Algorithms for Combinatorial Reoptimization

Gal Tamir
Algorithms for Combinatorial Reoptimization

Research Thesis

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Gal Tamir

Submitted to the Senate of the Technion — Israel Institute of Technology
Iyar 5776 Haifa June 2016
This research was carried out under the supervision of Prof. Hadas Shachnai, in the Computer Science Department.

The results in this thesis appear in the following papers:


Acknowledgements

I would like to thank my advisor, Prof. Hadas Shachnai, for her help and guidance throughout this research, and for pointing out many interesting new research directions. Hadash has patiently helped me make my first steps as a researcher, providing me with inspiration, freedom and encouragement. I would like to thank my co-author Prof. Tami Tamir, her original ideas and experienced suggestions were a great help to this research. I am especially grateful to Prof. Seffi Naor for his inspiration, support and wise advice throughout all these years; I am also thankful to the administrative staff, and especially to Yardena Kolet for her help and support.

I am deeply indebted to Eilon Bilinsky, for showing me the beauty of mathematics and its great importance in many academic fields. Thank you for the long calls through nights and weekends, and for inspiring me with outstanding logics and views.

I would also like to thank my colleagues at the department, and especially to Ariel Kulik, Eyal Rozenberg, Zohar Karnin, Roy Schwartz, Moran Feldman, Liran Katzir and Shir Ben-Israel, for many helpful and interesting discussions. I hope to keep working with you in the future; you made this experience truly meaningful and enjoyable.

I am thankful to my public office mentors Ori Avi-Noam and Eyal Rozenberg, dear friends who opened my eyes to the importance of mutual care and organized unions while working together at the Graduate Student Organization (GSO) executive committee.

Most importantly, I would like to thank my dear wife Keren for her helpful ideas, brilliances, support, encouragement, patience, and love. It was an absolute pleasure to share this experience with you. Keren, thank you for your friendship, for your great dedication to my success. I would like to thank my parents, Tal and Yuval, and my
sister Hila for their great help and love throughout this journey. I am proud to see you all following your own unique path so successfully and gracefully. Special thanks to my grandparents who always inspired me to do great things. My work would never be the same without you all.

The generous financial support of the Thecnion is gratefully acknowledged.
3.2.1 Analysis ................................................. 31
3.3 Minimal Cost Algorithm for Arbitrary Number of Servers ............ 36
  3.3.1 The Algorithm ...................................... 37
  3.3.2 Analysis ............................................. 39

4 The Reoptimization Model ......................................... 41
  4.1 The Model ............................................... 41
  4.2 Approximate Reoptimization ................................ 42
  4.3 Budgeted Reoptimization ................................... 45

5 Frameworks for Obtaining Reapproximation Algorithms .................. 47
  5.1 Reoptimization of DP-benevolent Problems ....................... 47
    5.1.1 Polynomially Bounded Transition Costs .................. 47
    5.1.2 Arbitrary Transition Costs ............................. 53
    5.1.3 Example: Reoptimization of the Knapsack Problem ........ 55
  5.2 Reapproximation via Budgeted Reoptimization ...................... 56
    5.2.1 Proof of Theorem 5.5 ................................ 58
  5.3 Reoptimization of Subset Selection Problems ..................... 58
    5.3.1 Optimal Reoptimization of Weighted Subset Selection Problems 58
    5.3.2 Reapproximation of Weighted Subset Selection Problems ..... 61

6 Reapproximation Algorithms for Selected Problems ..................... 63
  6.1 Reoptimization of the k-Center Problem ........................ 63
  6.2 Obtaining Reapproximation Algorithms via Our Frameworks ......... 65
    6.2.1 The Surgery Room Allocation Problem .................... 65
    6.2.2 The Cloud Provider Problem ............................ 65
    6.2.3 The Global Cloud Provider Problem ....................... 67

7 Conclusions and Open Problems .................................... 69

Hebrew Abstract .................................................. i
List of Figures

1.1 A perfect placement before (a) and after (b) the popularity change. Bold edges represent changes in storage assignment. ................................................. 10

3.1 The reconfiguration problem induced by Example 2. (a) initial assignment, (b) reconfiguration having cost 8, and (c) an optimal reconfiguration having cost 2. 30

3.2 Algorithm for updating data placement on a set of servers. ............... 31

4.1 The feasible solutions of an input $I$. ............................................... 43

5.1 An illustration of the budgeted functions $OPT_B$, $A_B$, and $r_2 \cdot A_B$ for a given input $I$. ................................................................. 57
Abstract

This work aims to better model the dynamic nature of real-life applications in which the goal is to maximize (or minimize) certain objective function. Traditional combinatorial optimization problems require finding solutions for a single problem instance. However, 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. Moreover, since the transition from one solution to another may incur some cost, a natural goal is to have the solution for the new instance close to the original one (under a certain distance measure).

We first study a reconfiguration problem arising in storage area network, where files are placed on a set of servers. Each server has a storage capacity and a load capacity. We seek to minimize the cost of moving from one file configuration to another, following the frequent changes in demands for these files. This may require moving files from one server to another, as well as modifying the load capacity allocated to each file. We present algorithms that achieve the optimal cost by using servers whose load capacities are increased by $O(1)$.

Motivated by this natural real-life scenario, we then 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, for some $r, \rho \geq 1$, we say that $A$ is an $(r, \rho)$-reapproximation algorithm if it achieves a $\rho$-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 $r = \rho = 1$, we call $A$ a reoptimization algorithm.

In this model we derive reoptimization and 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, a $(1, 6)$-reapproximation algorithm for the metric $k$-Center problem, and a reoptimization algorithm for polynomially solvable subset-selection problems.

We further show that our results can be extended to the reoptimization variants of subset selection problems that are known to be NP-hard, such as real-time scheduling and the maximum generalized assignment problem (GAP), via a non-standard use of Lagrangian relaxation of the underlying optimization problems.
Abbreviations and Notations

\( \mathbb{R} \) : The set of real numbers
\( \mathbb{N} \) : The set of non-negative integers
\( \log(z) \) : The base 2 logarithm of \( z \) \( \log_2(z) \)
\( \ln(z) \) : The natural logarithm of \( z \) \( \log_e(z) \)
; : The vector concatenation operator. If \( \bar{x} \in \mathbb{N}^a \) and \( \bar{y} \in \mathbb{N}^b \) then \( \bar{x};\bar{y} \in \mathbb{N}^{a+b} \)

Reoptimization Notations

\( \Pi \) : An optimization problem, often referred to as the base problem
\( p \) : The profit function of an optimization problem
\( I, I_0 \) : Input for \( \Pi \), usually \( I_0 \) is the initial input and \( I \) is the new one
\( \delta \) : The transition cost function
\( R(\Pi) \) : The reoptimization version of \( \Pi \)
\( I_R \) : The input for \( R(\Pi) \), containing the new input \( I \) and the transition cost \( \delta \)
\( R(\Pi, b) \) : A restricted version of \( R(\Pi) \) with budget \( b \) on total transition cost
\( \mathcal{O}(I_R) \) : The optimal solution value for the input \( I_R \) of the reoptimization problem
\( \mathcal{O}(I) \) : The optimal solution value for the input \( I \) of the base problem

Binary Relations

\( \preceq \) : Binary relations on a set \( Z \)

Binary Relations Properties

- reflexive : if for any \( z \in Z \): \( z \preceq z \)
- symmetric : if for any \( z, z' \in Z \): \( z \preceq z' \) implies \( z' \preceq z \)
- anti-symmetric : if for any \( z, z' \in Z \): \( z \preceq z' \) and \( z' \preceq z \) implies \( z = z' \)
- transitive : if for any \( z, z', z'' \in Z \): \( z \preceq z' \) and \( z' \preceq z'' \) implies \( z \preceq z'' \)

Order

- partial order : if \( \preceq \) is reflexive, anti-symmetric, and transitive
- quasi-order : if \( \preceq \) is reflexive, and transitive
- quasi-linear : if \( \preceq \) is quasi-order and any two elements of \( Z \) are comparable
Chapter 1

Introduction

1.1 Combinatorial Reoptimization

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 often 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 [hul] or Netflix [net], 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 has 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) [BKL06, HW11, SNS+13], Asynchronous Transfer Mode (ATM) [Nee00, Bla11], or Software Defined Networks (SDN) [RCK+12, TGG+12, MRF+13], 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:

1. **Computing** an optimal (or close to the optimal) solution for the new instance.

2. 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 Section 1.2).

In this thesis 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 Section 1.4). 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. 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)* [BMP13, BFH+07]).

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.

### 1.2 Applications

We describe below a few of the many applications where reoptimization problems naturally show up.

**Communication Services and Other Network Problems.** In communication networks such as *Multiprotocol Label Switching (MPLS)* [BKL06, HW11, SNS+13] *Asynchronous Transfer Mode (ATM)* [Nee00, Bla11], or *Software Defined Networks (SDN)* [RCK+12, TGG+12, MRF+13], data is directed and carried along

---

1. As discussed in Section 1.4, this is different than multi-objective optimization.
2. This is similar in nature, e.g., to *incremental optimization* studied in [LNRW10].
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 pairs \((i, j)\) for which it generates a routing scheme, using different performance measures (see, e.g., [BKL06]). 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.

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 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.

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.

Facility Location. Facilities need to be opened in order to minimize transportation costs [DH02]. Possible changes include: addition of a new facility, increase or decrease in facility capacity or in client demands.

1.3 Reoptimization Problems

1.3.1 The Minimal Cost Reconfiguration Problem

Video on Demand (VoD) services have become common in library information retrieval, entertainment and commercial applications. In a VoD system, clients are connected through a network to a set of servers which hold a large library of video programs. Each client can choose a program he wishes to view and the time he wishes to view it. The service should be provided within a small latency and guaranteeing an almost constant
transfer rate of the data. The transmission of a movie to a client requires the allocation of unit load capacity (or, a data stream) on a server which holds a copy of the movie.

Since video files are typically large, it is impractical to store copies of all movies on each server. Moreover, as observed in large VoD systems (see, e.g., [GBW97, YZZZ06]), the distribution of accesses to movie files is highly skewed; indeed, only small fraction of the movies are requested frequently, while the vast majority (i.e., more than 80%) of the movies are rarely accessed. Hence, the number of copies held for each movie needs to reflect the frequency of accesses to this movie. The goal is to store the movie files on the servers in a way which enables to satisfy as many client requests as possible, subject to the storage and load capacity constraints of the servers.

Formally, suppose that the system consists of $M$ video program files and $N$ servers. We assume throughout the discussion that all files have the same size. Each movie file $i$, $1 \leq i \leq M$, is associated with a popularity parameter $p_0^i \in (0, 1]$, where $\sum_{i=1}^{M} p_0^i = 1$. Each server $j$, $1 \leq j \leq N$, is characterized by

(i) Its storage capacity, $C_j$, that is the number of files that can reside on it.

(ii) Its load capacity, $L_j$, which is the number of data streams that can be read simultaneously from that server.

For a given popularity vector $\{p_0^1, \ldots, p_0^M\}$, the broadcast demand of file $i$ is $D_0^i = p_0^i L$, where $L = \sum_{j=1}^{N} L_j$ is the total load capacity of the system. The data placement problem is to determine a placement of file copies on the servers and the amount of load capacity assigned to each file copy, so as to maximize the total amount of broadcast demand satisfied by the system. A solution for the placement problem can be represented as two $M \times N$ matrices:

(i) The placement matrix $A$, a $\{0, 1\}$-matrix, $A_{i,j} = 1$ iff a copy of movie file $i$ is stored on server $j$.

(ii) The broadcast matrix $B$. $B_{i,j} \in \{0, 1, \ldots, L_j\}$.

$B_{i,j}$ is the number of broadcasts of movie $i$ transmitted from server $j$.

A legal placement has to satisfy the following conditions:

(i) $A_{i,j} = 0 \Rightarrow B_{i,j} = 0$. Clearly, server $j$ can transmit broadcasts of movie $i$ only if it holds a copy of this movie.

(ii) For each server $j$, $\sum_i B_{i,j} \leq L_j$, that is, the total number of broadcasts transmitted from server $j$ does not exceed its load capacity.

(iii) For each server $j$, $\sum_i A_{i,j} \leq C_j$, that is, the number of files stored on server $j$ does not exceed its storage capacity.

3The broadcast demands are assumed to be integers. Rounded values can be obtained by standard solutions for the apportionment problem [You95].
A placement is perfect if it satisfies the broadcast demands of all movie files. Formally, \( \forall i, \sum_j B_{i,j} = D_i^0 \). Under certain conditions, it is known that a perfect placement always exists (see Section 1.4).

The above static data placement problem captures well the goal of maximizing throughput in periods of time where broadcast requirements remain unchanged.\(^4\) However, in general, new movies are released and may become most popular, while the popularity of the previously hot movies drops. The system should be able to support such changes in the distribution on file popularities. Thus, in order to maintain high throughput, the system needs to adjust the placement of file copies and the allocation of load capacity to these copies. This involves replications and deletions of files. File replications incur significant cost as they require bandwidth and other resources on the source, as well as the destination server. Minimizing this cost is crucial for optimizing system performance.

Our dynamic data placement problem can be formalized as follows. Given a perfect placement of file copies on the servers, with the popularity vector \( \langle p_0^1, \ldots, p_M^0 \rangle \), suppose that the popularity vector changes to \( \langle p_1^1, \ldots, p_M^1 \rangle \), with the corresponding broadcast demands \( \langle D_1^1, \ldots, D_M^1 \rangle \). The reconfiguration problem is to modify the initial data placement to a perfect placement for \( \langle D_1, \ldots, D_M \rangle \) at minimum total cost. In updating system configuration, the cost of storing a new copy of movie file \( i \) on server \( j \) is given by \( s_{i,j} \), while the assignment of load capacity to existing copy of file \( i \) on server \( j \) is free. We denote by \( c_{i,j} \) the cost of having a copy of movie \( i \) on server \( j \) after the reconfiguration. Given the initial placement matrix \( A \), we denote by \( A' \) the placement after reconfiguration. Then, by definition, \( c_{i,j} = 0 \) if \( A_{i,j} = 1 \), and \( c_{i,j} = s_{i,j} \) if \( A_{i,j} = 0 \) and \( A'_{i,j} = 1 \). In other words, the cost of increasing the \( (i,j) \)-entry in the assignment matrix, \( A \), is \( s_{i,j} \) while changes in the broadcast matrix \( B \) are free. The total cost of switching from a placement \( A \) to a placement \( A' \) is given by \( \sum_{i,j} c_{i,j} \). Note that file deletions incur no cost. Clearly, the new assignment must satisfy the three legal-placement conditions.

A VoD system is homogeneous if all servers have the same load capacities, i.e., \( L_1 = \cdots = L_N = L \), and the same storage capacities, i.e., \( C_1 = \cdots = C_N = C \) (see, e.g., [GKK+09, KK06]). We assume below that the system is semi-homogeneous, i.e., all servers have the same load capacities, but may have arbitrary storage capacities.

**Example 1.1.** Consider a system of two servers which holds 6 movies. The popularity vector is \( \langle 0.05, 0.05, 0.15, 0.05, 0.1 \rangle \). Both servers have the same load capacity \( L_1 = L_2 = 10 \), while the storage capacities are \( C_1 = 3, C_2 = 4 \). Having \( L = 20 \), the demand vector is \( D^0 = \langle 1, 12, 1, 3, 1, 2 \rangle \). Figure 1.1(a) presents a possible perfect placement for this instance. The assignment is described by a bipartite graph, in which the left hand side nodes represent movie files and the right hand side nodes represent servers; an edge \( (i, j) \) implies that a copy of movie file \( i \) is stored on server

---

\(^4\)In VoD system design, this is also known as the static phase [WYS97].
Table 1.1: The assignment and broadcast matrices of the placement before (left) and after (right) the popularity change.

\[
\begin{array}{ccc|ccc|ccc|ccc}
A & s_1 & s_2 & B & s_1 & s_2 & A' & s_1 & s_2 & B' & s_1 & s_2 \\
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 2 \\
2 & 1 & 1 & 2 & 8 & 4 & 2 & 1 & 1 & 2 & 0 & 3 \\
3 & 1 & 0 & 3 & 1 & 0 & 3 & 1 & 0 & 3 & 1 & 0 \\
4 & 0 & 1 & 4 & 0 & 3 & 4 & 0 & 1 & 4 & 0 & 3 \\
5 & 0 & 1 & 5 & 0 & 1 & 5 & 1 & 0 & 5 & 9 & 0 \\
6 & 0 & 1 & 6 & 0 & 2 & 6 & 0 & 1 & 6 & 0 & 2 \\
\end{array}
\]

\(j\). The maximal degree of a server-node is its storage capacity. Assume that the popularity vector is changed to \(\langle 0.1, 0.15, 0.05, 0.15, 0.45, 0.1 \rangle\). Figure 1.1(b) presents a new placement, obtained from the previous one by adding (and deleting) copies of two files. The new placement is perfect for the new demand vector \(D = \langle 2, 3, 1, 3, 9, 2 \rangle\). The corresponding assignment and broadcast matrices are given in Table 1.1. The reconfiguration cost is \(c_{1,2} + c_{5,1}\).

