sum_{i \in S_0 \cap I \cap S_Y} \delta_{rem}(i) - \sum_{i \in S_Y \setminus S_0} \delta_{add}(i).$$

By definition of $\hat{w}$, this value can be bounded as follows.

$$\lambda \sum_{i=1}^{n} w'_i y_i - n\Delta \leq \sum_{i=1}^{n} \hat{w}_i \cdot y_i \leq \lambda \sum_{i=1}^{n} w'_i y_i + n\Delta.$$

By the optimality of X for $\Pi(I)$ with weights $\hat{w}$, it holds that $\sum_{i=1}^{n} \hat{w}_i \cdot x_i \geq \sum_{i=1}^{n} \hat{w}_i \cdot y_i$. Also, by definition of $\hat{w}$, it holds that

$$\lambda \sum_{i=1}^{n} w'_i x_i - n\Delta \leq \sum_{i=1}^{n} \hat{w}_i \cdot x_i \leq \lambda \sum_{i=1}^{n} w'_i x_i + n\Delta.$$

Combining with Eq. (3) (multiplied by λ), and the fact that $\lambda = 2n\Delta + 1$, we get the following contradiction:

$$\begin{aligned}\lambda \sum_{i=1}^{n} w'_i x_i + \lambda \leq \lambda \sum_{i=1}^{n} w'_i y_i &\leq \sum_{i=1}^{n} \hat{w}_i \cdot y_i + n\Delta \\ &\leq \sum_{i=1}^{n} \hat{w}_i \cdot x_i + n\Delta \leq \lambda \sum_{i=1}^{n} w'_i x_i + 2n\Delta < \lambda \sum_{i=1}^{n} w'_i x_i + \lambda.\end{aligned}$$

□

Claim 18 *Among all the optimal solutions of $\Pi(I)$ with weights w', the solution with weights $\hat{w}$ has the minimal transition cost from S_0.*

Proof Let $X = (x_1, \ldots, x_n)$ be an optimal solution for $\Pi(I)$ with weights $\hat{w}$. We already argued that X is an optimal solution for $\Pi(I)$ with weights w'. Let $Y = (y_1, \ldots, y_n)$ be another optimal solution for $\Pi(I)$ with weights w'. It follows that $\sum_{i=1}^{n} w'_i x_i = \sum_{i=1}^{n} w'_i y_i$. Let $D_X = \sum_{i=1}^{n} \hat{w}_i x_i - \sum_{i=1}^{n} w'_i x_i$. Since X is an optimal solution for $\Pi(I)$ with weights $\hat{w}$, $D_X \geq D_Y$. The total transition cost from S_0 to S_X is given by $\sum_{i \in S_X \setminus S_0} \delta_{add}(i) + \sum_{i \in (S_0 \cap I) \setminus S_X} \delta_{rem}(i)$. Note that by the definition of $\hat{w}$ the transition cost can be written as $\sum_{i \in S_0 \cap I} \delta_{rem}(i) - D_X$. Clearly, $D_X \geq D_Y$ implies that $\sum_{i \in S_0 \cap I} \delta_{rem}(i) - D_X \leq \sum_{i \in S_0 \cap I} \delta_{rem}(i) - D_Y$. Thus, the transition cost of Y is at least the transition cost of X. □

Combining the above claims, we get that the algorithm yields a $(1, 1)$-reoptimization for $R(\Pi)$. □

The above framework is unsuitable when the objective is finding a subset of minimum (maximum) *cardinality*. Moreover, in some cases, the non-weighted problem is polynomially solvable, while the reoptimization version is NP-hard.

Theorem 8 *There exist polynomially-solvable subset-selection problems whose reoptimization variants are NP-hard.*

Proof Consider the job scheduling problem $1||\sum U_j$. The input to this problem is a set of jobs, each associated with a processing time and a due date. The goal is to assign the jobs on a single machine in a way that minimizes the number of late jobs (whose completion time is after their due date). Since this goal is equivalent to maximizing the number of jobs that complete their processing on time, the problem can be viewed as a subset-selection problem: For any set of jobs, the set is feasible if and only if no job is late when the jobs are scheduled according to EDD order (non-decreasing due date). The problem is solvable by an algorithm of Moore [44].

In the reoptimization version of $1||\sum U_j$, we are given a schedule of an instance I_0. The instance is then modified to an instance I in which jobs may be added or removed, and job lengths or due dates may be modified. Transition costs are charged for changes in the set of jobs completing on time. The reoptimization problem $R(1||\sum U_j)$ is shown to be NP-hard via a reduction from the weighted problem $1||\sum w_j U_j$, which is known to be NP-hard [39]. Given an instance of the weighted problem, the instance I_0 of the reoptimization problem has the same set of jobs and processing times, but with very large due dates, so that all jobs complete on time. The instance I has the same set of jobs and processing times, but also the same due dates as in the original weighted instance. The cost of removing a job from the set of jobs completing on time is the weight of the job. Since the initial set includes all jobs, minimizing the transition cost is equivalent to minimizing the weight of late jobs. □

6 Discussion

In this paper we developed a framework for combinatorial reoptimization. We defined the notion of *reapproximation* and showed that for many optimization problems, strong reapproximation algorithms are unlikely to exist, unless P = NP. This led us to an alternative definition that is used to obtain reapproximation algorithms as well as FPTRS for a non-trivial subclass of DP-benevolent problems.

