A Theory and Algorithms for Combinatorial Reoptimization

Hadas Shachnai[1]*, Gal Tamir[1], and Tami Tamir[2]

[1] Computer Science Department, Technion, Haifa 32000, Israel.
E-mail: {hadas,galtamir}@cs.technion.ac.il.
[2] School of Computer science, The Interdisciplinary Center, Herzliya, Israel. E-mail: tami@idc.ac.il

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 certain distance measure).
In this paper we develop a general model for combinatorial reoptimization, 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. When $\rho = 1$ we get an $(r, 1)$-reoptimization algorithm.
Using our model we derive reoptimization and reapproximation algorithms for several important classes of optimization problems. This includes *fully polynomial time reapproximation schemes* for DP-benevolent problems, a class introduced by Woeginger (*Proc. Tenth ACM-SIAM Symposium on Discrete Algorithms, 1999*), reapproximation algorithms for metric Facility Location problems, 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 minimizing transition costs while guaranteeing a good approximation for the underlying optimization problem.

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, the solution for the new instance must be *close* to the former solution (under certain distance measure).

For example, in a video-on-demand (VoD) system, such as Hulu (*http://www.hulu.com*) or Netflix (*http://www.netflix.com*), movie popularities tend to change frequently. In order to satisfy new client requests, the content of the storage system needs to be modified. The new storage allocation needs to satisfy 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

* Work partially supported by the Technion V.P.R. Fund, by Smoler Research Fund, and by the Ministry of Trade and Industry MAGNET program through the NEGEV Consortium (www.negev-initiative.org).

communication networks, such as Multiprotocol Label Switching (MPLS) [5], or Asynchronous Transfer Mode (ATM) [32], 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.

Solving the above *reoptimization* problems involves two challenges: (*i*) *Computing* an optimal (or close to the optimal) solution for the new instance, and (*ii*) Efficiently *converting* the current solution to the new one.

In this paper we develop a general model for combinatorial reoptimization, encompassing objective functions that combine these two challenges. Our study differs from previous work in two aspects. One aspect is 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 Section 1.1). 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.[3] 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)* [4, 8]).

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 which 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.[4]

Applications: Reoptimization problems naturally arise in many real-life scenarios. Indeed, planned or unanticipated changes occur over time in almost any system. It is then required to respond to these changes quickly and efficiently. Ideally, the response should maintain high performance while affecting only a small portion of the system. In Appendix A we give a detailed description of some of the applications for which our reoptimization model fits well. This includes storage systems for VoD services, communication services and other network problems, stock trading, production planning and vehicle routing.

1.1 Related Work

The work on reoptimization problems started with the analysis of *dynamic graph* problems (see e.g. [17, 38] and a survey in [12]). These works focus on developing data structures supporting update and query operations on graphs. Reoptimization algorithms were developed also for some classic problems on graphs, such as shortest-path [33, 31] and 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 very efficiently, based on the local nature of the updates and on properties of optimal solutions.

[3] As discussed in section 1.1, this is different than multiobjective optimization.
[4] This is similar in nature, e.g., to *incremental optimization* studied in [28].

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*) TSP, in which the modification is a change in the cost of exactly one edge [4, 8], (*ii*) the Steiner Tree problem on weighted graphs, in which possible modifications are insertion/deletion of a vertex, or increase/decrease in the weight of a single edge [9, 16]. (*iii*) the Knapsack problem, in which the modification is an addition of a single element [1], and (*iv*) Pattern Matching problems; for example, reoptimization of the shortest common superstring problem [7], in which a single string is added to the input. A survey of other research in this direction is given in [3].

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 [39] reoptimization is used for efficient multiple sequence alignment - a fundamental problem in bioinformatics and computational biology. In [37], reoptimization in used to improve the performance of a branch-and-bound algorithm for the Knapsack problem.

Other related works consider *multiobjective* 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., [21, 22, 10]). 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 multiobjective optimization carry over to *budgeted* reoptimization. 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 multiobjective 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.

1.2 Our Contribution

We develop (in Section 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 Section 3) the class of DP-benevolent problems introduced by Woeginger [40]. The paper [40] 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)*.[5]

We introduce (in Definition 3) the notion of *fully polynomial time reapproximation scheme (FPTRS)*. Informally, 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

[5] 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.

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 problems admit fully polynomial time $(1+\varepsilon_1, 1+\varepsilon_2)$-reapproximation schemes, for any $\varepsilon_1, \varepsilon_2 > 0$.[6] We note that this is the best possible, unless $P = NP$.

In Section 4 we show how α-approximation algorithms for metric Facility Location problems can be used to obtain $(1, 3\alpha)$-reapproximation algorithms for their reoptimization variants. In Section 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.[7] We conclude (in Section 6) with a discussion of possible directions for future work.

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. Due to space constraints, some of the proofs are given in the Appendix.

2 Combinatorial Reoptimization: Definitions and Notations

In the following we formally define our model for combinatorial reoptimization. Given an optimization problem Π , let I_0 be an input for Π, and let $\mathcal{C}_{\mathcal{I}0} = \{C^1_{I_0}, C^2_{I_0}, \ldots\}$ be the set of configurations corresponding to the solution space of Π for I_0.[8] Each configuration $C^j_{I_0} \in \mathcal{C}_{\mathcal{I}0}$ has some value $val(C^j_{I_0})$. In the reoptimization problem, $R(\Pi)$, we are given a configuration $C^j_{I_0} \in \mathcal{C}_{\mathcal{I}0}$ of an initial instance I_0, and a new instance I derived from I_0 by various 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 seek 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 for 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 for G. For every edge $e \in E_T \setminus E_{T_0}$, we are given the cost $\delta_{add}(e)$ of adding e to the new solution, and for every edge $e \in E \cap (E_{T_0} \setminus E_T)$ 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 Section 5, $R(MST)$ belongs to a class of subset-selection problems that are polynomially solvable.[9]

The input for the reoptimization problem, I_R, contains both the new instance, I, and the transition costs δ (that may be encoded in different ways). Note that I_R does not include the

[6] We list some of these problems in Appendix B.
[7] For the recent theory of fixed-parameter algorithms and parameterized complexity, see, e.g., [14].
[8] A configuration can be any representation of a (partial) solution for Π.
[9] Clearly, if there is a unique MST for G, it suffices to find this MST.

initial configuration I_0 since, apart from determining the transition costs, it has no effect on the reoptimization problem.