Figure 1.1: A perfect placement before (a) and after (b) the popularity change. Bold edges represent changes in storage assignment.

As mentioned above, our reconfiguration problem is primarily motivated from the constant need for dynamic data placement in VoD systems. The problem shows up also in production planning, as well as in machine scheduling (see a survey in [ST01b]). Suppose that \(M\) tasks are processed by \(N\) machines. Each machine has limited amount of resources and a time interval in which it is active. The resource requirements of the tasks are changing over time. Tasks may need to be reassigned to the machines in order to fit their new requirement. Reassignment of tasks incurs some cost due to migration overhead and the set-up of the machines. The goal is to reassign the tasks to the machine so as to minimize the transition cost. Finally, our problem naturally arises in dynamic placement of clustered Web applications (see, e.g., [KKP+06]). Web applications are dynamically placed on server machines so as to adjust system configuration to the availability of resources. The goal is to maximize the amount of client demands that can be satisfied by the applications while minimizing the number of placement changes.
1.3.2 The Surgery Room Allocation Problem

In a hospital, a surgery room is a vital resource. Operations are scheduled by the severity of patient illness; however, operation schedules tend to change due to sudden changes in patient condition, the arrival of new patients requiring urgent treatment, or the unexpected absence of senior staff members. Schedule changes involve some costs, e.g., due to the need to rearrange the equipment, or to change the staff members taking care of the patients, as well as their individual schedules.

There is also a profit accrued from each operation. Indeed, some operations are more profitable than others, e.g., due to the coverage received from insurance companies, or due to higher charges in case the operation is scheduled after work hours.

Formally, suppose that the initial input, \( I_0 \), consists of \( n_0 \) patients. Each patient \( j \) is associated with a set \( A_{0,j} \) of possible time intervals in which \( j \) can be scheduled for operation. An interval \( I \in A_{0,j} \) is a half open time interval \( [s_0(I), e_0(I)] \), where \( s_0(I) \leq e_0(I) \). Each interval \( I \in A_{0,j} \) is associated with a profit \( p_0(I) \), for all \( 1 \leq j \leq n_0 \).

Let \( S_0 \) be a given operation schedule for \( I_0 \). Consider the input \( I \) derived from \( I_0 \) by adding or removing patients, by changing the possible time intervals for the patients, or the profits associated with the operations. Suppose that \( I \) consists of \( n \) patients; each patient \( j \) has a set of possible time intervals \( A_j \). Each interval \( I \in A_j \) has a profit \( p(I) \geq 0 \) and a transition cost \( \delta(I) \in \mathbb{N} \). This is either the cost of adding \( I \) to the schedule, if \( I \notin S_0 \), or the cost of omitting \( I \) from \( S_0 \). In any feasible schedule \( S \) for \( I \), at most one interval \( I \in A_j \) is selected, for all \( 1 \leq j \leq n \), and the surgery room is occupied by at most one patient at any time \( t \geq 0 \). The goal is to find a feasible schedule that maximizes the total profit, while minimizing the aggregate transition cost. In particular, we want to obtain a \((1, \alpha)\)-reapproximation algorithm for the problem, for some \( \alpha \geq 1 \).

1.3.3 The Cloud Provider Problem

Cloud computing refers to a model of network computing where a program or application runs on a connected server or servers rather than on a local computing device. Nowadays many companies provide a service of cloud computing and cloud storage (see, e.g., Microsoft Azure [azure.microsoft.com], Amazon Web Services [aws.amazon.com], Google Cloud Platform [cloud.google.com], IBM Cloud [www.ibm.com/cloud-computing], and HP Helion Public Cloud [www.hpcloud.com]). The provider delivers an infrastructure that enables clients to perform their own services on remote virtual machines. Users tend to add or eliminate machines depending on their needs. Some cloud providers even enable auto-scale of virtual machines, Examples for this notion can be found in several leading cloud provider sites, such as Microsoft’s Azure:

“Azure’s services can quickly scale up or down to match demand, so you only pay for what you use.”
Hence, the number of virtual machine that run on the system is frequency changing. Consequently, the provider may need to move virtual machines between hypervisors.\(^5\) Each move incurs a cost for the service. The cloud provider wants to maximize its profits while minimizing its cost.

Formally, in the cloud provider problem, there are \( n \) hypervisors, each having computing power of \( C_i \), and \( m \) virtual machines. Each virtual machine \( 1 \leq j \leq m \) requires \( r_j \) computing units and has a profit \( p_j \) and migration cost \( \delta_{i,j} \), for the reassignment of machine \( j \) to server (hypervisor) \( i \).\(^6\) An assignment is a function from a subset of the machines \( X \) to the hypervisor set, \( S : X \subseteq [m] \rightarrow [n] \). A feasible solution for the problem is an assignment \( S \) such that, for each hypervisor \( 1 \leq i \leq n \) the total amount of computing units required for the assignment does not exceed the computing power of hypervisor \( i \), i.e.,

\[
\forall 1 \leq i \leq n : \sum_{\{j | S(j) = i\}} r_j \leq C_i.
\]

The profit of solution \( S \) is

\[
p(S) = \sum_{j \in X} p_j.
\]

The transition cost of \( S \) is

\[
\delta(S) = \sum_{\{i,j | S(j) = i\}} \delta_{i,j}.
\]

The objective is to find a feasible solution of maximal profit with minimal transition cost. In particular, we want to obtain a \((1, \alpha)\)-reapproximation algorithm for the problem, for some \( \alpha \geq 1 \).

1.3.4 The Global Cloud Provider Problem

As the cloud industry grows, and customers demand global availability for cloud services, many companies provide a global distribution of datacenters. The following quote, taken from Microsoft Azure website, shows the importance of this feature to cloud providers companies:

“Azure operates out of 22 regions around the world. Geographic expansion is a priority for Azure because it enables our customers to achieve higher performance and it supports their requirements and preferences regarding data location”.

Another quote supporting this notion (from Amazon Web Services (AWS)):

---

\(^5\)A hypervisor is a piece of computer software, firmware or hardware that creates and runs virtual machines.

\(^6\)For simplicity, we eliminate the initial input, \( I_0 \), and the initial assignment of machines to the servers. They are implicitly reflected by the migration costs.
“We are steadily expanding global infrastructure to help our customers achieve lower latency and higher throughput, and to ensure that their data resides only in the region they specify. As our customers grow their businesses, AWS will continue to provide infrastructure that meets their global requirements.”

In cloud services supporting a geographical distribution of the hypervisors, the profits and resource requirements associated with each virtual machine may depend on the hypervisor to which it is assigned. Formally, in the global cloud provider problem (GCPP), there are $n$ hypervisors, each having computing power of $C_i$, and $m$ virtual machines. Each requires $r_{i,j}$ computing units and has a profit of $p_{i,j}$ when assigned to hypervisor $i$. Also, there is a migration cost of $\delta_{i,j}$ for the reassignment of machine $j$ to server (hypervisors) $i$.

A feasible solution for the problem is an assignment $S : X \subseteq [m] \rightarrow [n]$ such that for each hypervisor $i$, the total amount of computing units required for the assignment does not exceed the computing power of the hypervisor, i.e.,

$$\sum_{\{j|S(j)=i\}} r_{i,j} \leq C_i.$$ 

The profit of solution $S$ is

$$p(S) = \sum_{\{i,j|S(j)=i\}} p_{i,j}.$$ 

The transition cost of $S$ is

$$\delta(S) = \sum_{\{i,j|S(j)=i\}} \delta_{i,j}.$$ 

The goal is to find a feasible solution of maximal profit with minimal transition cost. In particular, we want to obtain a $(1, \alpha)$-reapproximation algorithm for the problem, for some $\alpha \geq 1$.

1.4 Related Work

1.4.1 Combinatorial Reoptimization

The study of reoptimization problems started with the analysis of dynamic graph problems (see e.g. [DEI99, Tho00] and a survey in [CDI03]). 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 [PS03, NPW03] and the minimum spanning tree [ICI97]. 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, in which the modification is a change in the cost of exactly one edge [ABS03, AEMP09, BFH+07],

(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 [BHS11, EMP09],

(iii) The Knapsack problem, in which the modification is an addition of a single element [ABS10]

(iv) Pattern Matching problems: for example, reoptimization of the shortest common superstring problem [BBK+11], in which a single string is added to the input.

A survey of other research in this direction is given in [GAE11]. 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 [YT08], reoptimization is used for efficient multiple sequence alignment — a fundamental problem in bioinformatics and computational biology. In [TNP06], reoptimization is 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., [RG96, GZ10, BBGS11]). 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. However, we focus here 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 4.1), 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 \( k \)-Center problem has been widely studied (see, e.g., [HS85, DF85], and [JMF99, ABC+14] 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., [Hoc97]). The best possible ratio of 2 was obtained in [HS85, DF85]. To the best of our knowledge, the reoptimization version of the problem is studied here for the first time.

there has been some recent work, which adopts our general reoptimization model, as presented in [STT12]. In particular, our framework, which considers the underlying optimization problem, as well as transition costs, has been applied for rescheduling [BT12, BFF+15] and for storage reallocation [BFF+14].

1.4.2 Storage Management for VoD Services

The data placement problem has been extensively studied (see, e.g., [WYS97, GKK+09, KK06, ST12, KKP+06] and a comprehensive survey in [Kas07]). The paper [ST01a] considers the problem of finding a perfect placement of movie files on the servers. The paper shows the hardness of the perfect placement problem and that such a placement always exists, e.g., when \( \sum_{j=1}^{N} C_j \geq M + N - 1 \). The paper [ST01a] also presents an algorithm for the data placement problem, for inputs in which the ratio \( L_j/C_j \) is equal for all \( 1 \leq j \leq N \) (uniform ratio servers). The paper shows that the algorithm achieves a ratio of \( 1 - \frac{1}{1+C_{min}} \) to the optimal, where \( C_{min} = \min_j C_j \). Golubchik et al. gave in [GKK+09] a tighter analysis of this algorithm and showed that it achieves the ratio \( 1 - \frac{1}{(1+\sqrt{C_{min}})^2} \), and that this ratio is optimal for any algorithm for this problem. The paper [GKK+09] also presents a PTAS for the data placement problem with uniform ratio servers. Later papers considered a generalized version of the problem, where files may be of different sizes (see, e.g., [KK06, ST12]).

For the more realistic model, where file popularities may change over time, there has been some earlier work which refers to the resulting data migration problem: Compute an efficient plan for moving data stored on devices (e.g., a set of servers) in a network from one configuration to another. Since the servers are constrained in handling simultaneous transmissions of files, data migration is done in rounds, where each round handles the delivery of a subset of the files to their destinations. Common objective functions are minimizing the makespan of the migration schedule, or the sum of completion times of the servers (see, e.g., [KKW04, Kim03]). The paper [KKP+06] considers a somewhat ‘dual’ reconfiguration problem: the goal is to convert the existing layout to a good new layout (that is part of the solution), using a limited number of migration rounds. Surveys of known results for the data migration problem are given in [KKP+06, GHKS06]. The data migration problem differs from our reconfiguration problem in several ways:

(i) The final configuration is given as part of the input for data migration, while it is part of the solution for our problem.

(ii) In data migration the output is a migration schedule, while no assignment schedule is output when solving the reconfiguration problem,
(iii) In data migration we measure the quality of the migration schedule, while in our problem we measure the cost of the final configuration.

There has been some other work on reconfiguration of data placement, in which heuristic solutions were investigated through experimental studies (e.g., [LLG00, ZX02, DJ04, GLYW08]). The paper [KKP+06] studies a generalization of our reconfiguration problem, in which file deletions incur unit costs, and the files are of arbitrary sizes. The paper presents experimental results for greedy-based heuristics for the problem. We are not aware of earlier theoretical results for the reconfiguration problem.

1.5 Main results

Our first contribution is a comprehensive study of the minimal cost reconfiguration problem. We show (in Chapter 3) that the reconfiguration problem is NP-hard already on very restricted instances. We then develop algorithms that achieve the optimal cost by using servers whose load capacities are increased by \(O(1)\), in particular, by factor \(1 + \delta\) for any small \(0 < \delta < 1\) when the number of servers is fixed, and by factor of \(2 + \varepsilon\) for arbitrary number of servers, for some \(\varepsilon \in [0,1)\).

The main contribution of this thesis is a general model for combinatorial reoptimization that captures many real-life scenarios (see Chapter 4). Using our model, we derive reoptimization and reapproximation algorithms for several important classes of optimization problems. In particular, we consider (in Chapter 5) the class of DP-benevolent problems introduced by Woeginger [Woe01]. The paper [Woe01] 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).

We introduce (in Chapter 4) the notion of fully polynomial time reapproximation scheme (FPTRS). Informally, given an optimization problem \(\Pi\), 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 \(\Pi\) (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 \(\varepsilon_1, \varepsilon_2 > 0\). We note that this is the best possible, unless \(P = NP\).

We also show (in Chapter 5) that for any subset-selection problem \(\Pi\) over \(n\) elements, which can be optimally solved in time \(T(n)\), there is a reoptimization algorithm for the reoptimization version of \(\Pi\), whose running time is \(T(n')\), where \(n'\) is the size of the modified input. This yields a polynomial time reoptimization algorithm for a large set of polynomially solvable problems, as well as for problems that are fixed parameter tractable.\(^7\)

\(^7\)For the recent theory of fixed-parameter algorithms and parameterized complexity, see, e.g., [DF13].
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.

In Chapter 6 we present reapproximation algorithms for the reoptimization versions of several problems. This includes the classic metric $k$-Center, the surgery room allocation problem, and the (global) cloud provider problem.

1.6 Organization of the Thesis

We first introduce (in Chapter 2) some definitions and notation that will be useful in developing the technical results. In Chapter 3 we give hardness and approximation results for the minimum cost reconfiguration problem.

In Chapter 4 we formally define our reoptimization model, and Chapter 5 describes in detail the frameworks that we developed for obtaining reoptimization and reapproximation algorithms. This includes (i) a framework for obtaining an FPTRS for any reoptimization problem in DP-B; (ii) a framework for reapproximation via budgeted reoptimization, and (iii) frameworks for obtaining reoptimization and reapproximation algorithms for the reoptimization versions of a wide class of subset selection problems.

In Chapter 6 we show how our frameworks can be applied to obtain reapproximation algorithms for selected reoptimization problems. We conclude in Chapter 7 with some open problems and possible directions for future research in this area.
Chapter 2

Preliminaries

In this chapter we provide some definitions and notation that will be used for presenting our results. In Section 2.2 we give a detailed overview of the class of DP-benevolent problems introduced in [Woe01]. We study the reoptimization variants of these problems in Chapter 5. In Section 2.3 and 2.4 we discuss the notion of budgeted optimization that is used in Chapter 5 to develop a framework for reoptimization.

2.1 Combinatorial Optimization Problems and Approximation Algorithms

In a combinatorial optimization problem, we seek the best solution over a discrete space of feasible solutions. The goal can be one of minimizing or maximizing an objective function over all feasible solutions.

2.1.1 Algorithm

An algorithm is a set of operations to be performed on an instance in order to obtain a solution for a problem. If the solution output by an algorithm $A$ is the best possible, for any instance $I$ of the problem $\Pi$, then $A$ is called an exact algorithm for $\Pi$; otherwise, $A$ is an approximation algorithm for $\Pi$.