The theoretical model introduced in this paper serves as a first step in the study of the reoptimization variants of classical problems, which arise in many practical scenarios. Our results show how known techniques from combinatorial optimization can be enhanced to obtain efficient reapproximation algorithms (or reapproximation schemes). It is natural to try and develop a generic approach for obtaining reapproximation algorithms for a family of metric network design problems, based on known approximation algorithms for these problems. More generally, in the study of reoptimization variants of NP-hard problems, suppose that there exists an α-approximation algorithm for such optimization problem Π. Is there a polynomial time (r, α)-reapproximation algorithm for $R(\Pi)$, for some bounded value $r > 1$? We have shown that any (weighted) subset selection problem that is polynomially solvable admits a $(1, 1)$-reoptimization algorithm. The existence of such optimal algorithms for a wider class of problems remains open.

Finally, while our model captures well the transition from one solution to the other, namely, scenarios where an initial input I_0 changes to a new one, I, it is interesting to consider also scenarios in which a *sequence* of changes is applied to an initial input. Formally, in addition to the initial input I_0 and a solution S_0, the instance of the reoptimization problem consists also of a sequence $(I', I'', \ldots)$ of inputs. The goal is to find a solution for each of the inputs in the sequence optimizing the quality of the solutions and the total transition cost (with no transition costs such a problem is studied in [25]). An optimal solution for sequence reoptimization may be significantly different from the solution derived by combining the optimal transition for each pair of consecutive solutions in the sequence. It is natural to examine also the usage of the techniques developed for *incremental approximation* (see, e.g., [43]). Here, the algorithms grad-

ually modify the solutions for a given sequence of inputs, while guaranteeing that, for any $i > 1$, the i-th solution contains the first $(i-1)$ solutions.

Acknowledgements This work was partly carried out during a visit of the second author to DIMACS supported by the National Science Foundation under Grant Number CCF-1144502. Work partially supported also by the Technion V.P.R. Fund, and by the Smoler Research Fund. We thank Rohit Khandekar and Viswanath Nagarajan for stimulating discussions on the paper.

Appendix A: Additional Properties of the Class *DP-B*

In this section we give some more definitions and the conditions that a problem Π must satisfy (as given in [56]) to be DP-benevolent. The class of DP-B problems contains a large set of problems that admit an $FPTAS$ via dynamic programming. The reader may find it useful to refer to Sect. 3.4 that illustrates the definitions for the Knapsack problem.

An optimization problem Π is called DP-$simple$ if it can be expressed via a simple dynamic programming formulation DP as specified in Definitions 8–10.

The following gives a measure for the closeness of two states, using a *degree vector* $\bar{D}$ of length β, which indicates the distances between two states coordinate-wise, and a value Δ.

Definition 19 For any real number $\Delta > 1$ and three vectors $\bar{D}, \bar{S}, \bar{S}' \in \mathbb{N}^\beta$, we say that $\bar{S}$ is $(\bar{D}, \Delta)$-close to $\bar{S}'$ if $\Delta^{-D_\ell} \cdot s_\ell \leq s'_\ell \leq \Delta^{D_\ell} \cdot s_\ell$ for every coordinate $1 \leq \ell \leq \beta$. Note that if two vectors are $(\bar{D}, \Delta)$-close to each other then they must agree in all coordinates ℓ with $D_\ell = 0$.

Let $\preceq_{dom}$ and $\preceq_{qua}$ denote two partial orders on the state space of the dynamic program, where $\preceq_{qua}$ is an extension of $\preceq_{dom}$.[7] We say that $\bar{S} \preceq_{dom} \bar{S}'$ if $\bar{S}'$ is 'better' than $\bar{S}$ (whatever choices are made later). Note that the partial order $\preceq_{dom}$ does not depend on the algorithm used for solving the problem. In some cases, the partial order $\preceq_{dom}$ is undefined for $\bar{S}$ and $\bar{S}'$. In such cases, we may define the partial order using $\preceq_{qua}$.

We now give some conditions that will be used to identify the class of DP-benevolent problems.

Condition A1 *Conditions on the function in $\mathcal{F}$.*
The following holds for any $\Delta > 1$, $F \in \mathcal{F}$ and $\bar{X} \in \mathbb{N}^\alpha$, and for any $\bar{S}, \bar{S}' \in \mathbb{N}^\beta$.