2.1 Approximate Reoptimization

When the problem Π is NP-hard, or when the reoptimization problem $R(\Pi)$ is NP-hard,[10] 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 value of $\Pi(I)$ (i.e., the instance I of Π). A configuration $C_I^k \in \mathcal{C}_\mathcal{I}$ yields a ρ-approximation for $\Pi(I)$, for $\rho \geq 1$, if its value is within ratio ρ from $\mathcal{O}(I)$. That is, if Π is a minimization problem then $val(C_I^k) \leq \rho\mathcal{O}(I)$; if Π is a maximization problem then $val(C_I^k) \geq (1/\rho)\mathcal{O}(I)$. 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 $C_I^k \in \mathcal{C}_\mathcal{I}$ 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 combinatorial 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.*

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. This alternative measure in fact helps us achieve our preliminary goal, namely, finding a good approximation for the optimization problem; to that end, we compare the incurred reoptimization cost with a higher optimum. Formally,

Definition 2. *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 $r \cdot \mathcal{O}_R(I_R)$.*

Clearly, any strong (r, ρ)-reapproximation for $R(\Pi)$ is also an (r, ρ)-reapproximation. For $\rho = 1$, we say that an $(r, 1)$-reapproximation algorithm is also an $(r, 1)$-reoptimization algorithm (as it yields an optimal solution). In this case, Definitions 1 and 2 coincide.

Example: We demonstrate the usage of the above definitions by presenting a simple strong $(1, 2 - \frac{1}{m'})$-reapproximation algorithm for the minimum makespan problem, where the allowed modification is removal of machines. Let Π be the minimum makespan scheduling problem [23] (denoted in standard scheduling notation $P||C_{max}$). An instance of Π consists of a set of n jobs

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

and m parallel identical machines (where m is part of the input). 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 $C^\ell_{I_0}$ be a solution of Π for I_0. That is, $C^\ell_{I_0}$ specifies the set of jobs assigned to machine ℓ, $1 \leq \ell \leq m$. Π is a minimization problem and $val(C^\ell_{I_0})$ 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^k_I, the transition cost of a job $j \in I$ from $C^\ell_{I_0}$ to C^k_I is defined to be 0 if j remains on the same machine, and 1 if j is assigned to different machines in $C^\ell_{I_0}$ and C^k_I.

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 after 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-schedule on the whole instance. The transition cost of this algorithm is exactly $|S|$, which is the minimal possible transition cost from $C^\ell_{I_0}$ 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 $(2-\frac{1}{m})$-approximation for the minimum makespan problem [23], for any $m \geq 1$, we get that this algorithm gives a strong $(1, 2-\frac{1}{m'})$-reapproximation for $R(\Pi)$.

Our study encompasses a non-trivial subclass of optimization problems that admit FPTAS. Approximating the reoptimization versions of these problems involves two error parameters, $\varepsilon_1, \varepsilon_2$. This leads to the following extension for the classic definition of FPTAS.

Definition 3. *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 $1/\varepsilon_2$.*

Budgeted Reoptimization: The budgeted reoptimization problem $R(\Pi, m)$ is a restricted version of $R(\Pi)$, in which we add the constraint that the transition cost is at most m, for some budget $m \geq 0$. 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 produced from the initial configuration with transition cost at most m.

Definition 4. *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 $\mathcal{O}(I)$.

3 Reoptimization of DP-benevolent Problems

In this section we consider the class of DP-benevolent problems introduced in [40]. We give a detailed description of this class in Appendix B. For short, we call the class *DP-B*. We show that a non-trivial subclass of DP-B problems admit FPTRS.

Theorem 2. *[40] If an optimization problem Π is DP-benevolent then it has an FPTAS.*

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 Polynomially Bounded Transition Costs

We first consider instances in which the transition costs are polynomially bounded in the input size. In Section 3.2 we show how our results can be extended to instances with arbitrary costs. Let $\mathcal{F}$ be the set of mappings among states corresponding to (partial) solutions for a problem $\Pi \in$ DP-B (see Definition 13). Our first main result is the following.

Theorem 3. *Let $R(\Pi)$ be the reoptimization version of a problem $\Pi \in$DP-B, for which $|\mathcal{F}|$ is fixed, then there exists a fully polynomial time $(1, 1+\varepsilon)$-reapproximation scheme 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.

Recall that $R(\Pi, m)$ is a restricted version of $R(\Pi)$, in which the total transition cost is at most m, for some integer $m \geq 0$, and $\mathcal{O}(I_R, m)$ is the optimal value of $R(\Pi, m)$ for the input I_R. 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 described in Definitions 6 - 9. In the second step, we show that $R(\Pi, m)$ satisfies Conditions B1 – B4.

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

Next, we show that $R(\Pi, m)$ satisfies Conditions B1 – B4.

Theorem 4. *For any $\Pi \in DP$-B, for which $|\mathcal{F}|$ is fixed, and any $m \in \mathbb{N}$, $R(\Pi, m) \in DP$-B.*

Intuitively, budgeted Π is also in DP-B, since the budget induces a new 'knapsack-like dimension' on Π. We give the formal proof in Appendix C.

3.2 Arbitrary Transition Costs

Let I_R be an input for $R(\Pi, m)$, for some integer $m \geq 0$. Given the set of vectors $\bar{X} \in I$ containing the parameters of the input elements, we denote by $\bar{Y} = (\bar{X}; \bar{R})$ the vector corresponding to each vector $\bar{X}$ in the reoptimization instance I_R, where $\bar{R}$ is the transition cost vector associated with $\bar{X}$ in I_R. To obtain approximate solutions for instances with arbitrary transition costs, we first apply a transformation on the cost vector $\bar{R}$.