For an instance $I$ of $\Pi$, let $A(I)$ denote the objective value when running algorithm $A$ on $I$, and let $OPT(I)$ denote the optimal objective value. The approximation ratio of $A$ for the instance $I$ is $R_A(I) = \frac{A(I)}{OPT(I)}$, thus, when $\Pi$ is maximization problem $R_A(I) \leq 1$.

2.2 Polynomial Time Approximation Schemes

A polynomial time approximation scheme (PTAS) is an algorithm that takes as input an additional parameter, $\varepsilon > 0$, which determines the desired approximation ratio. As $\varepsilon$ approaches 0, the approximation ratio gets arbitrarily close to 1. The time complexity
of the scheme is polynomial in the input size, but may be exponential in $\frac{1}{\varepsilon}$. This gives a clear trade-off between running time and quality of approximation.

**Definition 2.1.** An approximation scheme for a minimization problem $\Pi$ is an algorithm $A$ which takes as input an instance $I$ of $\Pi$ and an error bound $\varepsilon > 0$, runs in time polynomial in $|I|$ and has approximation ratio $R(I, \varepsilon) \leq (1 + \varepsilon)$. In fact, such an algorithm $A$ is a family of algorithms $A_\varepsilon$ such that for any instance $I$, $R_{A_\varepsilon}(I) \leq (1 + \varepsilon)$.¹

The approximation algorithm $A$ may be deterministic or randomized. In the latter case, the result is a randomized approximation scheme.

Depending on the function $f(|I|, \frac{1}{\varepsilon})$ which gives the running time of the scheme, some schemes are classified as quasi polynomial and others as fully polynomial. In particular, when the running time is $O(n^{\text{polylog}(n)})$ we get a quasi PTAS; when the running time is polynomial in both $|I|$ and $\frac{1}{\varepsilon}$ we get a fully polynomial time approximation scheme (FPTAS).

### 2.2.1 The Class of DP-benevolent Problems

Woeginger [Woe01] identified some interesting properties satisfied by a wide class of problems, all of which admit FPTAS. This class, of DP-benevolent (for short DP-B) problems, contains such classical problems as Knapsack, scheduling on two identical machines to minimize the latest completion time ($P_2||C_{\max}$), and scheduling on two identical machines to minimize the weighted sum of completion times ($P_2||\sum w_j C_j$).

**Theorem 2.1.** [Woe01] If an optimization problem $\Pi$ is DP-benevolent then it has an FPTAS.

We give below some definitions and the conditions that a problem $\Pi$ must satisfy (as given in [Woe01]) to be in DP-B. We use this formulation to show in Chapter 5 that the reoptimization version of all of the problems in the class of DP-B admit fully polynomial time reapproximation schemes (FPTRS).

**Definition 2.2.** Structure of the input for $\Pi$.

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

Intuitively, $\alpha$ 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 Subsection 5.1.3), $\alpha = 2$, 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.

¹The definition for a maximization problem is analogous.
**Definition 2.3.** Structure of the dynamic program $DP$.

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

Intuitively, $\beta$ 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 2.4.** Iterative computation of the state space in $DP$.

The set $F$ is a finite set of mappings $\mathbb{N}^\alpha \times \mathbb{N}^\beta \rightarrow \mathbb{N}^\beta$. The set $H$ is a finite set of mappings $\mathbb{N}^\alpha \times \mathbb{N}^\beta \rightarrow \mathbb{R}$. For every mapping $F \in F$, there is a corresponding mapping $H_F \in H$.

In the initialization phase of $DP$, the state space $S_0$ 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 $S_k$ is obtained from the state space $S_{k-1}$ via $S_k = \{ F(X_k, \bar{S}) : F \in F, \bar{S} \in S_{k-1}, H_F(X_k, \bar{S}) \leq 0 \}$.

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

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

**Definition 2.5.** Objective value in $DP$.

The function $G : \mathbb{N}^\beta \rightarrow \mathbb{N}$ is a non-negative function. The optimal objective value of an instance $I$ of $\Pi$ is $O(I) = \max\{ G(\bar{S}) : \bar{S} \in S_n \}$.

An optimization problem $\Pi$ is called $DP$-simple if it can be expressed via a simple dynamic programming formulation $DP$ as described in Definitions 2.2 - 2.5.

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

**Definition 2.6.** 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 $\ell$ 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 $\preceq_{dom}$. We say that $\bar{S} \preceq_{dom} \bar{S}'$ if $\bar{S}'$ is

---

2This is made precise below.
‘better’ than $\vec{S}$. 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 $\vec{S}$ and $\vec{S}'$. 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 2.1.** Conditions on the function in $\mathcal{F}$. The following holds for any $\Delta > 1$, $F \in \mathcal{F}$ and $\vec{X} \in \mathbb{N}^\alpha$, and for any $\vec{S}, \vec{S}' \in \mathbb{N}^\beta$.

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

2. If $\vec{S} \preceq_{dom} \vec{S}'$ then $F(\vec{X}, \vec{S}) \preceq_{dom} F(\vec{X}, \vec{S}')$.

**Condition 2.2.** Conditions on the functions in $\mathcal{H}$. The following holds for any $\Delta \geq 1$, $H \in \mathcal{H}$ and $\vec{X} \in \mathbb{N}^\alpha$, and for any $\vec{S}, \vec{S}' \in \mathbb{N}^\beta$.

1. If $\vec{S}$ is $(\vec{D}, \Delta)$-close to $\vec{S}'$, and if $\vec{S} \preceq_{qua} \vec{S}'$ then $H(\vec{X}, \vec{S}') \leq H(\vec{X}, \vec{S})$.

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

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

**Condition 2.3.** Conditions on the function $G$.

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

2. For any $\vec{S}, \vec{S}' \in \mathbb{N}^\beta$ with $\vec{S} \preceq_{dom} \vec{S}'$, $G(\vec{S}') \geq G(\vec{S})$.

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 2.4.** Technical properties.

1. 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.

2. The cardinality of $\mathcal{F}$ is polynomially bounded in $n$ and $\log \bar{x}$.

3. For any instance $I$ of $\Pi$, the state space $\mathcal{S}_0$ can be computed in time that is polynomially bounded in $n$ and $\log \bar{x}$.  

22
(iv) For an instance $I$ of $\Pi$, 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 $S_k$, $1 \leq k \leq n$. 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 $\Pi$ is called $DP$-benevolent if there exist a partial order $\succeq_{dom}$, a quasi-linear order $\succeq_{qua}$, and a degree-vector $\bar{D}$, such that its dynamic programming formulation satisfies Conditions 2.1 – 2.4.

**Some Problems in DP-B.** Woeginger showed in [Woe01] that the following problems are in $DP$-B.

- Makespan on two identical machines, $P_2||C_{\text{max}}$.
- Sum of cubed job completion times on two machines, $P_2||\sum C_j^3$
- Total weighted job completion time on two identical machines, $P_2||\sum w_jC_j$
- Total completion time on two identical machines with time dependent processing times, $P_2|\text{time-dep}|\sum C_j$
- 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, $1||\sum w_jU_j$
- Batch scheduling to minimize the weighted number of tardy jobs, $1|\text{batch}|\sum w_jU_j$
- Makespan of deteriorating jobs on a single machine, $1|\text{deteriorate}|C_{\text{max}}$
- Total late work on a single machine, $1||\sum V_j$
- Total weighted late work on a single machine, $1||\sum w_jV_j$

### 2.3 Budgeted Optimization Problems

In the following we list some budgeted optimization problems that will be used (in Chapters 5 and 6) to demonstrate the applicability of our reapproximation techniques.
Budgeted Real Time Scheduling (BRS). The input is a set $A = \{A_1, \ldots, A_m\}$ of activities, where each activity consists of a set of instances; an instance $I \in A_i$ is defined by a half open time interval $[s(I), e(I))$ in which the instance can be scheduled ($s(I)$ is the start time, and $e(I)$ is the end time), a cost $c(I) \in \mathbb{N}$, and a profit $p(I) \in \mathbb{N}$. A schedule is feasible if it contains at most one instance of each activity, and for any $t \geq 0$, at most one instance is scheduled at time $t$. The goal is to find a feasible schedule, in which the total cost of all the scheduled instances is bounded by a given budget $B \in \mathbb{N}$, and the total profit of the scheduled instances is maximized. Budgeted continuous real-time scheduling (BCRS) is a variant of this problem where each instance is associated with a time window $I = [s(I), e(I))$ and length $\ell(I)$. An instance $I$ can be scheduled at any time interval $[\tau, \tau + \ell(I))$, such that $s(I) \leq \tau \leq e(I) - \ell(I))$. The non-budgeted version of this problem is discussed in [BBF+01], among with other variants and specific cases of the problem. BRS and BCRS arise in many scenarios in which we need to schedule activities subject to resource constraints, e.g., storage requirements for the outputs of the activities.

Budgeted Separable Assignment with Linear Constraint (BSAP$L$). We are given $n$ items $A = \{a_1, \ldots, a_n\}$ and $m$ bins, such that bin $i$ has a $d_2$-dimensional capacity $b_i$. Each item $a_j$ has a $d_2$-dimensional size $s_{i,j} \geq 0$, a $d_1$-dimensional cost vector $c_{i,j}$, and a profit $p_{i,j} \geq 0$ that is gained when $a_j$ is assigned to bin $i$. Also, we are given a $d_1$-dimensional budget vector $L$.

We say that a subset of items $S_i \subseteq A$ is a feasible assignment for bin $i$ if $\sum_{a_j \in S_i} s_{i,j} \leq b_i$. We also define the cost and profit of assigning $S_i$ to bin $i$ by $\bar{c}(i, S_i) = \sum_{a_j \in S_i} c_{i,j}$, and $p(i, S_i) = \sum_{a_j \in S_i} p_{i,j}$, respectively. A solution for the problem is a tuple of $m$ disjoint subsets of items $S = (S_1, \ldots, S_m)$, such that each set $S_i$ is a feasible assignment for bin $i$. We define the cost of $S$ by $\bar{c}(S) = \sum_{i=1}^m \bar{c}(i, S_i) = \sum_{i=1}^m \sum_{a_j \in S_i} c_{i,j}$, and its profit by $p(S) = \sum_{i=1}^m p(i, S_i) = \sum_{i=1}^m \sum_{a_j \in S_i} p_{i,j}$. We say that a solution $S$ is feasible if $\bar{c}(S) \leq L$ (its total cost is bounded by $L$). The problem is to find a feasible solution of maximal profit. The non-budgeted variant is the separable assignment problem with linear constraint (SAP$L$), for which the best known approximation ratio is $(1 - e^{-1} - \varepsilon)$, due to [FGMS11].

We note that the special case of SAP$L$ in which $d_2 = 1$ is the well known generalized assignment problem (GAP). For BSAP$L$, Kulik [Kul11] presented a randomized $(1 - e^{-1} - \varepsilon)$-approximation algorithm.

Budgeted Maximum Weight Independent Set (BWIS). Given a budget $B$ and a graph $G = (V, E)$, where each vertex $v \in V$ has an associated profit $p_v$ (or, weight) and associated cost $c_v$, choose a subset $V' \subseteq V$ such that $V'$ is an independent set (i.e., for any $e = (v, u) \in E$, $v \notin V'$ or $u \notin V'$), the total cost of vertices in $V'$, given by $\sum_{v \in V'} c_v$, is bounded by $B$, and the total profit of $V'$, $\sum_{v \in V'} p_v$, is maximized. BWIS is a generalization of the classical maximum independent set (IS) and maximum weight
independent set (WIS) problems.

2.4 Approximation via Lagrangian Relaxation

Lagrangian relaxation is a fundamental technique in combinatorial optimization. It has been used extensively in the design of approximation algorithms for a variety of problems for solving constrained optimization problems. Jain and Vazirani developed in [JV01] a general framework for using Lagrangian relaxation to derive approximation algorithms. This led to efficient approximations for a variety of budgeted problems. Building on the framework of [JV01], the paper [KS08] presents a general result demonstrating the power of Lagrangian relaxation in solving constrained maximization problems of the following form. Given a universe $U$, a weight function $w : U \to \mathbb{R}^+$, a function $f : U \to \mathbb{N}$ and an integer $L \geq 1$, we want to solve

$$\Pi : \max_{s \in U} f(s)$$

subject to: $w(s) \leq L$.

The paper shows how to solve $\Pi$ by finding an efficient solution for the Lagrangian relaxation of $\Pi$, given by

$$\Pi(\lambda) : \max_{s \in U} f(s) - \lambda \cdot w(s),$$

for some $\lambda \geq 0$.

A traditional approach for using Lagrangian relaxation in approximation algorithms is based on initially finding two solutions, $SOL_1$, $SOL_2$, for $\Pi(\lambda_1), \Pi(\lambda_2)$, respectively, for some $\lambda_1, \lambda_2$, such that each of the solutions is an approximation for the corresponding Lagrangian relaxation; while one of these solutions is feasible for $\Pi$ (i.e., satisfies the weight constraint), the other is not. A main challenge is then to find a way to combine $SOL_1$ and $SOL_2$ to a feasible solution that yields a good approximation for $\Pi$.

The following general result allows to obtain a solution for $\Pi$ based on one of the solutions only.

Theorem 2.2. [KS08] For any $\varepsilon > 0$ and $\lambda_1, \lambda_2$ that satisfy $\lambda_2 \leq \lambda_1 \leq \lambda_2 + \varepsilon'$ with $\varepsilon' = \frac{\varepsilon}{\varepsilon + 1}$, let $s_1 = SOL_1$ and $s_2 = SOL_2$ be $r$-approximate solutions for $\Pi(\lambda_1), \Pi(\lambda_2)$, such that $w(s_1) \leq L \leq w(s_2)$. Then for any $\alpha \in [1 - r, 1]$, at least one of the following holds:

1. $f(s_1) \geq \alpha r f(s^*)$

2. $f(s_2) \geq (1 - \alpha - \varepsilon) f(s^*) \frac{w(s_2)}{L}$.

In particular, with appropriate selection of the parameters $\lambda_1, \lambda_2$ in the Lagrangian relaxation, we can obtain solutions $SOL_1$, $SOL_2$ such that one of them yields an efficient
approximation for the original problem $\Pi$. The resulting technique leads to fast and simple approximation algorithms for a wide class of subset selection problems with linear constraints.

**Corollary 2.3.** [KS08] Given a subset selection problem $\Pi$ with a linear constraint, an algorithm $A$ that yields an $r$-approximation for $\Pi(\lambda)$, and $\lambda_{\text{max}}$, such $w(A(\lambda_{\text{max}})) \leq L$, there is an $(\frac{r}{1+r} - \varepsilon)$-approximation algorithm, such that the number of calls of the algorithm to $A$ is polynomial in $\varepsilon$, the size of the universe, $|U|$, and the input size.

In Chapter 5 we use Corollary 2.3 to develop a simple and general framework that enables to obtain $(1, \alpha)$-reapproximation algorithms for a wide class of subset selection problems for $\alpha \in (0,1)$.

### 2.4.1 Applications to Budgeted Subset Selection

In the following we show how the Lagrangian relaxation technique can be applied to yield approximation algorithms for budgeted subset selection problems. We exemplify this on the budgeted real-time scheduling problem. BRS can be interpreted as the following subset selection problem with linear constraint. The universe $U$ consists of all instances associated with the activities $\{A_1, \ldots, A_m\}$. The domain $X$ is the set of all feasible schedules; for any $S \in X$, $f(S)$ is the profit from the instances in $S$, and $w(S)$ is the total cost of the instances in $S$ (note that each instance is associated with specific time interval). The Lagrangian relaxation of this problem is the classic interval scheduling problem discussed in [BBF+01]: the paper gives a $\frac{1}{2}$-approximation algorithm, whose running time is $O(n \log n)$, where $n$ is the total number of instances in the input. Clearly, $p_{\text{max}} = \max_{s \in U} f(s)$ can be used as $\lambda_{\text{max}}$.

By Corollary 2.3, we can find an $(\frac{r}{1+r} - \varepsilon)$-approximation algorithm where $r = \frac{1}{2}$. Thus, we have the following.