(i) *If $\bar{S}$ is $(\bar{D}, \Delta)$-close to $\bar{S}'$, and if $\bar{S} \preceq_{qua} \bar{S}'$, then*
  (a) *$F(\bar{X}, \bar{S}) \preceq_{qua} F(\bar{X}, \bar{S}')$, and $F(\bar{X}, \bar{S})$ is $(\bar{D}, \Delta)$-close to $F(\bar{X}, \bar{S}')$, or*
  (b) *$F(\bar{X}, \bar{S}) \preceq_{dom} F(\bar{X}, \bar{S}')$.*
(ii) *If $\bar{S} \preceq_{dom} \bar{S}'$ then $F(\bar{X}, \bar{S}) \preceq_{dom} F(\bar{X}, \bar{S}')$.*

Condition A2 *Conditions on the functions in $\mathcal{H}$.*
The following holds for any $\Delta \geq 1$, $H \in \mathcal{H}$ and $\bar{X} \in \mathbb{N}^\alpha$, and for any $\bar{S}, \bar{S}' \in \mathbb{N}^\beta$.

[7] This is made precise below.

(i) *If* $\bar{S}$ *is* $(\bar{D}, \Delta)$*-close to* $\bar{S}'$*, and if* $\bar{S} \preceq_{qua} \bar{S}'$ *then* $H(\bar{X}, \bar{S}') \leq H(\bar{X}, \bar{S})$.
(ii) *If* $\bar{S} \preceq_{dom} \bar{S}'$ *then* $H(\bar{X}, \bar{S}') \leq H(\bar{X}, \bar{S})$.

We now refer to our objective function (assuming a maximization problem).

Condition A3 *Conditions on the function G.*

(i) *There exists an integer* $g \geq 0$ *(whose value only depends on the function G and the degree-vector* $\bar{D}$*) such that, for any* $\Delta \geq 1$*, and for any* $\bar{S}, \bar{S}' \in \mathbb{N}^{\beta}$ *the following holds: If S is* $(\bar{D}, \Delta)$*-close to* $\bar{S}'$*, and* $\bar{S} \preceq_{qua} \bar{S}'$ *then* $G(\bar{S}) \leq \Delta^g \cdot G(\bar{S}')$.
(ii) *For any* $\bar{S}, \bar{S}' \in \mathbb{N}^{\beta}$ *with* $\bar{S} \preceq_{dom} \bar{S}'$, $G(\bar{S}') \geq G(\bar{S})$.

The next condition gives some technical properties guaranteeing that the running time of the algorithm is polynomial in the input size, i.e., in n and $\bar{x}$, where $\bar{x} = \sum_{k=1}^{n} \sum_{i=1}^{\alpha} x_{i,k}$ is the total sum of entries in the input vectors.

Condition A4 *Technical properties.*

(i) *Every* $F \in \mathcal{F}$ *can be evaluated in polynomial time. Also, every* $H \in \mathcal{H}$ *can be evaluated in polynomial time; the function G can be evaluated in polynomial time, and the relation* $\preceq_{qua}$ *can be decided in polynomial time.*
(ii) *The cardinality of* $\mathcal{F}$ *is polynomially bounded in n and* $\log \bar{x}$.
(iii) *For any instance I of* Π*, the state space* $\mathcal{S}_0$ *can be computed in time that is polynomially bounded in n and* $\log \bar{x}$.
(iv) *For an instance I of* Π*, and for a coordinate* $1 \leq c \leq \beta$*, let* $\mathcal{V}_c(I)$ *denote the set of the values of the c-th component of all vectors in all state spaces* $\mathcal{S}_k$, $1 \leq k \leq n$. *Then the following holds for every instance I. For all coordinates* $1 \leq c \leq \beta$*, the natural logarithm of every value in* $\mathcal{V}_c(I)$ *is bounded by a polynomial* $p_1(n, \log \bar{x})$ *in n and* $\log \bar{x}$. *Equivalently, the length of the binary encoding of every value is polynomially bounded in the input size, Moreover, for any coordinate c with* $D_c = 0$*, the cardinality of* $\mathcal{V}_c(I)$ *is bounded by a polynomial* $p_2(n, \log \bar{x})$ *in n and* $\log \bar{x}$.

A *DP-simple* optimization problem Π is called *DP-benevolent* if there exist a partial order $\preceq_{dom}$, a quasi-linear order $\preceq_{qua}$, and a degree-vector $\bar{D}$, such that its dynamic programming formulation satisfies Conditions A1–A4.

Some Problems in DP-B

Woeginger showed in [56] that the following problems are in *DP-B*.