Definition 5. *Given an input I_R for $R(\Pi)$, let γ be a rounding function that accepts as input the cost vector $\bar{R}$ and the parameters $n, m \in \mathbb{N}$ and $\varepsilon \in (0, 1)$, then*

$$\gamma(\mathbf{r}, n, m, \varepsilon) = \left(\left\lfloor \frac{r_{F_1} \cdot n}{m \cdot \varepsilon} \right\rfloor, \left\lfloor \frac{r_{F_2} \cdot n}{m \cdot \varepsilon} \right\rfloor, \ldots\right).$$

Now, given an input I_R for $R(\Pi, m)$, we modify each element $\bar{Y} = (\bar{X}; \bar{R}) \in I_R$ to $\bar{Y}' = (\bar{X}; \gamma(\bar{R}, n, m, \varepsilon)$. Denote the rounded instance by $\hat{I}_{R,\varepsilon}$. Since the transition costs are rounded down, it holds that $\mathcal{O}(I_R, m) \leq \mathcal{O}(\hat{I}_{R,\varepsilon}, m)$.

Our main result for optimization problems in DP-B is the following.

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

We omit the proof (that is similar to the proof of Theorem 3).

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 minimization 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.

We now summarize the steps of the algorithm for finding a $(1+\varepsilon_1, 1+\varepsilon_2)$-reapproximation for $R(\Pi)$, where $\Pi \in$DP-B. Let DP' be the dynamic program for $R(\Pi, m) \in$ DP-B, and 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.

1. Given an input I_R for $R(\Pi)$ and $\varepsilon_1, \varepsilon_2 > 0$, compute the rounded instance $\hat{I}_{R,\varepsilon_1}$.
2. Let $\varepsilon_2 = \varepsilon_2/2$.
3. Find $\ell^* = \ell(R(\Pi), I_R, \varepsilon_1, \varepsilon_2)$.
4. Return a state in DP' corresponding to a solution for $R(\Pi)$ whose value is $T(R(\Pi, \ell^*), I_R, \varepsilon_1, \varepsilon_2)$.

Note that ℓ^* can be found in polynomial time (e.g., by using a binary search); $T(R(\Pi, \ell^*), I_R, \varepsilon_1, \varepsilon_2)$ can also be computed in polynomial time.

In Appendix B we give a list of problems in DP-B that admit an FPTRS. Also, Appendix B.2 gives an example showing how our framework is applied to the 0/1-knapsack problem.

4 Reoptimization of Metric Facility Location

In this section we show how approximation algorithms for classic network design problems can be used to obtain reapproximation algorithms with similar performance ratios and the minimum reoptimization cost. We exemplify this on the Center Selection problem. The input is a set of n sites $s_1, \ldots, s_n$ in a metric space. The goal is to select the locations of k centers (on the plane) so that the maximum distance from a site to its nearest center is minimized. Let $\Pi(I_0)$ be the Center Selection problem over an instance I_0. In the reoptimization problem, the instance I_0 is modified. The change can involve insertion or deletion of sites, 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 in the reoptimization problem 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 0, while there is a uniform (positive) cost associated with opening a center at any other location. W.l.o.g, we assume a unit opening cost for any new center. Denote by $c(\ell) \geq 0$ the cost of opening a center in location ℓ; when ℓ is the location of a site, s_j, we may use the notation $c(s_j)$. The transition cost from S_0 to S is the sum of the costs of the new centers, i.e., $\sum_{\ell \in S \setminus S_0} c(\ell)$, where ℓ is a location in which we open a center. Suppose that for some $\alpha \geq 1$, we have an α-approximation algorithm for the Center Selection problem, denoted by $\mathcal{A}_{CS}$. We give below a reapproximation algorithm for $R(Center\_Selection)$. We measure the approximation ratio of our algorithm using Definition 2. This is due to the fact that it is NP-hard to obtain a *strong* reapproximation algorithm (see in Section 2.1).

Let $dist(m, \ell)$ denote the distance from location m to location ℓ (where *location* may also be a site). We say that a site s_j is covered if there is a center that is 'close enough' to s_j (see below). Initially, all sites are *uncovered*.

Algorithm $\tilde{\mathcal{A}}_{CS}$ for $R(Center\_Selection)$:

1. Preprocessing step: Use algorithm $\mathcal{A}_{CS}$ to obtain α-approximation for the Center Selection problem with the input I. Let $\hat{d} = D(\mathcal{A}_{CS})$ be the maximal distance from any site to the closest center output by $\mathcal{A}_{CS}$.
2. Let $S = \emptyset$ and $L = \{\ell \mid c(\ell) = 0\}$.
3. Let $U = \{s_1, \ldots, s_n\}$ be the set of uncovered sites.
4. While there exists ($\ell \in L$ and $s_j \in U$ with $dist(s_j\ell) \leq \hat{d}$) do
   (a) $S = S \cup \{\ell\}$.
   (b) For any site s_j with $dist(j, \ell) \leq 3\hat{d}$ do $U = U \setminus \{s_j\}$.
5. $k' = 0$, and set $D(\mathcal{A}_{CS}) = \infty$.
   While $D(\mathcal{A}_{CS}) > \hat{d}$ do
   (a) $k' = k' + 1$
   (b) Run Algorithm A_{CS} with the set of sites U and parameter k'.
6. Let $S_{A_{CS}}$ be the set of centers opened by A_{CS}, then $S \cup S_{A_{CS}}$. Output S.

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

We note that for the case where $\alpha = 2$, we can obtain a better reapproximation ratio for the Center Selection problem. This can be done by modifying algorithm $\tilde{\mathcal{A}}_{CS}$ to use the generic 2-approximation algorithm for Center Selection (see in [24] and [15]). Thus, we have

Theorem 7. *There is a $(1, 4)$-reapproximation algorithm for $R(Center\_Selection)$.*

The above approach can be applied also for deriving a $(1, 3\alpha)$-reapproximation algorithm for the metric k-Median problem, using an α-approximation algorithm. In k-Median, the goal is to select k locations for facilities so as to minimize the sum of the distances of the sites to the nearest facility (see, e.g., in [28] and a comprehensive survey in [36]). The α-approximation algorithm can be used, as before, to find an initial approximation for the optimum sum of distances, as well as for *covering* sites at cost 1. Covering sites at cost 0 is guided by the distances of sites to the closest centers as found in the preprocessing step. The analysis uses ideas similar to those used in the proof of Lemma 3.3 in [28]. (We omit the details.)