**Theorem 2.4.** There is a polynomial time algorithm that yields an approximation ratio of $(\frac{1}{2} - \varepsilon)$ for BRS.

Similar results can be obtained for other budgeted variants of the problems studied in [BBF+01].
Chapter 3

Minimal Cost Reconfiguration of Data Placement in a Storage Area Network

In this chapter we study the minimal cost reconfiguration problem. We first show (in Section 3.1) that the problem is NP-hard, already when the system consists of two servers, with unit reconfiguration costs and very restricted changes in file popularity. In practical scenarios, it is often the case that the new popularity vector has a perfect placement. This occurs, e.g., where the new vector is a permutation of the initial vector, that is, the popularity distribution function remains unchanged. For such scenarios we give in Sections 3.2 and 3.3, algorithms that solve the reconfiguration problem optimally, by using servers whose load capacities are increased by a small constant factor. Specifically, for a fixed number of servers, we give in Section 3.2 an algorithm which accepts as parameter a value $0 < \delta < 1$ and achieves the optimal reconfiguration cost by using servers whose load capacities are $L(1 + \delta)$. The running time of the algorithm depends on the value of $\delta$ (see Section 3.2). For more general inputs, in which the number of servers may be arbitrarily large, we give in Section 3.3 an algorithm that achieves the optimal cost, by using servers whose load capacities are increased by factor of $2 + \varepsilon$, for some $\varepsilon \in [0, 1)$.

Our main approximation technique, applied (in Section 3.3) to general instances of the reconfiguration problem, relies on finding a linear programming relaxation whose optimal (fractional) solution is a lower bound on the optimal solution for our problem, and for which we can apply rounding without increasing the total cost. To find such a relaxation, we iteratively modify the initial linear relaxation for our problem until we obtain a linear program which reduces our problem to job scheduling on unrelated machines. It is worth noting that even though the optimal integral solutions for the programs in this sequence cannot be related to the optimum cost for our problem, it holds that the optimal (fractional) solution for each program is a lower bound for the
optimum cost for our problem.

## 3.1 Hardness Result

We show that the reconfiguration problem is NP-hard even if the system consists of only two servers, and even if popularity changes are limited such that the new popularity vector is a permutation of the previous one. In other words, the popularity distribution function is preserved. We use a reduction from a variant of the subset-sum problem. For a set of integers $X$, let $S_X$ denote the total size of elements in $X$.

**Definition 3.1.** The **smallest subsets with a given difference** problem is defined as follows. Given are two sets of non-negative integers $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$, and an integer $z$. W.l.o.g, $n_X \leq n_Y$. It is known that there exists a subset $Y'' \subseteq Y$ of size $n_X$ satisfying $S_X = S_{Y''} + z$. The goal is to find the smallest integer $k \geq 1$ such that there exist subsets $X', Y'$, where $|X'| = |Y'| = k$, and $S_{X'} = S_{Y'} + z$. Note that such an integer $k$ must exist, since for $k = n_X$, the sets $X' = X, Y' = Y''$ form a solution.

**Example 3.1.** Let $X = \{2, 3, 4, 5\}$, $Y = \{1, 2, 3, 4, 5\}$, and $z = 4$. For $Y'' = \{1, 2, 3, 4\}$ it holds that $|Y''| = n_X = 4$ and $S_X = 14 = S_{Y''} + z$. For this instance, the required $k$ is 1 since there are two subsets of size 1, specifically, $X' = \{5\}, Y' = \{1\}$, such that $S_{X'} = 5 = S_{Y'} + z$.

**Lemma 3.1.** The **smallest subsets with a given difference** problem is NP-hard.

**Proof.** We show that the corresponding decision problem is NP-hard. That is, given $X, Y, z, k$, such that there exists a subset $Y'' \subseteq Y$ of size $n_X$ satisfying $S_X = S_{Y''} + z$, the goal is to decide if there exist subsets $X', Y'$, each having $k$ elements and $S_{X'} = S_{Y'} + z$.

It is easy to verify that the minimization problem is solvable in polynomial time if the decision problem is.

The reduction is from the cardinality subset-sum problem, which is known to be NP-hard [CKPP00]. Given a set $X = x_1, x_2, \ldots, x_{n_X}$ of positive integers, a cardinality constraint $k < n_X$, and a target integer $w$, the goal is to find a subset $X' \subseteq X$ such that $|X'| = k$ and $S_{X'} = w$. W.l.o.g, we assume that

1. $w > k$ (else, a solution can be found by checking if $X$ contains $k$ units)
2. The largest $k$ elements in $X$ have total size at least $w$ (else, the answer is trivially negative)
3. All the elements in $X$ are integers larger than 1 (else, $X, w$ can be scaled)

Given $X, k, w$, an instance for cardinality subset-sum, construct the following instance for the decision problem of smallest subsets with a given difference: $k = k$, $z = w - k$, $X = X$, $Y$ consists of $n_X - 1$ units, and a single element of value $S_X - n_X + 1 - w + k$. Note that the instance is defined properly since, by assumption
1. $z$ is positive, and by assumption

2. The last element in $Y$ is positive

Also, as required, for $Y'' = Y$ we get that $Y$ has a subset of $n_X$ elements of total size $n_X - 1 + S_X - n_X + 1 - w + k = S_X - z$.

**Claim 3.2.** The set $X$ has a subset of $k$ elements having total size $w$ if and only if the answer to the above decision problem of smallest subsets with a given difference is positive.

**Proof.** There are only two types of subsets of $k$ elements in $Y$. The first one has $k$ unit elements. For this type of subset, $X$ has a subset $X'$ of $k$ elements summing up to $w$ iff $S_{X'} = w = k + z = S_{Y'} + z$, that is, iff the required subsets exist. The other type of subset $Y' \subseteq Y$ of $k$ elements consists of the last element and $k - 1$ units. The total size of elements in this subset is $S_X - n_X + 1 - w + k + (k - 1) = S_X - n_X - z + k$. We show that no subset of $k$ elements from $X$ can sum to this value plus $z$. Let $X'$ be a subset of $k$ elements from $X$. There are $n_X - k$ elements in $X \setminus X'$, whose total size is at least $2(n_X - k)$ (by assumption (iii)). Thus, $S_{X'} \leq S_X - 2(n_X - k)$, which is strictly smaller than $S_X - n_X + k = S_{Y'} + z$. This completes the proof of the lemma. □

Based on the hardness of the *smallest subsets with a given difference* problem, we can prove the following.

**Theorem 3.1.** The reconfiguration problem is NP-hard already for a system of two servers with uniform replication costs, and even if popularity changes are restricted such that the new popularity vector is a permutation of the previous one.

**Proof.** We reduce the *smallest subsets with a given difference* problem to a particular instance of the reconfiguration problem. Given $X, Y, z$, such that there exists a subset $Y'' \subseteq Y$ of size $n_X$ satisfying $S_X = S_{Y''} + z$, consider the following instance of reconfiguration: $M = n_X + n_Y$; the demands are $D^0 = (Y'', Y \setminus Y'' \cup X)$, where $Y''$ is a vector consisting of the $n_X$ elements of $Y''$, and $Y \setminus Y'' \cup X$ is a vector of $n_Y$ elements consisting of elements from $Y \setminus Y''$ followed by the elements of $X$. The system has two servers with $C_1 = n_X$, $L_1 = S_X - z$, $C_2 = n_Y$, $L_2 = S_Y + z$. A possible perfect placement is to store the first $n_X$ movie files on the first server, and the remaining $n_Y$ movie files on the second server. The load capacities of the servers satisfy exactly the broadcast demands. Assume further that the demands are changed to be the values of the elements in $X, Y$. Specifically, $D = (x_1, x_2, \ldots, x_{n_X}, y_1, y_2, \ldots, y_{n_Y})$. Note that the total demand of the first $n_X$ movies is increased by $z$, while the total demand of the remaining $n_Y$ movies is decreased by $z$. Finally, let $s_{i,j} = 1$ be the uniform replication cost.
Figure 3.1 depicts the reconfiguration problem induced by Example 2 above. The system consists of $M = 9$ movies and two servers having parameters $C_1 = 4, L_1 = 10, C_2 = 5, L_2 = 19$. Figure 3.1(a) shows a perfect assignment for the demand vector $D^0 = \langle 1, 2, 3, 4, 5, 2, 3, 4, 5 \rangle$. Assume that the popularity changes so that the new demand vector is $D = \langle 2, 3, 4, 5, 1, 2, 3, 4, 5 \rangle$. The guaranteed $Y''$ implies the solution (b) having cost 8. An optimal solution (c) has cost 2.

Since the total storage capacity of the servers is exactly $n_X + n_Y$, in any perfect placement there is exactly one copy of each movie stored on one of the two servers. Thus, the reconfiguration in this case consists of swapping the storage of $2k$ movie files. By definition of the subset problem, it is known that there exists a perfect assignment that can be achieved by swapping $2n_X$ movie files (of $Y''$ and $X$). An optimal reconfiguration swaps the minimal number of files. Thus, an optimal solution for the reconfiguration problem specifies subsets $X' \subseteq X$, and $Y' \subseteq Y$ such that $|X'| = |Y'| = k$, $S_{X'} = S_{Y'} + z$, and $k$ is minimal. The storage capacity is clearly preserved by this swapping. Also, the total demand of the movies assigned to the first server is now $S_{X \setminus X'} + S_{Y'} = S_X - z = L_1$. Similarly, the total demand of the movies assigned to the second server is now $S_{Y \setminus Y'} + S_{X'} = S_Y + z = L_2$. Therefore, the resulting assignment is perfect. In general, any reconfiguration that ends up with a perfect assignment, corresponds to two subsets $X', Y'$ of $X, Y$ respectively, for which $|X'| = |Y'|$ and the difference in the total new demand of the corresponding movies is $z$. We conclude that a solution for the reconfiguration problem induces a solution for the smallest subsets with a given difference problem, implying that the reconfiguration problem is NP-hard. \qed
Reconfiguration Algorithm
For each storage and load allocation to the big movies satisfying \((P_1)\) do:
  Let \(L^b_j\) and \(C^b_j\) be the total load and storage allocation to big movies on server \(j\).
  For all \(1 \leq j \leq N\) do \(L_j := L - L^b_j + 2\delta\) and \(C_j := C_j - C^b_j\).
  Find a minimum cost placement of the small movies on the servers assuming server \(j\) has load capacity \(L_j\) and storage capacity \(C_j\).
  Select a configuration for the big movies which yields minimum total cost.

Figure 3.2: Algorithm for updating data placement on a set of servers.

3.2 Minimal Cost Algorithm for Fixed Number of Servers

In this section we present a polynomial time algorithm which finds a minimal-cost reconfiguration for a semi-homogeneous system, assuming that the number of servers, \(N\), is some fixed constant. The algorithm outputs a placement which achieves the optimal cost and uses servers with load capacities \((1 + 3\delta)L\), for a small parameter \(\delta \in (0, 1]\).

Given the parameter \(0 < \delta \leq 1\), a movie file \(i\) is considered big, if \(D_i \geq \delta L\), else, movie \(i\) is small. Our algorithm handles separately the two types of movies. It produces allocations with the following properties:

\((P_1)\) For any big movie \(i\), and any server \(j\), the broadcast allocation \(B_{i,j}\) of server \(j\) to movie \(i\) is either 0 or at least \(\delta L\) and an integral multiple of \(\delta^2 L\), i.e., \(B_{i,j} = k\delta^2 L\), where \(k\) is an integer in \(\left[\frac{1}{3\delta^2}, \frac{1}{\delta^2}\right]\).

\((P_2)\) Each small movie is stored on a single server, on which it is allocated all of its broadcast demand. Formally, for any small movie \(i\), for a single server \(j\), \(B_{i,j} = D_i\), and for any \(j' \neq j\), \(B_{i,j'} = 0\).

We show below that by allowing a slight increase in the load capacities, we can find in polynomial time a minimal cost reconfiguration satisfying the above properties. In addition, restricting the set of configurations to those that satisfy the properties does not affect the minimal cost. An overview of the algorithm is given in Figure 3.2.

3.2.1 Analysis

In analyzing the algorithm we use the next technical lemmas.

**Lemma 3.3.** Restricting the allocation to one that satisfies \((P_1)\) and \((P_2)\) may require an increase by at most a factor of \((1 + 2\delta)\) in the load capacity, with no change in the reconfiguration cost.

**Proof.** We show that any assignment in which the total broadcast allocation of each server is at most \(L\) can be converted into one which satisfies the two properties. The only changes are in the broadcast matrix, \(B\). The assignment matrix, \(A\), remains unchanged, therefore, the reconfiguration cost is the same.
Consider the servers one after the other. For each server \( j \) and big movie, \( i \) if \( B_{i,j} \geq \delta L \), then round \( B_{i,j} \) up to the next multiple of \( \delta^2 L \). Note that there are at most \( \frac{1}{\delta} \) such movies, thus, the total overflow on any server due to this step is at most \( \delta L \). If \( 0 < B_{i,j} < \delta L \) and the total demand already allocated to movie \( i \) on previously considered servers is less than \( D_i \) then set \( B_{i,j} = \delta L \). Similarly, for the small movies, when server \( j \) is considered, if for some small movie \( i \), \( 0 < B_{i,j} \leq D_i \) and movie \( i \) was not assigned its total demand on a previously considered server, then set \( B_{i,j} = D_i \).

Indeed, the total overflow in the server due to this step might be much larger than \( \delta L \), because the allowed overflow (in this stage) is per big movie and not per server.

Following the above step, the assignment matrix \( A \) remains the same, and both properties \((P_1)\) and \((P_2)\) hold. However, some servers have a big overflow, of more than \( 2\delta L \), some are assigned total load capacity less than \( L \), and the rest are assigned total load capacity between \( L \) and \((1+2\delta)L\) - with an allowed overflow. Denote the above sets of servers `overflowed`, `vacant`, and `fair` respectively. Consider a graph in which the vertices are the servers, and there is a clique for each movie. That is, two servers \( j_1, j_2 \) are connected if for some movie \( i \), \( A_{i,j_1} = A_{i,j_2} = 1 \). A `balancing step` is defined by a path in this graph. The starting server is overflowed, the middle servers are fair, and the last server is vacant. Server \( j_2 \) can follow server \( j_1 \) in the path if \( j_1 \) is overflowed or fair, \( j_2 \) is fair or vacant, and for at least one movie \( i \) shared by the servers, some amount of broadcasts can be migrated from \( j_1 \) to \( j_2 \). If the edge exists due to a small movie \( i \) then \( D_i \) broadcasts can be migrated. If it exists due to a big movie, then some amount between \( \delta L \) and \( 2\delta L \) broadcasts can be migrated. Specifically, the allocation satisfies that \( B_{i,j_1} \geq \delta L \). If \( B_{i,j_1} = \delta L \) or \( B_{i,j_1} \geq 2\delta L \), then exactly \( \delta L \) broadcasts are transmitted. If \( B_{i,j_1} \leq 2\delta L \) then \( B_{i,j_1} \) broadcasts are migrated. This ensures that the resulting assignment also satisfies \((P_1)\). Balancing steps are performed as long as one exists. Note that we do not care about the time complexity of detecting and performing balancing steps, as we only need to prove the existence of an assignment which satisfies \((P_1)\) and \((P_2)\). The following claim completes the proof.

**Claim 3.4.** If no balancing step exists, then all servers are allocating at most \((1+2\delta)L\) broadcasts.

**Proof.** Let \( M(j) \) be the set of movies that are allocated broadcasts from server \( j \), that is, \( i \in M(j) \) if and only if \( B_{i,j} > 0 \). Assume that no balancing step exists, but there exists an overflowed server \( j \). Since the allocation satisfies \((P_1)\) and \((P_2)\), for every \( i \in M(j) \), if \( i \) is big then \( B_{i,j} \geq \delta L \), and if \( i \) is small then \( B_{i,j} = D_i \). Clearly, \( j \) can migrate broadcasts to any of his neighbors in the cliques corresponding to the movies in \( M(j) \). After such a migration is done, any server \( j' \) in each such clique can similarly migrate broadcasts to any server in the cliques corresponding to movies in \( M(j) \cup M(j') \). By repeating this process, we calculate the set of all servers that may be allocated additional broadcasts due to a balancing path starting at server \( j \). Note that this set of servers forms a connected component in the graph, which is servicing some set
of movies. If all the servers in this component are overflowed or fair then the total broadcast allocation to movies stored on this component is larger than their allocation in the given assignment - a contradiction. Otherwise, there must be a vacant server, \( v \), in the component, and by the way the component was formed, there is a balancing path from \( j \) to \( v \).