- Makespan on two identical machines ,$P2||C_{max}$.
- Sum of cubed job completion times on two machines, $P2||\sum C_j^3$
- Total weighted job completion time on two identical machines, $P2||\sum w_j C_j$
- Total completion time on two identical machines with time dependent processing times, $P2|time - dep|\sum C_j$
- Weighted earliness-tardiness about a common non-restrictive due date on a single machine, $1||\sum w_j|C_j|$

- 0/1-knapsack problem
- Weighted number of tardy jobs on a single machine, $1||\sum w_j U_j$
- Batch scheduling to minimize the weighted number of tardy jobs, $1|batch|\sum w_j U_j$
- Makespan of deteriorating jobs on a single machine, $1|deteriorate|C_{max}$
- Total late work on a single machine, $1||\sum V_j$
- Total weighted late work on a single machine, $1||\sum w_j V_j$

Appendix B: $R(\Pi, m) \in DP\text{-}B$ (Proof of Theorem 4)

Let $\bar{D}' = (\bar{D}; 0)$. Thus, if two states are $(\bar{D}', \Delta)$-close then they have the same transition cost. For any pair of states $\bar{S}, \bar{S}' \in \mathbb{N}^{\beta}$ and $c, c' \in \mathbb{N}$, let $\bar{\Upsilon} = (\bar{S}; c)$ and $\bar{\Upsilon}' = (\bar{S}', c')$. We now define a partial order on the state space. We say that $\bar{\Upsilon} \preceq'_{dom} \bar{\Upsilon}'$ iff $\bar{S} \preceq_{dom} \bar{S}'$ and $c' \leq c$. Also, $\bar{\Upsilon} \preceq'_{qua} \bar{\Upsilon}'$ iff $\bar{S} \preceq_{qua} \bar{S}'$. We note that the following holds.

Observation 9 *If $\bar{\Upsilon}$ is $(\bar{D}', \Delta)$-close to $\bar{\Upsilon}'$, then $\bar{S}$ is $(\bar{D}, \Delta)$-close to $\bar{S}'$, and $c = c'$.*

We now show that the four conditions given in Appendix A are satisfied for $R(\Pi, m)$.

C.1 For any $\Delta > 1$ and $F' \in \mathcal{F}'$, for any input $\bar{Y} = (\bar{X}; \bar{R}) \in \mathbb{N}^{\alpha'}$, and for any pair of states $\bar{\Upsilon} = (\bar{S}; c)$, $\bar{\Upsilon}' = (\bar{S}', c') \in \mathbb{N}^{\beta'}$:

(i) If $\bar{\Upsilon}$ is $(\bar{D}', \Delta)$-close to $\bar{\Upsilon}'$ and $\bar{\Upsilon} \preceq'_{qua} \bar{\Upsilon}'$ then, by the definition of $(\bar{D}', \Delta)$-closeness, $\bar{S}$ is $(\bar{D}, \Delta)$-close to $\bar{S}'$. In addition, $\bar{\Upsilon} \preceq'_{qua} \bar{\Upsilon}'$ implies that $\bar{S} \preceq_{qua} \bar{S}'$. Now, we consider two sub-cases.

(a) If Condition A.1 (i) (a) holds then we need to show two properties.

- We first show that $F'(\bar{Y}, \bar{\Upsilon}) \preceq'_{qua} F'(\bar{Y}, \bar{\Upsilon}')$. We have that $F(\bar{X}, \bar{S}) \preceq_{qua} F(\bar{X}, \bar{S}')$, therefore, $F(\bar{X}, \bar{S}); (c + r_F) \preceq'_{qua} F(\bar{X}, \bar{S}'); (c + r_F)$. Indeed, the $\preceq'_{qua}$ relation is not affected by the cost component of two states. Moreover, by Observation 9, $c = c'$. By the definition of F', $F'(\bar{X}; \bar{R}, \bar{S}; c) \preceq'_{qua} F'(\bar{X}; \bar{R}, \bar{S}'; c')$, i.e., $F'(\bar{Y}, \bar{\Upsilon}) \preceq'_{qua} F'(\bar{Y}, \bar{\Upsilon}')$.
- We now show that $F'(\bar{Y}, \bar{\Upsilon})$ is $(\bar{D}', \Delta)$-close to $F'(\bar{Y}, \bar{\Upsilon}')$. We have that $F(\bar{X}, \bar{S})$ is $(\bar{D}, \Delta)$-close to $F(\bar{X}, \bar{S}')$. This implies
  Claim 20 $F(\bar{X}, \bar{S}); (c + r_F)$ is $(\bar{D}', \Delta)$-close to $F(\bar{X}, \bar{S}'; (c' + r_F))$.

  The claim holds since $\bar{D}' = (\bar{D}; 0)$. It follows that in the first β components in $\bar{D}'$, $F(\bar{X}, \bar{S})$ and $F(\bar{X}, \bar{S}')$ are close with respect to $\bar{D}$, and in the last coordinate we have equality, since $c + r_F = c' + r_F$ (using Observation 9). By the definition of F', we have that $F(\bar{X}, \bar{S}); (c + r_F)) = F'(\bar{X}; \bar{R}, \bar{S}; c)$, therefore $F'(\bar{X}; \bar{R}, \bar{S}; c)$ is $(\bar{D}', \Delta)$-close to $F'(\bar{X}; \bar{R}, \bar{S}'; c')$, or $F'(\bar{Y}, \bar{\Upsilon})$ is $(\bar{D}', \Delta)$-close to $F'(\bar{Y}, \bar{\Upsilon}')$.