5 Optimal Reoptimization of Weighted Subset-Selection

In this section we show that for any subset-selection problem Π over n elements, that 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. 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 $i \in S_0 \setminus S$, and an addition-cost $\delta_{add}(i)$ to be charged if $i \in S \setminus S_0$. The transition cost from S_0 to S is defined as the sum of the corresponding removal- and addition-costs.

A $(1,1)$-reoptimization algorithm for $R(\Pi)$:

(i) Let $\Delta = \max(\max_{i \in S_0 \cap I} \delta_{rem}(i), \max_{i \in I \setminus S_0} \delta_{add}(i))$.
(ii) Let $\lambda = 2|I|\Delta + 1$.
(iii) Define for every $i \in I$ a new weight, $\hat{w}_i$, as follows: for every $i \in S_0 \cap I$, let $\hat{w}_i = \lambda w'_i + \delta_{rem}(i)$. For every $i \in I \setminus S_0$, let $\hat{w}_i = \lambda w'_i - \delta_{add}(i)$.
(iv) Solve $\Pi(I)$ with weights $\hat{w}$.

The proof of the following theorems are given in the Appendix 5.

Theorem 8. *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 $R(\Pi)$.*

The above framework does not fit for problems in which the optimal algorithm is for the objective of finding a subset of minimum (maximum) *cardinality*. Moreover, we show in the Appendix

Theorem 9. *There are polynomially-solvable subset-selection problems whose reoptimization variants are NP-hard.*

6 Discussion

In this paper we developed a general model 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 [18]). 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., [28]). Here, the algorithms gradually 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.

Acknowledgments: We thank Baruch Schieber and Rohit Khandekar for helpful discussions.

References

1. C. Archetti, L. Bertazzi, M.G. Speranza, Reoptimizing the 0-1 knapsack problem, *Discrete applied mathematics*, vol. 158(17), 2010.
2. G. Amato, G. Cattaneo, G. F. Italiano. Experimental analysis of dynamic minimum spanning tree algorithms. *In Proc. 8th ACM-SIAM Annual Symp. on Discrete Algorithms* (SODA), 1997.
3. G. Ausiello, V. Bonifaci, and B. Escoffier, Complexity and approximation in reoptimization. In *Computability in Context: Computation and Logic in the Real World*, B. Cooper, A. Sorbi Eds., Imperial College Press/World Scientific, 2011.
4. G. Ausiello, B. Escoffier, J. Monnot and V. Th. Paschos, Reoptimization of minimum and maximum traveling salesmans tours, *Journal of Discrete Algorithms* 7(4):453-463, 2009.
5. R. Bhatia, M. Kodialam and T.V. Lakshman, Fast network re-optimization schemes for MPLS and optical networks, *Computer Network*, vol. 50:3, 2006.
6. H. Balakrishnan, S. Seshan, R.H. Katz, Improving reliable transport and handoff performance in cellular wireless networks, *Wireless Networks*, vol. 1(4), 1995.
7. D. Bilo, H. Böckenhauer, D. Komm, R. Královič, T. Mömke, S. Seibert, A. Zych. Reoptimization of the shortest common superstring problem. *Symp. on Combinatorial Pattern Matching*, (CPM) 2009.
8. H.J. Böckenhauer, L. Forlizzi, J. Hromkovič, J. Kneis, J. Kupke, G. Proietti, P. Widmayer: On the approximability of TSP on local modifications of optimally solved instances. *Algorithmic Operations Research* 2(2), 2007,
9. H.J. Böckenhauer, J. Hromkovič, T. Mömke, P. Widmayer: On the hardness of reoptimization. Proc. of the 34th *Conference on Theory and Practice of Computer Science* (SOFSEM), pp. 50-65, 2008.
10. A. Berger, V. Bonifaci, F. Grandoni, G. Schäfer, Budgeted matching and budgeted matroid intersection via the gasoline puzzle. In proc. *IPCO 2008*, pp. 273–287, 2008.
11. C.F. Chou, L. Golubchik, and J.C.S. Lui. A performance study of dynamic replication techniques in continuous media servers. *Proc. of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (IEEE MASCOTS)*, pp. 256–264, 2000.
12. C. Demetrescu, I. Finocchi, and G.F. Italiano. Dynamic graph algorithms. *Handbook of Graph Theory*, J. Yellen and J.L. Gross eds., CRC Press Series, in Discrete Math and Its Applications, 2003.
13. Z. Drezner, H. Hamacher (Eds.), Facility location: applications and theory. *Springer*, 2002.
14. R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer-Verlag, New York, 1999.
15. M. E. Dyer and A. M. Frieze. A simple heuristic for the p-center problem. Oper. Res. Lett., 3:285-288, 1985.
16. B. Escoffier, M. Milanič, V. Th. Paschos: Simple and fast reoptimizations for the Steiner tree problem. *DIMACS Technical Report* 2007-01.
17. D. Eppstein, Z. Galil and G.F. Italiano, Dynamic graph algorithms, Chapter 8. in *CRC Handbook of Algorithms and Theory of Computation*, ed. M. J. Atallah, CRC Press, 1999.
18. A. Frangioni and A. Manca. A Computational study of cost reoptimization for min-cost flow problems. *INFORMS Journal on Computing* Volume 18 , Issue 1, 2006.