A single balancing step may involve migrations of up to \( 2\delta L \) broadcasts, as well as migrations of small amounts of broadcasts (\( D_i \) for some small movie \( i \)). Therefore, a fair server may become overflowed during a balancing step; however, this overflowed server will be handled later, by another balancing step. The process must end with no overflowed server since, as shown above, as long as there is an overflowed server, there is also a vacant server that can receive additional broadcasts.

**Lemma 3.5.** The set of possible configurations of copies of the big movies, along with the corresponding allocations of load capacities, has a polynomial size.

**Proof.** Consider the entries corresponding to big movies in the broadcast matrix \( B \). We show that there is a polynomial number of ways to fill these entries. For each server (column in \( B \)) there are at most \( \frac{1}{\delta} \) positive entries in the corresponding column, each has a value \( k\delta^2 L \), for an integer \( k \in \left[ \frac{1}{\delta}, \frac{1}{\delta^2} \right] \). Thus, there are at most \( (M \cdot \frac{1}{\delta^2})^2 \) possible ways to fill each column in \( B \), and \( (\frac{M}{\delta})^N \) possible ways to determine the allocation to big movies. This value is polynomial since \( N \) is fixed.

**Lemma 3.6.** Given a configuration of the big movies on the servers, there exists a polynomial time algorithm which finds for the small movies a placement of minimum cost, fulfilling property \((P_2)\), where each server \( j \) has storage capacity \( C_j \) and load capacity \( L_j(1 + \delta) \).

**Proof.** Let \( R \) denote the set of small movies, and \( M_r = |R| \). Index the small movies \( 1, \ldots, M_r \). Denote by \( x_{i,j} \in \{0,1\} \) an indicator variable for the assignment of movie \( i \) to server \( j \), \( i \in R \) and \( 1 \leq j \leq N \). The costs \( c_{i,j} \) are the given replication costs. Note that once the big movies have been assigned, the servers may have different load capacities; however, by scaling the broadcast requirements of the movies, we may assume, w.l.o.g., that the load capacities of the servers satisfy \( L_1 = \cdots = L_N = \hat{L} \). Specifically, for a movie \( i \) and server \( j \), define \( D_{i,j} = D_i \cdot \hat{L}/L_j \). The assignment will be determined by rounding the solution for the following linear program, \( LP \).

**Claim 3.7.** Given an optimal solution for \( LP \), with the cost \( C \), there exists a polynomial time algorithm which finds an assignment of the small movies to servers of load capacity \( \hat{L}(1 + \delta) \), whose total cost is at most \( C \).

**Proof.** Given an optimal (fractional) solution for \( LP \), we use a rounding technique of Shmoys and Tardos [ST93]. Specifically, we construct a bipartite graph in which server \( j \)
Formally, sort the small movies in non-increasing order by load requirements. Let 
\( G_B = (V \cup U, E) \) be a bipartite graph, where \( U = \{u_i|1 \leq i \leq M_r\} \) represents the set of small movies, and \( V \) is the set of server vertices: \( V = \{v_{j,k}|1 \leq j \leq N, 1 \leq k \leq \sigma_j\} \) 
where \( \sigma_j = \lceil \sum_{i=1}^{M_r} x_{i,j} \rceil \) is the total number of small movies stored on server \( j \). Clearly, \( \sigma_j \leq C_j \). The vertices \( v_{i,1}, \ldots, v_{i,\sigma_i} \) correspond to server \( j, 1 \leq j \leq N \).

The set of edges \( E \) of \( G_B \) is defined as follows. Given the values of \( x_{i,j} \) for \( 1 \leq i \leq M_r, 1 \leq j \leq N \), for any server \( j \):

(i) If \( \sum_{i=1}^{M_r} x_{i,j} \leq 1 \) then there is a single vertex \( v_{i,1} \in V \) corresponding to server \( j \). In this case, for any \( 1 \leq i \leq M_r \) such that \( x_{i,j} \geq 0 \), we add in \( G_B \) an edge \( (u_i, v_{j,1}) \), and set its weight to be \( w(u_i, v_{j,1}) = x_{i,j} \).

(ii) If \( \sum_{i=1}^{M_r} x_{i,j} > 1 \), find the minimum index \( i_1 \) such that \( \sum_{i=1}^{i_1} x_{i,j} \geq 1 \), then \( E \) contains all the edges \( (u_i, v_{j,1}), 1 \leq i \leq i_1 - 1 \) for which \( x_{i,j} > 0 \). For each of these edges set \( w(u_i, v_{j,1}) = x_{i,j} \). Now, add to \( E \) an edge \( (u_{i_1}, v_{j,1}) \), whose weight is \( w(u_{i_1}, v_{j,1}) = 1 - \sum_{i=1}^{i_1-1} w(u_i, v_{j,1}) \). Thus, the sum of weights of the edges incident to \( v_{j,1} \) is exactly 1. If \( \sum_{i=1}^{i_1} x_{i,j} > 1 \) add an edge \( (u_{i_1}, v_{j,2}) \), whose weight is \( w(u_{i_1}, v_{j,2}) = (\sum_{i=1}^{i_1} x_{i,j}) - 1 \). Proceed next to movies with \( i > i_1 \) i.e., those with smaller broadcast requirements on server \( j \). Similar to the above process for \( v_{j,1} \), add edges incident to \( v_{j,2} \), until a total of exactly one movie is assigned to \( v_{j,2} \), and so on. Let \( i' \) be the index of the last movie for which an edge is assigned this way, i.e., \( i' = i_{\sigma_j-1} \). Now, for any \( i > i' \) for which \( x_{i,j} > 0 \) add an edge \( (u_i, v_{j,\sigma_j}) \) and set \( w(u_i, v_{j,\sigma_j}) = x_{i,j} \).

The cost of an edge \( (u_i, v_{j,k}) \) is \( c_{i,j} \), for \( 1 \leq i \leq M_r, 1 \leq j \leq N, \) and \( 1 \leq k \leq \sigma_j \). For each server vertex \( v_{j,k} \), let \( D_{j,k}^{\text{max}} (D_{j,k}^{\text{min}}) \) denote the maximum (minimum) of the

\[
(LP): \begin{align*}
\text{minimize} & \quad \sum_{i=1}^{M_r} \sum_{j=1}^{N} x_{i,j} \cdot c_{i,j} \\
\text{subject to:} & \quad \sum_{i=1}^{M_r} x_{i,j} \cdot D_{i,j} \leq \hat{L} \quad \text{for } 1 \leq j \leq N, \\
& \quad \sum_{i=1}^{M_r} x_{i,j} \leq C_j \quad \text{for } 1 \leq j \leq N, \\
& \quad \sum_{j=1}^{N} x_{i,j} = 1 \quad \text{for } 1 \leq i \leq M_r, \\
& \quad x_{i,j} \geq 0 \quad \text{for } 1 \leq j \leq N, \ 1 \leq i \leq M_r.
\end{align*}
\]
broadcast requirements $D_{i,j}^\text{max}$ corresponding to the edges $(u_i, v_{j,k})$ incident to $v_{j,k}$. Then, for all $1 \leq j \leq N$ and $1 \leq k \leq \sigma_j - 1$,

$$D_{j,k}^\text{min} \geq D_{j,k+1}^\text{max}. \quad (3.1)$$

Recall that a vector $y$ on the edges of a graph is a fractional matching if, for each vertex $w$, the sum of entries of $y$ corresponding to the edges incident to $w$ is at most 1. The fractional matching exactly matches a vertex $w$ if the corresponding sum is equal to 1. A fractional matching is a matching if each entry of $y$ is in $\{0, 1\}$. We note that the weight function on the edges of $G_B$ defines a fractional matching, in which all movie vertices $u_i$, $1 \leq i \leq M_r$, and all server vertices $v_{j,k}$, $1 \leq j \leq N$, $1 \leq k \leq \sigma_j$, are exactly matched. Thus, there exists an integral matching of the same cost that matches all the movie vertices (see, e.g., [Azi12]).

We now summarize the steps of the rounding procedure which assigns the small movies to the servers.

1. Given an optimal solution for LP, form the bipartite graph $G_B$.
2. Find a min-cost (integer) matching $H$ that exactly matches all movie vertices in $G_B$.
3. For each edge $(u_i, v_{j,k}) \in H$ place movie $i$ on server $j$.

We show that the assignment obtained in Step 3. of the algorithm has cost $C$, and that the overall load capacity used on any server is at most $\hat{L}(1 + \delta)$. By the above discussion, the integral matching found in Step 2. has cost $C$. Since the cost of the assignment is equal to the cost of the matching, the solution output by the algorithm has the optimal cost $C$.

Next, we show that the total broadcast requirement assigned on each server is at most $\hat{L}(1 + \delta)$. Consider the movies assigned to server $j$. For any $1 \leq j \leq N$, there are $\sigma_j \leq C_j$ vertices representing server $j$ in $G_B$. Each of these vertices $v_{j,k}$ adds at most one movie file to server $j$ (the movie which corresponds to the edge selected for the matching $H$, among those incident to $v_{j,k}$). Therefore, at most $C_j$ small movies are assigned to server $j$. It follows that the total broadcast requirement of the movies on server $j$ is at most

$$\sum_{k=1}^{\sigma_j} D_{j,k}^\text{max} \leq D_{j,1}^\text{max} + \sum_{k=2}^{\sigma_j} D_{j,k}^\text{max} \leq \delta \hat{L} + \sum_{k=1}^{\sigma_j - 1} D_{j,k}^\text{min}$$

$$\leq \delta \hat{L} + \sum_{k=1}^{\sigma_j} \sum_{i \in \{i | (u_i, v_{j,k}) \in E\}} D_{i,j} \cdot w(u_i, v_{j,k}) = \delta \hat{L} + \sum_{i=1}^{M_r} D_{i,j} x_{i,j} \leq \hat{L}(1 + \delta).$$

The second inequality follows from (3.1) and the fact that $D_{i,j} \leq \delta \hat{L}$ for all $1 \leq i \leq M_r$. This completes the proof.
Combining the above lemmas, we summarize in the next result.

**Theorem 3.2.** Given a system of $N$ servers, each having load capacity $L$ and arbitrary storage capacities $C_j \geq 1, 1 \leq j \leq N$, the Reconfiguration algorithm finds in polynomial time a placement of the files whose cost is optimal, by using servers with load capacities $L(1 + 3\delta)$.

### 3.3 Minimal Cost Algorithm for Arbitrary Number of Servers

In this section we consider a system with arbitrary number of servers. We first show that when the number of movies is fixed, our problem can be optimally solved for any number of servers.

**Theorem 3.3.** The reconfiguration problem is solvable in polynomial time when $M$, the number of movies, is fixed.

**Proof.** Number the collection of subsets of the $M$ movies by $1, \ldots, 2^M$, then the assignment of movie files to the $N$ servers can be represented as an assignment vector of length $2^M$, in which the $k$th entry gives the number of servers that contain the $k$th subset of movies. The number of possible assignment vectors is $\binom{2^M + N - 1}{2^M}$. It is possible to compute the cost of each assignment vector as follows. Construct the bipartite graph $G_B = (U, V, E)$, in which $|U| = |V| = N$. Each vertex $u \in U$ corresponds to a server, and each vertex $v \in V$ corresponds to a subset of movies for which there is a non-zero entry in the assignment vector. In other words, if the $k$th entry in the vector is equal to $h, 1 \leq h \leq N$, then in $G_B$ there are $h$ vertices in $V$ which correspond to the $k$th subset of movies. There is an edge $(u, v) \in E$ if the server corresponding to $u$ has storage capacity larger than the number of movies in the subset that corresponds to $v$; the cost of $(u, v)$ is the cost of assigning this subset of movies on $v$. Next, solve the minimum weight perfect matching problem on $G_B$. The cost of each perfect matching (if one exists) is the cost of the corresponding assignment vector. Given a perfect matching of minimum cost, use a flow network to test the feasibility of the given assignment vector, namely, to verify that each movie $i$ can be allocated its load requirement $D_i$. Finally, among the feasible assignments, select the one having the smallest cost.

For the case where $M$ may be arbitrarily large, we present below a polynomial time algorithm which finds a minimal-cost reconfiguration in a semi-homogeneous system. Given an instance $I$, our algorithm outputs a minimal-cost placement, by using servers of load capacities $(2 + \varepsilon)L$ where

$$\varepsilon = \max\{t \mid D_i > L\}\{D_i/L - \lfloor D_i/L \rfloor\}. \tag{3.2}$$
3.3.1 The Algorithm

The following is an overview of the algorithm.

1. Partition the movies to *big* and *small*: movie *i* is big if \( D_i > L \), else movie *i* is small.

2. Solve a linear programming relaxation to obtain a lower bound on the optimal solution for the reconfiguration problem.

3. Round the (fractional) solution of the linear program to obtain an integral solution of optimal cost.

4. Use the integral solution to assign movie copies to \( N \) servers, where server *j* has storage capacity \( C_j \) and load capacity \((2 + \varepsilon)L\) for some \( \varepsilon \in [0, 1) \) (as defined in Equation 3.2).

**Solving an LP Relaxation.** In the following we show how a natural LP relaxation for our problem can be modified to obtain another linear program, from which we derive an optimal integral solution. Let \( x_{ij} \in [0, 1] \) denote the fraction of the load capacity \( L \) of server *j* allocated to big movie *i*. We denote by \( y_{ij} \) the fraction of \( D_i \) allocated to small movie *i* on server *j*. Also, \( c_{i,j} \) is the given replication cost (which depends on the initial configuration).

Consider the following linear programming relaxation, \( LP_1 \), for the reconfiguration problem.

\[
(LP_1): \quad \text{minimize} \quad \sum_{i \in \text{Big}} \sum_{j=1}^{N} x_{i,j} \cdot c_{i,j} + \sum_{i \in \text{Small}} \sum_{j=1}^{N} y_{i,j} \cdot c_{i,j} \\
\text{subject to:} \quad \sum_{i \in \text{Big}} x_{i,j} \cdot L + \sum_{i \in \text{Small}} y_{i,j} \cdot D_i \leq L \quad \text{for } 1 \leq j \leq N \quad (3.3) \\
\sum_{i \in \text{Big}} x_{i,j} + \sum_{i \in \text{Small}} y_{i,j} \leq C_j \quad \text{for } 1 \leq j \leq N \quad (3.4) \\
\sum_{j=1}^{N} x_{i,j} = \frac{D_i}{L} \quad \text{for } i \in \text{Big} \quad (3.5) \\
\sum_{j=1}^{N} y_{i,j} = 1 \quad \text{for } i \in \text{Small} \quad (3.6) \\
0 \leq x_{i,j} \leq 1 \quad \text{for } 1 \leq j \leq N, \ i \in \text{Big} \\
0 \leq y_{i,j} \leq 1 \quad \text{for } 1 \leq j \leq N, \ i \in \text{Small}
\]

Constraints (3.3) ensure that the total load capacity used by copies of the big movies and by the small movies on each server is at most \( L \). Constraints (3.4) ensure that the total storage required on server *j* is at most \( C_j \). The constraints (3.5) and (3.6),
together with constraints (3.3), guarantee that each (big or small) movie is allocated \( D_i \) broadcasts.