(b) If Condition A.1 (i) (b) holds then we have $F\left(\bar{X}, \bar{S}\right) \preceq_{dom} F\left(\bar{X}, \bar{S}'\right)$. Therefore,

$$F\left(\bar{X}, \bar{S}; (c + r_F)\right) \preceq'_{dom} F\left(\bar{X}, \bar{S}'\right); (c + r_F). \tag{4}$$

This follows from the fact that $c' = c$ and from the definition of $\preceq'_{dom}$. By the definition of F', we get that

$$F'\left(\bar{X}; \bar{R}, \bar{S}; c\right) \preceq'_{dom} F'\left(\bar{X}; \bar{R}, \bar{S}'; c'\right), \tag{5}$$

or

$$F'\left(\bar{Y}, \bar{\Upsilon}\right) \preceq'_{dom} F'\left(\bar{Y}, \bar{\Upsilon}'\right). \tag{6}$$

(ii) If $\bar{\Upsilon} \preceq_{dom} \bar{\Upsilon}'$ then, by the definition of $\preceq'_{dom}$, we have that $\bar{S} \preceq_{dom} \bar{S}'$. Hence, we get (4), (5) and (6).

C.2 For any $\Delta \geq 1$, $H'_F \in \mathcal{H}'$, $\bar{Y} = \bar{X}; \bar{R} \in \mathbb{N}^{\alpha} \times \{0,1\}^{|\mathcal{F}|}$ and any pair of states $\bar{\Upsilon} = \bar{S}; c$ and $\bar{\Upsilon}' = \bar{S}'; c' \in \mathbb{N}^{\beta'}$, we show that the following two properties hold.

(i) If $\bar{\Upsilon}$ is $(\bar{D}', \Delta)$-close to $\bar{\Upsilon}'$ and $\bar{\Upsilon} \preceq'_{qua} \bar{\Upsilon}'$ then we need to show that $H'_F(\bar{Y}, \bar{\Upsilon}') \leq H'_F(\bar{Y}, \bar{\Upsilon})$. We distinguish between two cases.

(a) If $c + r_F > m$ then $H'_F\left(\bar{Y}, \bar{\Upsilon}'\right) \leq H'_F\left(\bar{Y}, \bar{\Upsilon}\right) = 1$.

(b) If $c + r_F \leq m$ then since $\bar{S}$ is $\left(\bar{D}, \Delta\right)$-close to $\bar{S}'$, $c = c'$ (by Observation 9) and $\bar{S} \preceq'_{qua} \bar{S}'$ (follows immediately from the fact that $\bar{\Upsilon} \preceq'_{qua} \bar{\Upsilon}'$), we have from Condition A.2 (i) that

$$H_F\left(\bar{X}, \bar{S}'\right) \leq H_F\left(\bar{X}, \bar{S}\right). \tag{7}$$

Hence, we have

$$\begin{aligned} H'_F\left(\bar{Y}, \bar{\Upsilon}'\right) &= H'_F\left(\bar{X}; \bar{R}, \bar{S}'; c'\right) = H_F\left(\bar{X}, \bar{S}'\right) \\ &\leq H_F\left(\bar{X}, \bar{S}\right) = H'_F\left(\bar{X}; \bar{R}, \bar{S}; c\right) = H'_F\left(\bar{Y}, \bar{\Upsilon}\right). \end{aligned}$$

The second equality holds since $c + r_F \leq m$. The inequality follows from (7).

(ii) We need to show that if $\bar{\Upsilon} \preceq'_{dom} \bar{\Upsilon}'$ then $H'_F\left(\bar{Y}, \bar{\Upsilon}'\right) \leq H'_F\left(\bar{Y}, \bar{\Upsilon}\right)$. Note that if $\bar{\Upsilon} \preceq'_{dom} \bar{\Upsilon}'$ then, by the definition of the order $\preceq'_{dom}$, it holds that $c' \leq c$ and $\bar{S} \preceq'_{dom} \bar{S}'$. By Condition A.2(ii), it holds that $H_F\left(\bar{X}, \bar{S}'\right) \leq H_F\left(\bar{X}, \bar{S}\right)$. Now, we distinguish between two cases.

(a) If $c + r_F > m$ then recall that $H_F(\cdot) \leq 1$. Thus, $H'_F\left(\bar{Y}, \bar{\Upsilon}'\right) \leq 1 = H'_F\left(\bar{Y}, \bar{\Upsilon}\right)$.

(b) If $c + r_F \leq m$ then $c' + r_F \leq m$. Therefore,

$$\begin{aligned} H'_F\left(\bar{Y}, \bar{\Upsilon}'\right) &= H'_F\left(\bar{X}; \bar{R}, \bar{S}'; c'\right) = H_F\left(\bar{X}, \bar{S}'\right) \\ &\leq H_F\left(\bar{X}, \bar{S}\right) = H'_F\left(\bar{X}; \bar{R}, \bar{S}; c\right) = H'_F\left(\bar{Y}, \bar{\Upsilon}\right). \end{aligned}$$

The second equality follows from the definition of H'_F. The inequality follows from Condition A.2 (ii).