19. A. Gerald, Dynamic routing in telecommunication networks. *McGraw-Hill*, 1997.
20. L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu. Approximation algorithms for data placement on parallel disks. In *Proc. of SODA*, 223–232, 2000.
21. R. Ravi, M. X. Goemans. The constrained minimum spanning tree problem, In *5th Workshop on Algorithm Theory*, 66–75, 1996.
22. F. Grandoni, R. Zenklusen. Optimization with more than one budget. In *Proc. of ESA*, 2010.
23. R. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17:263–269, 1969.
24. D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-center problem. In *Math.of Operations Research*, 10:180-184, 1985.
25. A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi, Dynamic application placement for clustered web applications. In *Proc. of WWW*, 2006.
26. S. Kashyap, S. Khuller, Y-C. Wan and L. Golubchik. Fast reconfiguration of data placement in parallel disks. *ALENEX*, 2006.
27. A. Leon-Garcia and I. Widjaja. Communication networks, *McGraw-Hill*, 2003.
28. G. Lin, C. Nagarajan, R. Rajaraman, and D. P. Williamson. A general approach for incremental approximation and hierarchical clustering. SIAM J. Comput. 39(8): 3633–3669, 2010.
29. E.L. Lawler and J.M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Sci.*, 16:77–84, 1969.
30. J. M. Moore. An n-job, one machine sequencing algorithm for minimizing the number of late jobs. Management Science, vol.15(1), pp. 102-109, 1968.
31. E. Nardelli, G. Proietti, P. Widmayer: Swapping a failing edge of a single source shortest paths tree is good and fast. Algorithmica 35, 2003.
32. P.S. Neelakanta. A textbook on ATM telecommunications, principles and implementation. *CRC Press*. 2000.
33. S. Pallottino and M. G. Scutella; A new algorithm for reoptimizing shortest paths when the arc costs change, *Operations Research Letters*, vol. 31, 2003.
34. H. Shachnai, G. Tamir and T. Tamir, Minimal cost reconfiguration of data placement in storage area network. , The 7rd Workshop on Approximation and Online Algorithms (WAOA), 2009.
35. H. Shachnai and T. Tamir, On two class-constrained versions of the multiple knapsack problem. *Algorithmica*, vol. 29, 442–467, 2001.
36. M. Shindler, Approximation Algorithms for the Metric k-Median Problem. *Masters thesis*, Department of Computer Science, UCLA, 2008.
37. B. Thiongane, A. Nagih and G. Plateau, Lagrangian heuristics combined with reoptimization for the 0-1 bidimensional knapsack problem. *Discrete Appl. Math.*, vol. 154:15, 2006.
38. M. Thorup and D.R. Karger. Dynamic graph algorithms with applications. *In Proc of the 7th Scandinavian workshop on algorithm* (SWAT) 2000.
39. F. Yue, J. Tang, A new approach for tree alignment based on local re-optimization, In Proc of *International Conference on BioMedical Engineering and Informatics*, 2008.
40. G. J. Woeginger. When does a dynamic programming formulation Guarantee the Existence of an FPTAS? In Proc. of *SODA*, pp. 820–829, 1999.

A Applications

In this section we give a sample of the large set of applications in which reoptimization problems naturally arise.

Storage Systems in Video on Demand Services: In a VoD system, such as Hulu (*http://www.hulu.com*) or Netflix (*http://www.netflix.com*), 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, which is the number of files that can reside on it, and (ii) its load capacity, L_j, which is the number of data streams that can be read simultaneously 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*, in which each movie is allocated exactly D_i^0 broadcasts, always exists [34].

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 of reoptimization problem, the storage allocation reoptimization problem is defined as follows:

Let I be an input for the problem, given by the number and capacities of disks, and the number and broadcast demand of movies. Let C_I be a perfect assignment for I. That is, C_I consists of two (assignment and broadcast) matrices, A_I, B_I, such that each movie is allocated broadcasts that perfectly fit its demand. Let I' in an input derived from I 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 to assignment $A_{I'}$. Formally, $cost(C_I, C_{I'}) = \sum_{i,j} s_{i,j} \cdot max(A_{I'}[i,j] - A_I[i,j], 0)$. 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 $cost(C_I, C_{I'}))$ 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 answer the new demands by adding (and deleting) copies of two files. 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 $s_{1,2} + s_{5,1}$.

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

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

$B_{I'}$	1	2
1	0	2
2	0	3
3	1	0
4	0	3
5	9	0
6	0	2

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

The application of storage management has been extensively studied (see, e.g., [20, 35, 25]). 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 goal is to minimize the transition costs [34]. In other works, the reconfiguration should be performed within limited time, using

limited bandwidth [26]. In other works on reconfiguration of data placement, heuristic solutions were investigated through experiments (e.g., [11]).

Communication Services and Other Network Problems: In communication networks such as Multiprotocol Label Switching (MPLS) [5] or Asynchronous Transfer Mode (ATM) [32], 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., [5]). 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.

Communication networks define a wide research area (see a survey in [27]). In particular, routing problems were extensively studied even in a static environment [19]. Rerouting problems were considered in both centralized and distributed systems (e.g., [6, 5]).

Stock Trading: Assume that an investor owns a stock portfolio. The investor has the stock market forecast and would like to change his stock portfolio. There is a commission associated with each 'buy' or 'sell' operation. Thus, the investor would like to switch to a presumably better portfolio while paying minimum commission. This is a reoptimization problem in which the value of a configuration (stock portfolio) is the total invested money plus the forecasted revenue from owned stocks, and the transition cost between two configurations is the total commission to be paid for the transition.

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.

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

Crew Scheduling: Crews are assigned to operate transportation systems, such as a train or an aircraft. 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.

Facility Location: Facilities need to be opened in order to minimize transportation costs [13]. Possible changes include: addition of a new facility, increase or decrease in facility capacity or in client demands.

B The Class of DP-benevolent Problems

In this section we give some definitions and the conditions that a problem Π must satisfy (as given in [40]) to be DP-benevolent (for short, DP-B). The class of DP-B problems contains a large set of problems that admit an $FPTAS$ via dynamic programming.

Definition 6. *Structure of the input for Π.*
Let $\alpha \geq 1$ be a fixed constant. In any instance I of Π, the input is given by n vectors $\{\bar{X}_i \in \mathbb{N}^\alpha | \ 1 \leq i \leq n\}$. Each vector $\bar{X}_i$ consists of α non-negative integers $(x_{1,i}, ..., x_{\alpha,i})$. 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 in Appendix B.2), $\alpha = 2$, since each item i is given by the vector (p_i, w_i), where $p_i > 0$ is its profit, and $w_i > 0$ gives its weight.