Next, we modify \( LP_1 \) as follows. For any big movie \( i \), let \( k_i = \lfloor \frac{D_i}{L} \rfloor \). Consider the linear program \( LP_2 \). Note that constraints (3.7) allow to assign to big movie \( i \) some fraction of \( \frac{D_i}{k_i} \) on server \( j \); also, constraints (3.9) guarantee that big movie \( i \) is allocated \( D_i \) broadcasts.

\[
(LP_2) : \begin{align*}
\text{minimize} & \sum_{i \in \text{Big}} \sum_{j=1}^{N} x_{i,j} \cdot c_{i,j} + \sum_{i \in \text{Small}} \sum_{j=1}^{N} y_{i,j} \cdot c_{i,j} \\
\text{subject to:} & \sum_{i \in \text{Big}} x_{i,j} \cdot \frac{D_i}{k_i} + \sum_{i \in \text{Small}} y_{i,j} \cdot D_i \leq L \quad \text{for } 1 \leq j \leq N \quad (3.7) \\
& \sum_{i \in \text{Big}} x_{i,j} + \sum_{i \in \text{Small}} y_{i,j} \leq C_j \quad \text{for } 1 \leq j \leq N \quad (3.8) \\
& \sum_{j=1}^{N} x_{i,j} = k_i \quad \text{for } i \in \text{Big} \quad (3.9) \\
& \sum_{j=1}^{N} y_{i,j} = 1 \quad \text{for } i \in \text{Small} \quad (3.10) \\
& 0 \leq x_{i,j} \leq 1 \quad \text{for } 1 \leq j \leq N, \ i \in \text{Big} \\
& 0 \leq y_{i,j} \leq 1 \quad \text{for } 1 \leq j \leq N, \ i \in \text{Small}
\end{align*}
\]

Clearly, an optimal (fractional) solution for \( LP_2 \) is a lower bound for the optimal solution for the reconfiguration problem, since any feasible solution for \( LP_1 \) is also a feasible solution for \( LP_2 \).

Now, we partition each big movie \( i \) to \( k_i \) sub-movies. Thus, we replace the variables \( x_{i,j}, \ 1 \leq j \leq N, \ i \in \text{Big} \) by the set of variables \( x_{i,j,r}, 1 \leq r \leq k_i \). Intuitively, we partition the load requirement of movie \( i \) to \( k_i \), so we can now consider \( k_i \) sub-movies, where each needs to be allocated \( \hat{D}_i = \frac{D_i}{k_i} \) broadcasts. We note that \( \hat{D}_i \leq \max_{i \in \text{Big}} \{ \frac{D_i}{k_i} \} \).

We rewrite the linear programming relaxation as \( LP_3 \).

**Rounding the Fractional Solution.** Note that \( LP_3 \) can be viewed as the linear programming relaxation of an input for job scheduling on unrelated machines with cardinality constraints, in which we need to schedule a set of jobs on \( N \) unrelated machines. The set of jobs \( J \) corresponds to all the small movies and the collection of sub-movies for the big movies, i.e., \( |J| = |\text{Small}| + \sum_{i \in \text{Big}} k_i \). The processing time of a job corresponding to a small movie \( i \) on machine \( j \) is \( p_{ij} = D_i \), and the processing time of any of the jobs corresponding to the \( k_i \) sub-movies of big movie \( i \) on machine \( j \) is \( p_{ij} = \left\lfloor \frac{D_i}{k_i} \right\rfloor \). Note that if \( k_i = 1 \) then \( p_{ij} = D_i < 2L \). If \( k_i > 1 \) then \( \frac{D_i}{k_i} < 1.5L \) implying (for all \( L > 1 \)) \( \left\lfloor \frac{D_i}{k_i} \right\rfloor < 2L \). In the reduction to the scheduling problem, the cost of processing job \( i \) on machine \( j \) is \( c_{ij}, \forall \ i \) and \( 1 \leq j \leq N \). The makespan of any machine

---

38
j is at most $L$, and the maximal number of jobs that can be assigned to machine $j$ is $C_j$. The goal is to schedule all jobs on the machines, subject to the makespan and cardinality constraints, so as to minimize the total cost.

Given an optimal solution for $LP_3$, we can apply the rounding technique of Shmoys and Tardos [ST93], as described in the proof of Claim 3.7. The resulting integral solution can be used to determine the storage allocation. The assignment matrix is given by the variables $x_{i,j,r}, y_{i,j}$ and the broadcast matrix is given by the allocated processing time. Formally, for a small movie $i$, assign a single copy of $i$ on server $j$ if $y_{i,j} = 1$, i.e., $A_{i,j} = 1$ and $B_{i,j} = D_i$. For any big movie $i$, note that each sub-movie is allocated $\lceil D_i/k_i \rceil > L$ broadcasts, therefore (given that the makespan of the rounded solution is at most $2L$) for all $i,j$, $\sum_{r=1}^{k_i} x_{i,j,r} = 1 \in \{0,1\}$. If $\sum_{r=1}^{k_i} x_{i,j,r} = 1$ then assign a copy of big movie $i$ on server $j$, i.e., $A_{i,j} = 1$ and $B_{i,j} = \lceil D_i/k_i \rceil$.

$$
(LP_3): \text{minimize} \quad \sum_{i \in \text{Big}} \sum_{r=1}^{k_i} \sum_{j=1}^{N} x_{i,j,r} \cdot c_{i,j} + \sum_{i \in \text{Small}} \sum_{j=1}^{N} y_{i,j} \cdot c_{i,j}
$$

subject to:

$$
\sum_{i \in \text{Big}} \sum_{r=1}^{k_i} x_{i,j,r} \cdot \left\lceil \frac{D_i}{k_i} \right\rceil + \sum_{i \in \text{Small}} y_{i,j} D_i \leq L \quad 1 \leq j \leq N
$$

$$
\sum_{i \in \text{Big}} \sum_{r=1}^{k_i} x_{i,j,r} + \sum_{i \in \text{Small}} y_{i,j} \leq C_j \quad 1 \leq j \leq N
$$

$$
\sum_{j=1}^{N} x_{i,j,r} = 1 \quad \text{for} \quad i \in \text{Big}, \quad 1 \leq r \leq k_i
$$

$$
\sum_{j=1}^{N} y_{i,j} = 1 \quad \text{for} \quad i \in \text{Small}
$$

$$
0 \leq x_{i,j,r} \leq 1 \quad \text{for} \quad 1 \leq j \leq N, \quad i \in \text{Big}, \quad 1 \leq r \leq k_i
$$

$$
0 \leq y_{i,j} \leq 1 \quad \text{for} \quad 1 \leq j \leq N, \quad i \in \text{Small}
$$

3.3.2 Analysis

Let $OPT(I)$ denote the cost of an optimal solution for an instance $I$ of the reconfiguration problem.

**Lemma 3.8.** The optimal solution for $LP_3$ is a lower bound for $OPT(I)$.

**Proof.** Given an instance $I$, clearly, the optimal solution for $LP_1$ is a lower bound for $OPT(I)$. Also, any feasible solution for $LP_1$ is a feasible solution for $LP_2$; therefore, an optimal solution for $LP_2$ is a lower bound for the optimal solution of $LP_1$. Now, consider a feasible solution for $LP_2$. By setting in $LP_3 \ x_{i,j,r} = \frac{x_{i,j}}{k_i}$, for all $i \in \text{Big}, 1 \leq r \leq k_i$,

---

1This is the cardinality constraint.
and \(1 \leq j \leq N\), we get a feasible solution for \(LP_3\). It follows that the optimal solution for \(LP_3\) is a lower bound for the optimal solution of \(LP_2\). This completes the proof. \(\Box\)

**Theorem 3.4.** The above algorithm outputs in polynomial time a solution of cost at most \(OPT(I)\). The movies can be stored on \(N\) servers with storage capacities \(C_1, \ldots, C_N\) and load capacities \((2 + \varepsilon)L\), where \(\varepsilon\) is defined in (3.2).

**Proof.** By Lemma 3.8, the cost of an optimal solution for \(LP_3\) is at most \(OPT(I)\). As shown in the proof of Claim 3.7, the rounding procedure preserves the cost of the linear program. We now bound the extra amount of load capacity required for the placement of the files on the servers in the solution output by the algorithm. The maximal broadcast demand of any sub-movie of big movie \(i\) is at most \(L(1 + \varepsilon)\), where \(\varepsilon\) is defined in (3.2), and this is also the maximum processing time of any job in the input for the scheduling problem. Hence, by using the rounding technique of Shmoys and Tardos [ST93], the resulting assignment of files to server may require an increase of at most \(L(1 + \varepsilon)\) in the load capacity of any server. \(\Box\)
Chapter 4

The Reoptimization Model

4.1 The Model

In the following we formally define our model for combinatorial reoptimization. Let Π be an optimization problem, let $I_0$ be an input for Π, and let $C_{I_0} = \{C_{I_0}^1, C_{I_0}^2, \ldots\}$ be the set of configurations corresponding to the solution space of Π for the input $I_0$. Each configuration $C_{I_0}^j \in C_{I_0}$ has some value $val(C_{I_0}^j)$. In the reoptimization problem, $R(\Pi)$, we are given a configuration $C_{I_0}^j \in C_{I_0}$ 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 $C_{I_0}^j \in C_{I_0}$ 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 $C_{I_0}^j \in C_{I_0}$, we are given the transition cost of $i$ when moving from the initial configuration $C_{I_0}^j$ to the feasible configuration $C_I^j$ of the new instance. We denote this transition cost by $\delta(i, C_{I_0}^j, C_I^j)$. 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 $val(C_I^j)$ value, we are looking for a configuration $C_I^j$ for which the total transition cost, given by $\sum_{i \in I} \delta(i, C_{I_0}^j, C_I^j)$ is minimized.

Example 4.1. 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 $e \in E_T \setminus E_{T_0}$, we are given the cost $\delta_{\text{add}}(e)$ of including $e$ in the new solution, and for every edge $e \in E \cap (E_{T_0} \setminus E_T)$ we are given the cost $\delta_{\text{rem}}(e)$ of removing $e$ from the solution. The goal in the reoptimization problem $R(\text{MST})$ is to find an MST of $G$ with minimal total transition cost. As we show in Subsection 5.3.1, $R(\text{MST})$ belongs to a class of subset-selection problems that are polynomially solvable.

The input for the reoptimization problem, $I_R$, contains both the new instance $I$ and the transition costs $\delta$ (that may be encoded in various ways). Note that $I_R$ does not
include the initial configuration $I_0$ - since apart from determining the transition costs, it has no effect on the reoptimization problem.

4.2 Approximate Reoptimization

When the problem $\Pi$ is NP-hard, or when the reoptimization problem $R(\Pi)$ is NP-hard,\(^1\) 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 $O(I)$ the optimal solution value of $\Pi(I)$. A configuration $C^k_I \in C_T$ yields a $\rho$-approximation for $\Pi(I)$, for some $\rho \geq 1$, if its value is within ratio $\rho$ from $O(I)$. That is, if $\Pi$ is a minimization problem then $val(C^k_I) \leq \rho \cdot O(I)$; if $\Pi$ is a maximization problem then $val(C^k_I) \geq \frac{1}{\rho} \cdot O(I)$. Given a reoptimization instance $I_R$, for any $\rho \geq 1$, denote by $O_R(I_R, \rho)$ the minimal possible transition cost to a configuration $C^k_I \in C_T$ that yields a $\rho$-approximation for $O(I)$, and by $O_R(I_R)$ 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 4.1.** An algorithm $A$ yields a strong $(r, \rho)$-reapproximation for $R(\Pi)$, for $\rho, r \geq 1$, if, for any reoptimization input $I_R$, $A$ achieves a $\rho$-approximation for $O(I)$, with transition cost at most $r \cdot O_R(I_R, \rho)$.

Unfortunately, for many NP-hard optimization problems, finding a strong $(r, \rho)$-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 $\rho$-approximation for the optimal one (in which case, the transition cost to a $\rho$-approximate solution is equal to zero). We demonstrate this hardness for the Knapsack problem.

**Theorem 4.1.** For any $r, \rho \geq 1$, obtaining a strong $(r, \rho)$-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 $(r, \rho)$-reapproximation algorithm for Knapsack, for some $\rho \geq 1$, and define the following instance for the reoptimization problem.

1. The set of items consists of $A$ and an additional item $\hat{i}$ having size $B$ and value $\frac{P}{\rho}$. All items have transition cost 1 – charged if added to or removed from the knapsack.

\(^1\)As we show below, it may be that none, both, or only $R(\Pi)$ are NP-hard.
2. In the initial configuration, item \( \hat{i} \) is in the knapsack.

**Claim 4.1.** The \((r, \rho)\)-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 \( \rho \)-approximation. Thus, there exists a \( \rho \)-approximate solution for the reoptimization problem, whose transition cost is 0. For any \( r \), the \((r, \rho)\)-reapproximation algorithm must incur transition cost 0. The only possible packing of profit at least \( \rho P \) 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 \( \rho \)-approximate solution packs item \( \hat{i} \) as it leaves no space for additional items. In particular, no \((r, \rho)\)-reapproximation algorithm packs the item \( \hat{i} \). \( \square \)

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.

![Figure 4.1: The feasible solutions of an input \( I \). Each feasible solution (a dot) has objective value and transition cost. The highlighted area contains the \((r, \rho)\)-approximate solutions.](image)

Formally,
Definition 4.2. An algorithm $A$ yields an $(r, \rho)$-reapproximation for $R(\Pi)$, for $\rho, r \geq 1$, if, for any reoptimization input $I_R$, $A$ achieves a $\rho$-approximation for $O(I)$, with transition cost at most $r \cdot O_R(I_R)$.

Figure 4.1 depicts the sets of $(r, \rho)$-approximate solutions for the reoptimization versions of a maximization problem (upper chart) and minimization problem (lower chart).

Note that in both definitions, the quality of the solution is within factor $\rho$ 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 $\rho$-approximate solution. Clearly, any strong $(r, \rho)$-reapproximation yields also an $(r, \rho)$-reapproximation with respect to Definition 4.2.

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 4.1 and 4.2 coincide.

Example 4.2. 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 $\Pi$ be the minimum makespan scheduling problem [Gra69] (denoted in standard scheduling notation $P||C_{max}$). An instance of $\Pi$ 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 $C^f_{I_0} = (C^f_{I_0}(1), \ldots, C^f_{I_0}(m))$ be a solution of $\Pi$ for $I_0$. That is, $C^f_{I_0}(i)$ specifies the set of jobs assigned to machine $i$, $1 \leq i \leq m$. $\Pi$ is a minimization problem, and $val(C^f_{I_0})$ is the makespan achieved under the assignment $C^f_{I_0}$. Assume further that $C^f_{I_0}$ 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 $C^f_I$, the transition cost of a job $j \in I$ from $C^f_{I_0}$ to $C^f_I$ is equal to 0 if $j$ remains on the same machine, and to 1 if $j$ is assigned to different machines in $C^f_{I_0}$ and $C^f_I$.

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 $S$ 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 $C^f_{I_0}$ to any feasible configuration in $C_T$ (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 [Gra69] for an instance of $m$ machines, we have that this algorithm yields a strong $(1, 2 - \frac{1}{m'})$-reapproximation for $R(\Pi)$.

Our study encompasses a non-trivial class of optimization problems that admit fully polynomial time approximation schemes (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 4.3.** 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 \(|I_R|, \frac{1}{\varepsilon_1} \) and \( \frac{1}{\varepsilon_2} \).

**Example 4.3.** Assume that \( \Pi \) is the single source shortest path problem in a graph. Let \( I \) be an input for the problem, and let \( C^I_j \) be a particular (not necessarily optimal) solution of \( \Pi \) for \( I \). That is, \( C^I_j \) is a set of edges such that the source \( s \) is connected to all vertices in the subgraph induced by \( C^I_j \). The value of the solution is the total weight of edges in \( C^I_j \). Let \( I' \) be an input derived from \( I \) by changing the weight of some edges in the graph. Since the graph topology itself does not change, \( C^I_T = C^I_T' \). Assume that the transition cost for \( \Pi \) is the number of edges removed and added to the solution. Formally, \( \text{cost}(C^I_j, C^I_k) = C^I_j \otimes C^I_k \). The objective of the reoptimization problem is to find \( C^I_k \in C^I_T \) such that \( f(\text{val}(C^I_k), \text{cost}(C^I_j, C^I_k)) \) is minimal, where \( f \) is decreasing in both parameters.