C.3 This condition gives some properties of the function G.

(i) We need to show that there exists an integer $g \geq 0$ (which does not depend on the input), such that for all $\Delta \geq 0$, and for any two states $\bar{\Upsilon}$, $\bar{\Upsilon}'$ satisfying: $\bar{\Upsilon}$ is $(\bar{D}, \Delta)$-close to $\bar{\Upsilon}'$, it holds that

$$G\left(\bar{\Upsilon}\right) \leq G\left(\bar{\Upsilon}'\right) \cdot \Delta^g. \tag{8}$$

Assume that Π is a maximization problem. Intuitively, if we choose $\bar{\Upsilon}'$ instead of $\bar{\Upsilon}$, since $\bar{\Upsilon} \preceq'_{qua} \bar{\Upsilon}'$, then we are still close to the maximum profit, due to (8).

Since $\bar{\Upsilon}$ is $(\bar{D}, \Delta)$-close to $\bar{\Upsilon}'$, by Observation 9, it holds that $c = c'$ and also $\bar{S} \preceq'_{qua} \bar{S}'$. By Condition A.3(i), there exists $g \geq 0$ whose value does not depend on the input, such that $G(\bar{S}) \leq \Delta^g G(\bar{S}')$. We get that

$$\begin{aligned} G'\left(\bar{\Upsilon}\right) &\leq G'\left(\bar{S}; c\right) = G\left(\bar{S}\right) \\ &\leq \Delta^g G\left(\bar{S}'\right) = \Delta^g G\left(\bar{S}'; c'\right) = \Delta^g G'\left(\bar{\Upsilon}'\right). \end{aligned}$$

(ii) We need to show that if $\bar{\Upsilon} \preceq_{dom} \bar{\Upsilon}'$ then $G'(\bar{\Upsilon}') \geq G'(\bar{\Upsilon})$. Note that if $\bar{\Upsilon} \preceq_{dom} \bar{\Upsilon}'$ then it holds that $\bar{S} \preceq_{dom} \bar{S}'$. Therefore, by Condition A.3 (ii), we have that $G(\bar{S}') \geq G(\bar{S})$. Hence,

$$\begin{aligned} G'\left(\bar{\Upsilon}'\right) &= G'\left(\bar{S}'; c'\right) = G\left(\bar{S}'\right) \\ &\geq G\left(\bar{S}\right) = G'\left(\bar{S}; c\right) = G'\left(\bar{\Upsilon}\right). \end{aligned}$$

C.4 This condition gives several technical properties. Generally, we want to show the size of the table used by the dynamic program DP' is polynomial in the input size.

(i) By the definitions of F', H' and G', they can all be evaluated in polynomial time, based on F, H and G. The relation $\preceq'_{qua}$ can be decided in polynomial time, based on $\preceq_{qua}$. Also, the relation $\preceq'_{dom}$ is polynomial if $\preceq_{dom}$ is.

(ii) We have that $|\mathcal{F}'| = |\mathcal{F}|$, therefore $|\mathcal{F}'|$ is polynomial in n and $\log \bar{x}$.

(iii) Recall that $\mathcal{S}'_0 = \left\{(\bar{S}; 0) | \ \bar{S} \in \mathcal{S}_0\right\}$. Therefore, if $\mathcal{S}_0$ can be computed in time that is polynomial in n and $\log \bar{x}$, the same holds for $\mathcal{S}'_0$.

(iv) Given an instance I_R of $R(\Pi, m)$ and a coordinate $1 \leq j \leq \beta'$, let $\mathcal{V}'_j(I_R)$ denote the set of values of the j-th component of all vectors in the state spaces $\mathcal{S}'_k$, $1 \leq k \leq n$. Then, for any coordinate $1 \leq j \leq \beta$, $\mathcal{V}'_j(I_R) = \mathcal{V}_j(I)$ since, by definition, the first β coordinates in S_k and S'_k are identical. Also, $\mathcal{V}'_{\beta'}(I_R) \subseteq \{\ell \mid \ell \in \mathbb{N}, \ \ell \leq n\}$. This holds since we assume that the cost associated with each transition is in $\{0, 1\}$. Therefore, during the first k phases of DP', the cost per element is bounded by k. In fact, we may assume that the transition costs are polynomially bounded in n. In this case we have that, in phase k, $\left|\mathcal{V}'_{\beta'}(I_R)\right| \leq k \cdot poly(n)$.