Definition 7. *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 $\bar{S} = (s_1, ..., s_\beta) \in N^\beta$. The number β is a positive integer whose value depends on Π, but does not depend on any specific instance of Π.

Intuitively, 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 8. *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, $1 \leq k \leq n$, the state space $\mathcal{S}_k$ is obtained from the state space $\mathcal{S}_{k-1}$ via $\mathcal{S}_k = \{F(\bar{X}_k, \bar{S}) : \ F \in \mathcal{F}, \ \bar{S} \in \mathcal{S}_{k-1}, \ H_F(\bar{X}_k, \bar{S}) \leq 0\}$.

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 $F(\bar{X}_k, \bar{S})$ is feasible if $H_F(\bar{X}_k, \bar{S}) \leq 0$.

The next definition refers to the objective value for the dynamic program. For simplicity we refer only to maximization problems.

Definition 9. *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 $\mathcal{O}(I) = \max\{G(\bar{S}) : \bar{S} \in \mathcal{S}_n\}$.

An optimization problem Π is called *DP-simple* if it can be expressed via a simple dynamic programming formulation DP as described in Definitions 6 - 9.

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

Definition 10. *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 $\Delta^{-D_l} \cdot s_\ell \leq s'_\ell \leq \Delta^{D_l} \cdot s_\ell$ 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 $D_\ell = 0$.

Finally, 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 $\preceq_{dom}$.[11] For example, in the 0/1 Knapsack problem, we say that $\bar{S} \preceq_{dom} \bar{S}'$ if $\bar{S}$ is 'better' than $\bar{S}'$. 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 B1 *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 B2 *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 $\bar{S}, \bar{S}' \in \mathbb{N}^{\beta}$.

(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 $S \preceq_{dom} S'$ then $H(\bar{X}, \bar{S}') \leq H(\bar{X}, \bar{S})$.

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

Condition B3 *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. Let $\bar{x} = \sum_{k=1}^{n} \sum_{i=1}^{\alpha} x_{i,k}$.

Condition B4 *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 B1 – B4.

[11] This is made precise below.

B.1 Some Problems in DP-B

Woeginger showed in [40] that the following problems are in $DP\text{-}B$.

- 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, $1||\sum w_j V_j$

B.2 Example: Reoptimization of the Knapsack problem

The input for the Knapsack problem consists of n pairs of positive integers (p_k, w_k) and s 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.

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

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

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 $H_1(w_k, p_k, s_1, s_2) = s_1 + w_k - W$ 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 $R(\text{Knapsack}, m)$ gets as input n vectors $Y_k = (X_k; R_k) \in \mathbb{N}^2 \times \{0, 1\}^2$, where X_k is the original input vector, and R_k is the transition-cost vector for the k-th item. Specifically, $Y_k = (p_k, w_k, c(F_1, k), c(F_2, k))$, where $c(F_1, k)$ is the cost for placing the k-th item in the knapsack and $c(F_1, k)$ is the cost if not placing the k-th item in the knapsack. 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 $k \in K_0, c(F_1, k) = 0$ 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 $X_k = [p_k, w_k, c(F_1, k), c(F_2, k)]$. A state $S = [s_1, s_2, s_3]$ 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

$$H_1'(w_k, p_k, c(F_1,k), c(F_2,k), s_1, s_2, s_3) = \begin{cases} s_1 + w_k - W & \text{if } s_3 + c(F_1,k) \le m \\ 1 & \text{otherwise} \end{cases}$$

corresponding to F_1' and a function

$$H_2'(w_k, p_k, c(F_1,k), c(F_2,k), s_1, s_2, s_3) = \begin{cases} 0 & \text{if } s_3 + c(F_2,k) \le m \\ 1 & \text{otherwise} \end{cases}$$

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 $\mathcal{S}_0' = \{[0,0,0]\}$.

C $R(\Pi, m) \in DP\text{-}B$ (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 $\bar{\Upsilon}' = (\bar{S}', \bar{c'})$. 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 $c' \le c$. Also, $\bar{\Upsilon} \preceq'_{qua} \bar{\Upsilon}'$ iff $\bar{S} \preceq_{qua} \bar{S}'$. We note that the following holds.

Observation 1 *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 B 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 1, $c = c'$. 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. $F(\bar{X}, \bar{S}); (c + r_F)$ 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 1). 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 $F(\bar{X},\bar{S}) \preceq_{dom} F(\bar{X},\bar{S}')$. Therefore,

$$F(\bar{X},\bar{S};(c+r_F)) \preceq'_{dom} F(\bar{X},\bar{S}');(c+r_F). \tag{1}$$

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

$$F'(\bar{X};\bar{R},\bar{S};c) \preceq'_{dom} F'(\bar{X};\bar{R},\bar{S}';c'), \tag{2}$$

or

$$F'(\bar{Y},\bar{\Upsilon}) \preceq'_{dom} F'(\bar{Y},\bar{\Upsilon}'). \tag{3}$$

(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 (1), (2) and (3).

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 $H'_F(\bar{Y},\bar{\Upsilon}') \leq H'_F(\bar{Y},\bar{\Upsilon}) = 1$.

(b) If $c + r_F \leq m$ then since $\bar{S}$ is $(\bar{D},\Delta)$-close to $\bar{S}'$, $c = c'$ (by Observation 1) 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

$$H_F(\bar{X},\bar{S}') \leq H_F(\bar{X},\bar{S}). \tag{4}$$

Hence, we have

$$H'_F(\bar{Y},\bar{\Upsilon}') = H'_F(\bar{X};\bar{R},\bar{S}';c') = H_F(\bar{X},\bar{S}')$$

$$\leq H_F(\bar{X},\bar{S}) = H'_F(\bar{X};\bar{R},\bar{S};c) = H'_F(\bar{Y},\bar{\Upsilon}).$$

The second equality holds since $c + r_F \leq m$. The inequality follows from (4).

(ii) We need to show that if $\bar{\Upsilon} \preceq'_{dom} \bar{\Upsilon}'$ then $H'_F(\bar{Y},\bar{\Upsilon}') \leq H'_F(\bar{Y},\bar{\Upsilon})$. 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 $H_F(\bar{X},\bar{S}') \leq H_F(\bar{X},\bar{S})$. Now, we distinguish between two cases.