### 4.3 Budgeted Reoptimization

The budgeted reoptimization problem \( R(\Pi, b) \) is a single objective version of \( R(\Pi) \), in which we add the constraint that the transition cost is at most \( b \), for some budget \( b \geq 0 \). Formally, given a universe \( U \), a profit function \( p: U \to \mathbb{N} \), and a transition function \( \delta: U \to \mathbb{R}^+ \), we want to solve

\[
R(\Pi, b) : \max_{s \in U} p(s) \quad (4.1)
\]

subject to: \( \delta(s) \leq b \).

Thus, rather than optimizing on the transition cost, we seek the best possible solution for \( \Pi \), among those of total transition cost at most \( b \). The optimal profit for \( R(\Pi, b) \) is denoted \( p(O(I_R, b)) \), where \( O(I_R, b) \) is the best solution that can be reached from the initial solution with transition cost at most \( b \).

**Definition 4.4.** An algorithm \( \mathcal{A} \) yields an \( r \)-approximation for \( R(\Pi, b) \), for \( r \in (0, 1] \), if for any reoptimization input \( I \), \( \mathcal{A} \) yields a solution \( s \) of profit \( p(s) \geq r \cdot p(O(I_R, b)) \), and transition cost at most \( b \).

**Example 4.4.** Assume that \( \Pi \) is the 0-1 Knapsack problem. An instance \( I \) of \( \Pi \) consists of a bin of capacity \( B \) and \( n \) items with profits \( p_i \geq 0 \) and weights \( w_i \geq 0 \), for \( 1 \leq i \leq n \). The formal representation of the problem is \( U = \{ s | s \text{ is a feasible packing of the bin} \} \), and for any \( s \in U \), \( p(s) = \sum_{i \in s} p_i \). In the reoptimization version of the problem, \( R(\Pi) \), each instance \( I \) contains also the transition cost of item \( i \), given by \( \delta_i \geq 0 \), for \( 1 \leq i \leq n \).
In the budgeted reoptimization version, $R(\Pi, b)$, $U$ is restricted to contain solutions having transition cost at most $b$. Thus, $U = \{s \mid s$ is a feasible packing of the bin, and $\delta(s) \leq b\} = \{s \mid w(s) \leq B$, and $\delta(s) \leq b\}$.

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

In Section 5.1, we show how we can use efficient algorithms for $R(\Pi, b)$, for $b \in \mathbb{N}$, to obtain a reapproximation algorithm for $R(\Pi)$.

In Section 5.2, we use the fact that for various problems, $R(\Pi, b)$ satisfies the conditions of Corollary 2.3. In particular, given a problem $\Pi$, let $\Gamma_b = R(\Pi, b)$, for some $b \geq 0$, and let $\Gamma_b(\lambda)$ be the Lagrangian relaxation of $R(\Pi, b)$, i.e., $\Gamma_b(\lambda) = \max_{s \in U} p(s) - \lambda \cdot \delta(s)$. If $\Gamma_b(\lambda)$ yields an instance of $\Pi$ then, by Corollary 2.3, an $r$-approximation algorithm $A$ for $\Pi$, satisfying for certain value of $\lambda$: $w(A) \leq b$, yields an $\left(\frac{r}{r+1} - \varepsilon\right)$-approximation for $R(\Pi, b)$. 

46
Chapter 5

Frameworks for Obtaining Reapproximation Algorithms

In this chapter we describe in detail the frameworks that we developed for obtaining reoptimization and reapproximation algorithms. In Section 5.1 we introduce our framework for obtaining an FPTRS for any reoptimization problem in DP-B. Section 5.2 presents a framework for obtaining reapproximation algorithms by using a budgeted version of the reoptimization problem. Section 5.3 includes two frameworks that yield reoptimization and reapproximation algorithms for subset selection problems.

5.1 Reoptimization of DP-benevolent Problems

5.1.1 Polynomially Bounded Transition Costs

We first consider instances in which the transition costs are polynomially bounded in the input size. In Subsection 5.1.2 we show how our results can be extended to instances with arbitrary costs. The first main result is the following.

**Theorem 5.1.** Let $R(\Pi)$ be the reoptimization version of a problem $\Pi \in DP-B$, 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.

As defined in Section 4.3, the problem $R(\Pi, m)$ is a restricted version of $R(\Pi)$, in which the total transition cost is at most $m$, for some integer $m \geq 0$. Recall that $O(I_R, m)$ is the optimal value of $R(\Pi, m)$ for the input $I_R$. Let $DP'$ denote the dynamic program for $R(\Pi, m)$. 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 2.2 – 2.5. In the second step, we show that $R(\Pi, m)$ satisfies Conditions 2.1 – 2.4.
Lemma 5.1. For any \( \Pi \in DP-B \) and \( m \in \mathbb{N} \), the problem \( R(\Pi, m) \) is DP-simple.

Proof. We reformulate Definitions 2.2–2.5 and show that they hold for \( R(\Pi, m) \).

Following the notation of [Woe01] (see Section 2.2), we denote by \( \alpha \) the number of different parameters associated with any element in the input \( I \); \( \beta \) 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 \( \Pi \). Also, the cost of the transition \( F \) for element \( i \) is given by \( r_{F,i} \). Let \( \bar{R}_i = (r_{F_1,i}, r_{F_2,i}, \ldots) \) 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 5.1. Structure of the input for \( R(\Pi, m) \) (reformulation of Definition 2.2).

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 \( \alpha \) 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 2.2 holds.

Definition 5.2. Structure of the dynamic program \( DP' \) (reformulation of Definition 2.3).

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 \( S'_k \) of states. A state is a vector \( \bar{\Upsilon} = (\bar{S}; c) \), where \( \bar{S} \in S'_k \subseteq \mathbb{N}^\beta \), and \( 0 \leq c \leq m \). Let \( \beta' \) denote the length of a state vector for \( R(\Pi, m) \), then Definition 2.3 holds, with \( \beta' = \beta + 1 \).

Definition 5.3. Iterative computation of the state space in \( DP' \) (reformulation of Definition 2.4). 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 $S_0$ 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 $S_0' = \{ \bar{S} \mid \bar{S}' = (\bar{S}; 0), \bar{S} \in S_0 \}$. We note that $|S_0'| = |S_0|$. The state space for phase $k$ of $DP'$ is given by

$$S_k' = \{ F(\bar{Y}_k, \bar{T}) \mid F \in F', \bar{T} \in S_{k-1}', \text{ and } H_F(\bar{Y}_k, \bar{T}) \leq 0 \}.$$ 

Therefore, Definition 2.4 holds.

**Definition 5.4.** Objective value in $DP'$ (reformulation of Definition 2.5.)

For any $\bar{T} = \bar{S}; c \in S'$, the objective function $G' : \mathbb{N}^\beta' \to \mathbb{N}$ is defined by $G'(\bar{T}) = G(\bar{S})$. Therefore, Definition 2.5 holds.

This completes the proof of the lemma.

**Theorem 5.2.** For any $\Pi \in DP$ and $m \in \mathbb{N}$, $R(\Pi, m) \in DP$.

**Proof.** 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 $\bar{T} = (\bar{S}; c)$ and $\bar{T}' = (\bar{S}', c')$. We now define a partial order on the state space. We say that $\bar{T} \preceq_{\text{qua}} \bar{T}'$ iff $\bar{S} \preceq_{\text{dom}} \bar{S}'$ and $c' \leq c$. Also, $\bar{T} \preceq_{\text{qua}} \bar{T}'$ iff $\bar{S} \preceq_{\text{qua}} \bar{S}'$. We note that the following holds.

**Observation 5.2.** If $\bar{T}$ is $(\bar{D}', \Delta)$-close to $\bar{T}'$, then $\bar{S}$ is $(\bar{D}, \Delta)$-close to $\bar{S}'$, and $c = c'$.

We now show that the four conditions, 2.1 – 2.4, are satisfied for $R(\Pi, m)$.

C.1 For any $\Delta > 1$ and $F' \in F'$, for any input $\bar{Y} = (\bar{X}; \bar{R}) \in \mathbb{N}^\alpha'$, and for any pair of states $\bar{T} = (\bar{S}; c), \bar{T}' = (\bar{S}', c') \in \mathbb{N}^\beta'$:

(i) If $\bar{T}$ is $(\bar{D}', \Delta)$-close to $\bar{T}'$ and $\bar{T} \preceq_{\text{qua}} \bar{T}'$ then, by the definition of $(\bar{D}', \Delta)$-closeness, $\bar{S}$ is $(\bar{D}, \Delta)$-close to $\bar{S}'$. In addition, $\bar{T} \preceq_{\text{qua}} \bar{T}'$ implies that $\bar{S} \preceq_{\text{qua}} \bar{S}'$. 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 $F'(\bar{Y}, \bar{T}) \preceq_{\text{qua}} F'(\bar{Y}, \bar{T}')$. We have that $F(\bar{X}, \bar{S}) \preceq_{\text{qua}} F(\bar{X}, \bar{S}')$, therefore, $F(\bar{X}, \bar{S}); (c + r_F) \preceq_{\text{qua}} F(\bar{X}, \bar{S}); (c + r_F)$. Indeed, the $\preceq_{\text{qua}}$ relation is not affected by the cost component of two states. Moreover, by Observation 5.2, $c = c'$. By the definition of $F'$, $F'(\bar{X}; \bar{R}; \bar{S}; c) \preceq_{\text{qua}} F'(\bar{X}; \bar{R}; \bar{S}'; c'), i.e., F'(\bar{Y}, \bar{T}) \preceq_{\text{qua}} F'(\bar{Y}, \bar{T}')$.

- We now show that $F'(\bar{Y}, \bar{T})$ is $(\bar{D}', \Delta)$-close to $F'(\bar{Y}, \bar{T}')$. We have that $F(\bar{X}, \bar{S})$ is $(\bar{D}, \Delta)$-close to $F(\bar{X}, \bar{S}')$. This implies

**Claim 5.3.** $F(\bar{X}, \bar{S}); (c + r_F)$ 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 $\beta$ 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 5.2). 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 $F'(\bar{Y}, \bar{Y'})$ is $(\bar{D'}, \Delta)$-close to $F'(\bar{Y}, \bar{Y'})$.

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

$$F(\bar{X}, \bar{S}; (c + r_F)) \leq_{dom} F'(\bar{X}, \bar{S}'; (c + r_{F'}).$$

(5.1)

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

$$F'(\bar{X}; \bar{R}, \bar{S}; c) \leq_{dom} F'(\bar{X}; \bar{R}, \bar{S}'; c'),$$

or

$$F'(\bar{Y}, \bar{Y'}) \leq_{dom} F'(\bar{Y}, \bar{Y'}).$$

(5.2)

(ii) If $\bar{Y} \leq_{dom} \bar{Y}'$ then, by the definition of $\leq'_{dom}$, we have that $\bar{S} \leq_{dom} \bar{S}'$. Hence, we get (5.1), (5.2) and (5.3).

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

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

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

(b) If $c + r_F \leq m$ then since $\bar{S}$ is $(\bar{D}, \Delta)$-close to $\bar{S}'$, $c = c'$ (by Observation 5.2) and $\bar{S} \leq'_{qua} \bar{S}'$ (follows immediately from the fact that $\bar{Y} \leq'_{qua} \bar{Y}'$), we have from Condition A.2(i) that

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

(5.4)

Hence, we have

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

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

The second equality holds since $c + r_F \leq m$. The inequality follows from (5.4).

(ii) We need to show that if $\bar{Y} \leq_{dom} \bar{Y}'$ then $H'_F(\bar{Y}, \bar{Y}') \leq H'_F(\bar{Y}, \bar{Y})$. Note that if $\bar{Y} \leq'_{dom} \bar{Y}'$ then, by the definition of the order $\leq'_{dom}$, it holds that $c' \leq c$ and $\bar{S} \leq_{dom} \bar{S}'$. 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, \( H'_F(\bar{Y}, \bar{T}') \leq 1 = H'_F(\bar{Y}, \bar{T}) \).

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

\[
H'_F(\bar{Y}, \bar{T}') = H'_F(\bar{X}; \bar{R}, \bar{S}'; c') = H_F(\bar{X}, \bar{S}')
\]

\[
\leq H_F(\bar{X}, \bar{S}) = H'_F(\bar{X}; \bar{R}, \bar{S}; c) = H'_F(\bar{Y}, \bar{T}).
\]

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 \( \bar{Y}, \bar{T}' \) satisfying: \( \bar{Y} \) is \((D, \Delta)\)-close to \( \bar{T}' \), it holds that

\[ G(\bar{Y}) \leq G(\bar{T}') \cdot \Delta^g. \tag{5.5} \]

Assume that \( \Pi \) is a maximization problems. Intuitively, if we choose \( \bar{T}' \) instead of \( \bar{Y} \), since \( \bar{Y} \preceq_{qua} \bar{T}' \), then we are still close to the maximum profit, due to (5.5).

Since \( \bar{Y} \) is \((D, \Delta)\)-close to \( \bar{T}' \), by Observation \(5.2\), 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

\[ G'(\bar{Y}) \leq G'(\bar{S}; c) = G(\bar{S}) \leq \Delta^g G(\bar{S}') = \Delta^g G'(\bar{T}'). \]

(ii) We need to show that if \( \bar{Y} \preceq_{dom} \bar{T}' \) then \( G'(\bar{T}') \geq G'(\bar{Y}) \). Note that if \( \bar{Y} \preceq_{dom} \bar{T}' \) 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,

\[ G'(\bar{T}') = G'(\bar{S'}; c') = G(\bar{S'}) \geq G(\bar{S}) = G'(\bar{S}; c) = G'(\bar{Y}). \]

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 if $S_0$ can be computed in time that is polynomial in $n$ and $\log \bar{x}$, the same holds for $S'_0$.

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

This completes the proof of the Theorem. \hfill \qed

Let $O_R(I_R) \equiv O_R(I_R, 1)$ be the minimal transition cost required to get an optimal solution for $I$. The next lemmas will be used in the proof of Theorem 5.1.

**Lemma 5.4.** For any $\mu \geq 0$, if

\[ O_R(I_R) \leq \mu \]

\[(5.6) \]

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

**Proof.** If (5.6) holds then $\mu$ 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 $\mu$. This solution is feasible also for the input $I_R$ of $R(\Pi, m)$, for any $m \geq \mu$. \hfill \qed

Given a problem $\Pi \in \text{DP-B}$, 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 $\text{DP}'$ with $\varepsilon$ as a parameter.

**Lemma 5.5.** For any integer $m \geq 0$, if $O_R(I_R) \leq m$ then

\[ T(R(\Pi, m), I_R, \varepsilon) \geq (1 - \varepsilon)T(\Pi, I, \varepsilon) \]

**Proof.** Since $R(\Pi, m) \in \text{DP-B}$, by Theorem 2.1, for any $m \in \mathbb{N}$,

\[ T(R(\Pi, m), I_R, \varepsilon) \geq (1 - \varepsilon)O(I_R, m) = (1 - \varepsilon)O(I) \geq (1 - \varepsilon)T(\Pi, I, \varepsilon). \]

The equality follows from Lemma 5.4, and the second inequality holds since $\Pi$ is a maximization problem. \hfill \qed

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 $\mu$ satisfying (5.6). Formally,
Definition 5.5. For any maximization problem Π and ε ≥ 0,

\[ \ell(R(\Pi), I_R, \varepsilon) \equiv \min\{ m \in \mathbb{N} \mid T(R(\Pi, m), I_R, \varepsilon) \geq (1 - \varepsilon)T(\Pi, I, \varepsilon) \} \quad (5.7) \]

For short, let \( \ell^* = \ell(R(\Pi), I_R, \varepsilon) \) be the minimal value found in (5.7). The next two lemmas show that \( T(R(\Pi, \ell^*), I_R, \varepsilon) \) satisfies two properties: (i) The transition cost is at most \( \mathcal{O}(I_R) \), and (ii) the resulting solution yields a \((1 + 2\varepsilon)\)-approximation for the maximization problem Π.

Lemma 5.6. For any \( \varepsilon \geq 0 \), \( \ell^* \leq \mathcal{O}(I_R) \).

Proof. By Lemma 5.5, the inequality in (5.7) holds for any \( m \geq \mathcal{O}(I_R) \). Since we choose the minimum value of \( m \) satisfying (5.7), this gives the claim. \( \square \)

Lemma 5.7. For any \( \varepsilon \geq 0 \), \( T(R(\Pi, \ell^*), I_R, \varepsilon) \geq (1 - 2\varepsilon)\mathcal{O}(I) \).