This completes the proof of the Theorem. □

References

1. Abu-Khzam, F.N., Egan, J., Fellows, M.R., Rosamond, F.A., Shaw, P.: On the parameterized complexity of dynamic problems. Theor. Comput. Sci. **607**(P3), 426–434 (2015)
2. Amato, G., Cattaneo, G., Italiano, G.F.: Experimental analysis of dynamic minimum spanning tree algorithms. In: Proceedings of 8th ACM-SIAM Symposium on Discrete Algorithms (SODA) (1997)
3. An, H.-C., Bhaskara, A., Chekuri, C., Gupta, S., Madan, V., Svensson, O.: Centrality of trees for capacitated k-center. In: Proceedings of IPCO, pp. 52–63 (2014)
4. Archetti, C., Bertazzi, L., Speranza, M.G.: Reoptimizing the traveling salesman problem. Networks **42**, 154–159 (2003)
5. Archetti, C., Bertazzi, L., Speranza, M.G.: Reoptimizing the 0–1 knapsack problem. Discrete Appl. Math. **158**(17), 1879 (2010)
6. Ausiello, G., Escoffier, B., Monnot, J., Paschos, V.T.: Reoptimization of minimum and maximum traveling salesman's tours. J. Discrete Algorithms **7**(4), 453–463 (2009)
7. Ausiello, G., Bonifaci, V., Escoffier, B.: Complexity and approximation in reoptimization. In: Cooper, B., Sorbi, A. (eds.) Computability in Context: Computation and Logic in the Real World. Imperial College Press/World Scientific, London (2011)
8. Balakrishnan, H., Seshan, S., Katz, R.H.: Improving reliable transport and handoff performance in cellular wireless networks. Wirel. Netw. **1**(4), 469–481 (1995)
9. Baram, G., Tamir, T.: Reoptimization of the minimum total flow-time scheduling problem. Sustain. Comput. Inform. Syst. **4**(4), 241–251 (2014)
10. Bender, M., Farach-Colton, M., Fekete, S., Fineman, J., Gilbert, S.: Reallocation problems in scheduling. In: Proceedings of 22nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 271–279 (2013)
11. Bender, M., Farach-Colton, M., Fekete, S., Fineman, J., Gilbert, S.: Cost-oblivious storage reallocation. In: Proceedings of 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), pp. 278–288 (2014)
12. Bhatia, R., Kodialam, M., Lakshman, T.V.: Fast network re-optimization schemes for MPLS and optical networks. Comput. Netw. **50**(3), 317–331 (2006)
13. Bilo, D., Böckenhauer, H., Komm, D., Královič, R., Mömke, T., Seibert, S., Zych, A.: Reoptimization of the shortest common superstring problem. In: Proceedings of 20th Symposium on Combinatorial Pattern Matching (CPM) (2009)
14. Berger, A., Bonifaci, V., Grandoni, F., Schäfer, G.: Budgeted matching and budgeted matroid intersection via the gasoline puzzle. In: Proceedings of IPCO, pp. 273–287 (2008)
15. Böckenhauer, H.J., Forlizzi, L., Hromkovič, J., Kneis, J., Kupke, J., Proietti, G., Widmayer, P.: On the approximability of TSP on local modifications of optimally solved instances. Algorithmic Oper. Res. **2**(2), 83–93 (2007)
16. Böckenhauer, H.J., Hromkovič, J., Mömke, T., Widmayer, P.: On the hardness of reoptimization. In: Proceedings of 34th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM), pp. 50–65 (2008)
17. Chou, C.F., Golubchik, L., Lui, J.C.S.: A performance study of dynamic replication techniques in continuous media servers. In: Proceedings of International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (IEEE MASCOTS), pp. 256–264 (2000)
18. Demetrescu, C., Finocchi, I., Italiano, G.F.: Dynamic graph algorithms. In: Yellen, J., Gross, J.L. (eds.) Handbook of Graph Theory, Chapter 10.2. Series in discrete mathematics and its applications. CRC Press, Boca Raton (2003)
19. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Springer, London (2013)
20. Drezner, Z., Hamacher, H. (eds.): Facility Location: Applications and Theory. Springer, New York (2002)
21. Dyer, M.E., Frieze, A.M.: A simple heuristic for the p-center problem. Oper. Res. Lett. **3**, 285–288 (1985)
22. Eppstein, D., Galil, Z., Italiano, G.F.: Dynamic graph algorithms. In: Atallah, M.J. (ed.) CRC Handbook of Algorithms and Theory of Computation, Chapter 8. CRC Press, Boca Raton (1999)
23. Escoffier, B., Milanič, M., Paschos, V.T.: Simple and fast reoptimizations for the Steiner tree problem. Algorithmic Oper. Res. **4**(2), 86–94 (2009)
24. Feder, T., Greene, D.H.: Optimal algorithms for approximate clustering. In: Proceedings of 20th Annual ACM Symposium on Theory of Computing (STOC) 434–444 (1988)