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

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

$$H'_F(\bar{Y},\bar{\Upsilon}') = H'_F(\bar{X};\bar{R},\bar{S}';c') = H_F(\bar{X},\bar{S}')$$

$$\leq H_F(\bar{X},\bar{S}) = H'_F(\bar{X};\bar{R},\bar{S};c) = H'_F(\bar{Y},\bar{\Upsilon}).$$

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

$$G(\bar{\Upsilon}) \leq G(\bar{\Upsilon}') \cdot \Delta^g. \tag{5}$$

Assume that Π is a maximization problems. 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 (5).

Since $\bar{\Upsilon}$ is $(\bar{D}, \Delta)$-close to $\bar{\Upsilon}'$, by Observation 1, 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

$$G'(\bar{\Upsilon}) \leq G'(\bar{S}; c) = G(\bar{S})$$
$$\leq \Delta^g G(\bar{S}') = \Delta^g G(\bar{S}'; c') = \Delta^g G'(\bar{\Upsilon}').$$

(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,

$$G'(\bar{\Upsilon}') = G'(\bar{S}'; c') = G(\bar{S}')$$
$$\geq G(\bar{S}) = G'(\bar{S}; c) = G'(\bar{\Upsilon}).$$

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 $\mathcal{S}'_0 = \{(\bar{S}; 0) |\ \bar{S} \in \mathcal{S}_0\}$. 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, $|\mathcal{V}'_{\beta'}(I_R)| \leq k \cdot poly(n)$.

This completes the proof of the Theorem. □

D Some Proofs

Proof of Theorem 1: We show a reduction from the decision version of Knapsack:
Input: A set of items I each having a size and a profit, a knapsack of size B, and a value P.
Output: Is it possible to pack a subset having total value more than P.

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

(i) The set of items consists of I and an additional item $\hat{i}$ having size B and value ρP. 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. The (r, ρ)-reoptimization 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 I having total value more than P. Therefore, packing item $\hat{i}$ is a ρ-approximation. Thus, there exists a reoptimization with transition cost 0 that achieves a ρ-approximation. For any r, the (r,ρ)-reoptimization algorithm must use transition cost 0. The only possible packing with profit at least ρP and transition cost 0 is a packing of item $\hat{i}$.

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

Proof of Lemma 1: We reformulate Definitions 6− 9 and show that they hold for $R(\Pi, m)$. Recall that α is 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, whose cardinality depends on the problem Π and not on any specific instance of Π. Also, the cost of the transition F for element i is given by $r_{F,i}$. Let $\bar{R}_i = (r_{F_1,i}, r_{F_2,i}, ...)$ 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 6). In any instance I_R of $R(\Pi, m)$, the input is structured into n vectors*

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

That is, any vector $\bar{Y}_i$ consists in the first α coordinates of 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 Definition 6 holds.

Definition 12. *Structure of the dynamic program DP' (reformulation of Definition 7).*
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 $\bar{\Upsilon} = (\bar{S}; c)$, 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 Definition 7 holds, with $\beta' = \beta + 1$.

Definition 13. *Iterative computation of the state space in DP' (reformulation of Definition 8). The set $\mathcal{F}'$ is a set of mappings $\mathbb{N}^\alpha \times \{0,1\}^{|\mathcal{F}|} \times \mathbb{N}^{\beta'} \to \mathbb{N}^{\beta'}$. For any $F \in \mathcal{F}$, there is a corresponding mapping $F' \in \mathcal{F}'$, where*

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

We note that $|\mathcal{F}'| = |\mathcal{F}|$.

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

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

We note that $|\mathcal{H}'| = |\mathcal{H}|$.

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 $\mathcal{S}'_0 = \{\bar{S}' \mid \bar{S}' = (\bar{S}; 0),\ S \in \mathcal{S}_0\}$. We note that $|\mathcal{S}'_0| = |\mathcal{S}_0|$. The state space for phase k of DP' is given by

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

Therefore, Definition 8 holds.

Definition 14. *Objective value in DP' (reformulation of Definition 9.)*
For any $\bar{\Upsilon} = \bar{S}; c \in \mathcal{S}'$, the objective function $G' : \mathbb{N}^{\beta'} \to \mathbb{N}$ is defined by $G'(\bar{\Upsilon}) = G(\bar{S})$. Therefore, Definition 9 holds.

This completes the proof of the lemma. ∎

Proof of Theorem 3: Let $\mathcal{O}_R(I_R) \equiv \mathcal{O}_R(I_R, 1)$ be the minimal transition cost required to get 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*

$$\mathcal{O}_R(I_R) \leq \mu \tag{6}$$

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

Proof. If (6) 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 $R(\Pi, m)$. ∎

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 $T(R(\Pi, m), I_R, \varepsilon) \geq (1 - \varepsilon)T(\Pi, I, \varepsilon)$.*

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

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

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 (6). Formally,

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

$$\ell(R(\Pi), I_R, \varepsilon) \equiv \min\{m \in \mathbb{N} \mid T(R(\Pi, m), I_R, \varepsilon) \geq (1 - \varepsilon)T(\Pi, I, \varepsilon)\} \tag{7}$$

For short, let $\ell^* = \ell(R(\Pi), I_R, \varepsilon)$ be the minimal value found in (7). 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 (7) holds for any $m \geq \mathcal{O}_R(I_R)$. Since we choose the minimum value of m satisfying (7), this gives the claim. ∎

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

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

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

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

We can now proceed with the 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 get 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. ∎

Proof of Theorem 6: Consider an optimal algorithm, OPT, which opens the set of centers S_{OPT}. Let D_{OPT} denote the maximum distance from a site to the nearest center in S_{OPT}. For any site s_j, $1 \leq j \leq n$, select arbitrarily a center $\ell \in S_{OPT}$ of distance at most D_{OPT}. Denote by C_ℓ the set of sites covered by a center $\ell \in S_{OPT}$. We first show that $\tilde{\mathcal{A}}_{CS}$ outputs a feasible solution.