25. Frangioni, A., Manca, A.: A computational study of cost reoptimization for min-cost flow problems. INFORMS J. Comput. **18**(1), 61–70 (2006)
26. Gerald, A.: Dynamic Routing in Telecommunication Networks. McGraw-Hill, New York (1997)
27. Golubchik, L., Khanna, S., Khuller, S., Thurimella, R., Zhu, A.: Approximation algorithms for data placement on parallel disks. In: Proceedings of 11th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 223–232 (2000)
28. Graham, R.: Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. **17**, 263–269 (1969)
29. Grandoni, F., Ravi, R., Singh, M., Zenklusen, R.: New approaches to multi-objective optimization. Math. Program. **146**(1–2), 525–554 (2014)
30. Hochbaum, D.S.: Approximation Algorithms for NP-Hard Problems. PWS Publishing, Boston (1995)
31. Hochbaum, D.S., Shmoys, D.B.: A best possible heuristic for the k-center problem. Math. Oper. Res. **10**, 180–184 (1985)
32. Hochbaum, D.S., Shmoys, D.B.: A unified approach to approximation algorithms for bottleneck problems. J. ACM **33**(3), 533–550 (1986)
33. Hulu. http://www.hulu.com/
34. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: a review. ACM Comput. Surv. **31**(3), 264–323 (1999)
35. Karve, A., Kimbrel, T., Pacifici, G., Spreitzer, M., Steinder, M., Sviridenko, M., Tantawi, A.: Dynamic placement for clustered web applications. In: Proceedings of the 15th International Conference on World Wide Web (WWW '06), pp. 595–604. ACM, New York, NY, USA (2006)
36. Kashyap, S.: Algorithms for data placement, reconfiguration and monitoring in storage networks. Ph.D. Dissertation, Computer Science Department, Univ. of Maryland (2007)
37. Kashyap, S., Khuller, S., Wan, Y-C., Golubchik, L.: Fast reconfiguration of data placement in parallel disks. In: Proceedings of the Meeting on Algorithm Engineering & Experiments, pp. 95–107. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA
38. Łacki, J., Oćwieja, J., Pilipczuk, M., Sankowski, P., Zych, A.: The power of dynamic distance oracles: efficient dynamic algorithms for the Steiner tree. In: Proceedings of STOC (2015)
39. Lawler, E.L., Moore, J.M.: A functional equation and its application to resource allocation and sequencing problems. Manag. Sci. **16**, 77–84 (1969)
40. Leon-Garcia, A., Widjaja, I.: Communication Networks. McGraw-Hill, New York (2003)
41. Levin, M.S.: Restructuring in combinatorial optimization. arXiv:1102.1745 (2011)
42. Levin, M.S.: Towards integrated glance to restructuring in combinatorial optimization. arXiv:1512.06427 (2015)
43. Lin, G., Nagarajan, C., Rajaraman, R., Williamson, D.P.: A general approach for incremental approximation and hierarchical clustering. SIAM J. Comput. **39**(8), 3633–3669 (2010)
44. Moore, J.M.: An n-job, one machine sequencing algorithm for minimizing the number of late jobs. Manag. Sci. **15**(1), 102–109 (1968)
45. Nagarajan, V., Schieber, B., Shachnai, H.: The euclidean k-supplier problem. In: Proceedings of 16th International Conference on Integer Programming and Combinatorial Optimization (IPCO) (2013)
46. Nardelli, E., Proietti, G., Widmayer, P.: Swapping a failing edge of a single source shortest paths tree is good and fast. Algorithmica **35**, 56–74 (2003)
47. Neelakanta, P.S.: A Textbook on ATM Telecommunications, Principles and Implementation. CRC Press, Boca Raton (2000)
48. Netflix. http://www.netflix.com/
49. Pallottino, S., Scutella, M.G.: A new algorithm for reoptimizing shortest paths when the arc costs change. Oper. Res. Lett. **31**, 149–160 (2003)
50. Ravi, R., Goemans, M.X.: The constrained minimum spanning tree problem. In: Proceedings of 5th Workshop on Algorithm Theory, pp. 66–75 (1996)
51. Shachnai, H., Tamir, T.: On two class-constrained versions of the multiple knapsack problem. Algorithmica **29**, 442–467 (2001)
52. Shachnai, H., Tamir, G., Tamir, T.: Minimal cost reconfiguration of data placement in storage area network. In: Proceedings of 7rd Workshop on Approximation and Online Algorithms (WAOA) (2009)
53. Shachnai, H., Tamir, G., Tamir, T.: A theory and algorithms for combinatorial reoptimization. In: Proceedings of 10th Latin American Theoretical Informatics symposium (LATIN) (2012)
54. Thiongane, B., Nagih, A., Plateau, G.: Lagrangian heuristics combined with reoptimization for the 0–1 bidimensional knapsack problem. Discrete Appl. Math. **154**(15), 2200–2211 (2006)

55. Thorup, M., Karger, D.R.: Dynamic graph algorithms with applications. In: Proceedings of 7th Scandinavian Workshop on Algorithm (SWAT) (2000)
56. Woeginger, G.J.: When does a dynamic programming formulation guarantee the existence of an FPTAS? In: Proceedings of 10th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 820–829 (1999)
57. Wolf, J.L., Yu, P.S., Shachnai, H.: Disk load balancing for video-on-demand systems. ACM Multimed. Syst. **5**, 358–370 (1997)
58. Yue, F., Tang, J.: A new approach for tree alignment based on local re-optimization. In: Proceedings on International Conference on BioMedical Engineering and Informatics (2008)