Claim. The number of centers opened by $\tilde{\mathcal{A}}_{CS}$ is at most k.

Proof. Denote by $F \subseteq S$ the set of centers which $\tilde{\mathcal{A}}_{CS}$ opens at cost 0, and let $k_0 = |F|$. We show that there are k_0 distinct centers in $S_{OPT}(F) \subseteq S_{OPT}$ such that the centers in F cover the set of sites $C = \cup_{\ell \in S_{OPT}(F)} C_\ell$. We distinguish between two types of centers in F.

(i) Consider a center $u \in F$ that is opened in a site, s_j. Then there exists a center $\ell \in S_{OPT}$ whose distance from s_j is at most D_{OPT}. By the triangle inequality, any site in C_ℓ is in distance at most $2D_{OPT}$ from u. Thus, u covers all of the sites in C_ℓ with distance at most $2D_{OPT} \leq 2\hat{d}$.

(ii) Suppose that $u \in F$ is opened in some location in L (that is not a site). Then, by Step 5. of $\tilde{\mathcal{A}}_{CS}$, there exists a site s_j of distance at most $\hat{d}$ from u that is uncovered. Also, there is a center $\ell \in S_{OPT}$ at distance at most D_{OPT} from s_j. By the triangle inequality, u covers all the sites in C_ℓ with distance at most $\hat{d} + 2D_{OPT} \leq 3\hat{d}$. It follows that, by Step 5., algorithm $\tilde{\mathcal{A}}_{CS}$ covers the set of sites in C.

Thus, running algorithm $\mathcal{A}_{CS}$ in Step 5. with $k' = k - k_0$, the maximal distance to any covered site will be $D(\mathcal{A}_{CS}) \leq \hat{d}$. Thus, algorithm $\tilde{\mathcal{A}}_{CS}$ will terminate after opening at most k centers.

We now show that the reoptimization cost of $\tilde{\mathcal{A}}_{CS}$ is minimal. Let $C_0(OPT)$ denote the set of sites covered at cost 0 by OPT.

Claim. $\tilde{\mathcal{A}}_{CS}$ covers with cost 0 the set of sites in $C_0(OPT)$.

Proof. Consider a center $\ell in S_{OPT}$, that is opened at cost 0. If $\ell \in S$ then $\tilde{\mathcal{A}}_{CS}$ covers C_ℓ. Suppose then that $\ell \notin S$. We distinguish between two cases.

(i) If ℓ is in a site s_j, then s_j was not selected by $\tilde{\mathcal{A}}_{CS}$ since it was covered by the algorithm (in Step 4(b)), by a center in another site of distance at most $\hat{d}$ from s_j. hence, by the triangle inequality, s_j covers all the sites in C_ℓ with distance at most $D_{OPT} + \hat{d} \leq 2\hat{d}$.

(ii) If ℓ is some location (that is not a site), and it as not selected by the algorithm, then all of the sites in C_ℓ of distance at most $D_{OPT} \leq \hat{d}$ from ℓ, were covered already by the algorithm using centers of cost 0, with distance at most $2\hat{d}$.

Hence, $\tilde{\mathcal{A}}_{CS}$ covers at cost 0 at least all the sites in $C_0(OPT)$. ∎

Proof of Theorem 8: 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 a binary vector $X = (x_1, ..., x_n)$, where x_i indicates if i is in the solution. Let $X_S = (x_1, ..., x_n)$ be the vector representing the solution provided by the algorithm, and let S_X denote the associated set of elements.

Claim. 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 $Y = (y_1, ..., y_n)$ is a vector representing a better solution. Therefore,

$$1 + \sum_{i=1}^{n} w'_i \cdot x_i \leq \sum_{i=1}^{n} w'_i \cdot y_i. \tag{8}$$

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

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

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 $\sum_{i=1}^{n} \hat{w}_i \cdot x_i > \sum_{i=1}^{n} \hat{w}_i \cdot y_i$. 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 Equation 8(multiplied by λ), and the fact that $\lambda = 2n\Delta + 1$, we get the following contradiction:

$$\lambda \sum_{i=1}^{n} w'_i x_i + \lambda > \lambda \sum_{i=1}^{n} w'_i x_i + 2n\Delta \geq \lambda \sum_{i=1}^{n} w'_i y_i \geq \lambda \sum_{i=1}^{n} w'_i x_i + \lambda.$$

Claim. 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. Assume by way of contradiction that $Y = (y_1, ..., y_n)$ is an optimal solution for $\Pi(I)$ with weights w', and its transition cost from S_0 is lower than the transition cost of X_S. We already argued that X_S is an optimal solution for $\Pi(I)$ with weights w', therefore, $\sum_{i=1}^{n} w'_i x_i = \sum_{i=1}^{n} w'_i y_i$, and clearly, $\sum_{i=1}^{n} \lambda w'_i x_i = \sum_{i=1}^{n} \lambda w'_i y_i$.

The total transition cost from S_0 to S_Y is given by $\sum_{i \in S_Y \setminus S_0} \delta_{add}(i) \cdot y_i + \sum_{i \in S_0 \setminus S_Y} \delta_{rem}(i) \cdot y_i$

Note that the definition of $\hat{w}$ assigns a positive component to elements in $S_0 \cap I$, and a negative component to elements in $I \setminus S_0$, thus, for every solution vector X, the total difference between from $\sum_{i=1}^{n} \lambda w'_i x_i$ and $\sum_{i=1}^{n} \hat{w}_i x_i$ is exactly the transition cost from S_0 to S_X. In particular, the optimality of S_X for $\Pi(I)$ with weights $\hat{w}$, implies that the total transition cost from S_0 to S_Y, is higher than the total transition cost from S_0 to S_X.

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

Proof of Theorem 9: 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 maximizes the number of non-late jobs (whose completion time is at most their due-date). This problem can be viewed as a subset-selection problem since 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 [30].

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. Using a reduction from the weighted problem $1||\sum w_j U_j$, which is known to be NP-hard ([29]), we get that the reoptimization problem $R(1||\sum U_j)$ is NP-hard. ■