Proof. By the definition of \( \ell^* \), 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 \( \Pi \in DP-B \). \( \square \)

Proof of Theorem 5.1. 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 Lemma 5.6 and Lemma 5.7.

Next, we show that the resulting reapproximation scheme is polynomial in \( n \) and in \( \frac{1}{\varepsilon} \). This holds since, by Theorem 5.2, \( R(\Pi, m) \in DP-B \) for any \( m \in \mathbb{N} \). Also, \( \ell^* \) can be found in polynomial time in the input size. \( \square \)

We now summarize the steps of the algorithm for finding a \((1, 1 + \varepsilon)\)-reapproximation for \( R(\Pi) \), where \( \Pi \in DP-B \).

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Given an input ( I_R ) for ( R(\Pi) ) and ( \varepsilon &gt; 0 ), let ( \varepsilon = \varepsilon/2 ).</td>
</tr>
<tr>
<td>2.</td>
<td>Find ( \ell^* = \ell(R(\Pi), I_R, \varepsilon) ).</td>
</tr>
<tr>
<td>3.</td>
<td>Return a state in ( DP' ) corresponding to a solution for ( R(\Pi) ) whose value is ( T(R(\Pi, \ell^*), I_R, \varepsilon) ).</td>
</tr>
</tbody>
</table>

5.1.2 Arbitrary Transition Costs

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

**Definition 5.6.** Given an input $I_R$ for $R(\Pi)$, let $\gamma$ be a function that accepts as input the cost vector $\hat{R}_i$, $1 \leq i \leq n$, and the parameters $d,n \in \mathbb{N}$ and $\varepsilon \in (0, 1)$, then $\gamma(\hat{R}_i, d, n, \varepsilon) = (\gamma(r_{F_1,i}), \gamma(r_{F_2,i}), \ldots)$, where

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

In applying $\gamma$ on $I_R$, each element $\hat{Y}_i = (\hat{X}_i; \hat{R}_i) \in I_R$ is modified to $\hat{Y}_i' = (\hat{X}_i; \gamma(\hat{R}_i, d, n, \varepsilon))$. The resulting instance is denoted $\hat{I}_{R,\varepsilon}$. The following is the pseudocode of our algorithm for finding a $(1 + \varepsilon_1, 1 + \varepsilon_2)$-reapproximation for $R(\Pi)$, where $\Pi \in \text{DP-B}$.

**Algorithm 5.1 $A_{\text{DP-B}}$**

1. Given an input $I_R$ for $R(\Pi)$ and $\varepsilon_1, \varepsilon_2 > 0$, guess the maximum transition cost, $0 \leq C_{\max} \leq \max_{1 \leq i \leq n}\{\max_{1 \leq j \leq n_i} r_{F_j,i}\}$, 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, $\Pi$, within factor $(1 + \varepsilon_2)$, using the rounded instance $\hat{I}_{R,\varepsilon_1}$.
3. Output a $(1 + \varepsilon_1, 1 + \varepsilon_2)$-reapproximation solution for $R(\Pi)$, i.e., a solution that is $(1 + \varepsilon_2)$-approximation for $\Pi$ on the input $I_R$, whose total transition cost is at most $m(1 + \varepsilon_1)$.

**Theorem 5.3.** Let $R(\Pi)$ be the reoptimization version of a problem $\Pi \in \text{DP-B}$, 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 $\gamma(\hat{R}_i, m, n, \varepsilon_1)$ for each cost vector $\hat{R}_i$, $1 \leq i \leq n$, are rounded down, it holds that $O_R(\hat{I}_{R,\varepsilon_1}) \leq O_R(I_R)$. Also, the transition costs in $\hat{I}_{R,\varepsilon_1}$ are polynomial. Thus, by Theorem 5.1, for any fixed $\varepsilon_2 > 0$, there exists an FPTRS that yields a $(1 + \varepsilon_1, 1 + \varepsilon_2)$-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 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 $O_R(I_R)$ at most $(1 + \varepsilon)^{h+1}$. Also, the total contribution of transitions with 'small'
cost (i.e., each such transition of an element 1 ≤ i ≤ n has cost 0 ≤ r_{F_i,i} < \frac{\varepsilon m}{n}) is at most εm, since there are at most n such transitions.

The running time of \( \mathcal{A}_{DP-B} \) 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, \( \hat{R}_i \), gives all the possible transition costs for element \( i \), for 1 ≤ i ≤ 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. \( \Box \)

We note that the result in Theorem 5.3 is the best possible, unless \( P = NP \). Indeed, there exist optimization problems \( \Pi \) 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 \( \Pi \) for \( I \) is equivalent to solving \( R(\Pi) \) for \( IR \).

The next result follows from applying Theorem 5.3 to the problems in DP-B.

**Corollary 5.4.** Each of the problems listed in Subsection 2.2.1 admits an FPTRS.

### 5.1.3 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 \( DP-B \) with \( \alpha = 2 \) and \( \beta = 2 \). For 1 ≤ k ≤ n define the input vector \( \bar{X}_k = [p_k, w_k] \). A state \( \bar{S} = [s_1, s_2] \) in \( S_k \) 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 \( S_0 = \{ [0, 0] \} \).

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 \( K \subseteq \{ x \in \mathbb{N} \mid 1 \leq x \leq n \} \) such that

1. \( \sum_{k \in K} w_k \leq W \)

2. The total transition cost is at most the budget, that is,
   \( \sum_{k \in K} c(F_1, k) + \sum_{k \in K} c(F_2, k) \leq m \)

3. The profit \( \sum_{k \in K} p_k \) is maximized.

\( R(\text{Knapsack}, m) \) is in \( DP\cdot B \) 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 \( S'_k \) 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 \( S'_0 = \{ [0, 0, 0] \} \).

### 5.2 Reapproximation via Budgeted Reoptimization

In the following we show how the Lagrangian relaxation technique described in Chapter 2 can be utilized to obtain reapproximation algorithms. Recall that \( R(\Pi, b) \) is a restricted version of \( R(\Pi) \), in which we add the constraint that the transition cost is at most \( b \), for some budget \( b \geq 0 \).

**Theorem 5.5.** Let \( I \) be an instance of the reoptimization problem \( R(\Pi) \). For \( r_1, r_2 \in (0, 1] \), given an \( r_1 \)-approximation algorithm \( A \) for \( \Pi \), and an \( r_2 \)-approximation algorithm \( A_b \) for \( R(\Pi, b) \), for all \( b \geq 0 \), Algorithm 5.2 yields in polynomial time a \( (1, r_1 \cdot r_2) \)-reapproximation for \( R(\Pi) \).
Algorithm 5.2 Reapproximating $R(\Pi)$ for an instance $I$

1. For $r_1, r_2 \in (0, 1]$, let $A$ be an $r_1$-approximation algorithm for $\Pi$.
2. Let $A_B$ be an $r_2$-approximation algorithm for $R(\Pi, B)$.
3. Approximate $\Pi(I)$ using $A(I)$:

$$Z \leftarrow p(A(I))$$

4. Use binary search to find a budget $b > 0$ satisfying:
   1. $p(A_b(I)) \geq r_2 \cdot Z$
   2. $p(A_{b-1}(I)) < r_2 \cdot Z$
5. Return $A_b(I)$

Recall that $O(I)$ is an optimal solution for $\Pi$, and $OPT$ is a solution for $R(\Pi)$ having the minimum transition cost, among the solutions of profit $p(O(I))$. In Subsection 5.2.1 we prove the theorem, by showing that the solution, $S_A$, output by Algorithm 5.2 has the following properties.

(i) The total transition cost of $S_A$ is at most the transition cost of $OPT$, i.e.,

$$\delta(S_A) \leq \delta(OPT)$$

(ii) The profit of $S_A$ satisfies

$$p(S_A) \geq r_1 \cdot r_2 \cdot p(OPT).$$

Figure 5.1: An illustration of the budgeted functions $OPT_B$, $A_B$, and $r_2 \cdot A_B$ for a given input $I$.

For a given input $I$, Figure 5.1 depicts the budgeted functions $OPT_B(I)$, $A_B$, and
r_2 \cdot A_B\) where \(\Pi\) is a maximization problem. As the budget grows, the best profit grows as well. Thus, all functions are monotone. Since \(A_b\) yields an \(r_2\)-approximation, for any \(b\), \(A_b(I) \leq OPT_b(I) \leq r_2 \cdot A_b(I)\).

An informal proof of Theorem 5.5 can be derived from Figure 5.1. Algorithm 5.2 starts by calculating \(Z = A(I)\), then it continues by searching for a budget \(b\) on the intersection of the function \(r_2 \cdot A_B(I)\) and the profit \(Z\). This intersection divides the plane into two areas (the red and the green). In the red area, the budget is too small to obtain a feasible solution with the desired profit. Thus, \(\delta(OPT) \geq b\).

### 5.2.1 Proof of Theorem 5.5

We use in the proof the next lemmas.

**Lemma 5.8.** The solution output by Algorithm 5.2 for an instance \(I\) satisfies \(\delta(S_A) \leq \delta(OPT)\).

**Proof.** Let \(OPT\) be a solution of minimum transition cost, among those that yield an optimal profit for \(\Pi\), and let \(b^* \geq \delta(OPT)\). By definition, \(OPT\) is a valid solution of \(R(\Pi, b^*)\). Hence, the optimal profit of \(R(\Pi, b^*)\) is \(p(OPT)\), and we have

\[
p(A_{b^*}(I)) \geq r_2 \cdot p(OPT) \geq r_2 \cdot Z.
\]

It follows that, for any budget \(b\) satisfying \(A_b(I) < r_2 \cdot Z\), we have \(b < \delta(OPT)\). By Step (2), the algorithm selects a budget \(b\) such that \(p(A_{b-1}(I)) < r_2 \cdot Z\). Hence, \(b - 1 < \delta(OPT)\). Since \(b\) and \(\delta(OPT)\) are integers, we have that \(b \leq \delta(OPT)\).

**Lemma 5.9.** The profit of \(S_A\) satisfies \(p(S_A) \geq r_1 \cdot r_2 \cdot p(OPT)\).

**Proof.** In Step 1, Algorithm 5.2 selects a solution of profit at least \(r_2 \cdot Z\). Also, \(Z\) is an \(r_1\)-approximation for \(\Pi\); therefore, \(Z \geq r_1 \cdot p(OPT)\). This yields the statement of the lemma.

**Lemma 5.10.** Algorithm 5.2 has polynomial running time.

**Proof.** The algorithm proceeds in three steps. Step 3 is polynomial since \(A\) runs in polynomial time. In Step 4 we search over all budgets \(0 \leq b \leq b_{\text{max}} = \sum_{a \in I} \delta(a)\). While \(b_{\text{max}}\) may be arbitrarily large, \(\log(b_{\text{max}})\) is polynomial in the input size, and indeed Algorithm 5.2 calls \(A_B\ \Theta(\log(b_{\text{max}}))\) times.

Combining the above lemmas, we have the statement of the theorem.

### 5.3 Reoptimization of Subset Selection Problems

#### 5.3.1 Optimal Reoptimization of Weighted Subset Selection Problems

In this section we show that for any subset-selection problem \(\Pi\) over \(n\) elements that can be solved optimally in \(T(n)\) time, there is a 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 \textit{fixed parameter tractable}, then so is $R(Π)$. 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 $Π(I_0)$, the goal in the reoptimization problem is to find an optimal solution $S$ of $Π(I)$ 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 $δ_{rem}(i)$ to be charged if $i \in S_0 \setminus S$, and an addition-cost $δ_{add}(i)$ 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. The following is a reoptimization algorithm for $R(Π)$.

**Algorithm 5.3 $A_{Subset-Select}$: A reoptimization algorithm for $R(Π)$**

1. Let $Δ = \max\{\max_{i \in S_0 \cap I} δ_{rem}(i), \max_{i \in I \setminus S_0} δ_{add}(i)\}$.
2. Let $λ = 2 |I| Δ + 1$.
3. for all $i \in I \cap S_0$ do
   4. $\hat{w}_i = λ w_i' + δ_{rem}(i)$.
5. end for
6. for all $i \in I \setminus S_0$ do
   7. $\hat{w}_i = λ w_i' − δ_{add}(i)$
8. end for
9. Solve $Π(I)$ with weights $\hat{w}$.

**Theorem 5.6.** An optimal solution for $Π(I)$ with weights $\hat{w}$ is an optimal solution for $Π(I)$ with weights $w'$, and a minimal transition cost, i.e., it is a reoptimization for $R(Π)$.

**Proof.** The proof combines two claims, regarding the optimality of the solution for $Π(I)$ and regarding the minimization of the transition costs. Being a subset-selection problem, the solution for Π is given by a binary vector $X = (x_1, ..., x_n)$, 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 in the solution.

**Claim 5.11.** 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,

\[
1 + \sum_{i=1}^{n} w_i' \cdot x_i \leq \sum_{i=1}^{n} w_i' \cdot y_i.
\]  

(5.8)

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

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

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 5.8(multiplied by \( \lambda \)), 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 5.12.** 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 \( X = (x_1, ..., x_n) \) 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 \( Y = (y_1, ..., y_n) \) 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 \( \sum_{i \in S_X \backslash S_0} \delta_{\text{add}}(i) + \sum_{i \in (S_0 \cap I) \backslash S_X} \delta_{\text{rem}}(i) \). Note that by the definition of \( \hat{w} \) the transition cost can be written as \( \sum_{i \in S_0 \cap \Gamma} \delta_{\text{rem}}(i) - D_X \). Clearly, \( D_X \geq D_Y \) implies that \( \sum_{i \in S_0 \cap \Gamma} \delta_{\text{rem}}(i) - D_X \leq
Combining the above claims, we get that the algorithm yields a 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 5.7.** 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). 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 [Moo68].

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 ([LM69]). 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.

### 5.3.2 Reapproximation of Weighted Subset Selection Problems

We now show how the Lagrangian relaxation technique described in Chapter 2 can be used to obtain \((1,\alpha)\)-reapproximation algorithms for subset selection problems, where \( \alpha \in (0, 1) \).

**Corollary 5.8.** Let \( R(\Pi) \) be the reoptimization version of a subset selection problem \( \Pi \), and let \( \Gamma_b = R(\Pi, b) \), for \( b \geq 0 \). Denote by \( A \) an \( r \)-approximation algorithm for \( \Pi \), for \( r \in (0, 1) \). If the lagrangian relaxation of \( \Gamma_b \), \( \Gamma_b(\lambda) \), yields an instance of the base problem \( \Pi \), for all \( b \geq 0 \), then for any \( \varepsilon > 0 \), Algorithm 5.2 is a \((1, \frac{r^2}{1+r^2} - \varepsilon)\)-reapproximation algorithm for \( R(\Pi) \).
Proof. By Corollary 2.3, given $\varepsilon' > 0$, we have an $(\frac{r}{r+1} - \varepsilon')$-approximation algorithm, $A_b$, for $R(\Pi, b)$, for any $b \geq 0$. Thus, using Theorem 5.5 with algorithms $A$ and $A_b$, and taking $\varepsilon' = \frac{\varepsilon}{r}$, we obtain a $(1, r \cdot (\frac{r}{r+1} - \frac{\varepsilon}{r}))$-reapproximation algorithm, i.e., a $(1, \frac{r^2}{r+1} - \varepsilon)$-reapproximation algorithm for $R(\Pi)$. \hfill \Box
Chapter 6

Reapproximation Algorithms for Selected Problems

6.1 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 [HS85] and [DF85]) and the minimum reoptimization cost. The input for the problem is a set $P$ 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 $S \setminus S_0$, 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 call $D_S = \max_{1 \leq j \leq n} d(j,S)$ the radius of $S$. We give below the pseudocode of $A_{R(k-Center)}$, a reapproximation algorithm for $R(k-Center)$. The transition cost incurred by the algorithm is measured relative to the optimum, as given in Definition 4.2.

Theorem 6.1. $A_{R(k-Center)}$ is a $(1,6)$-reapproximation algorithm for $R(k-Center)$.

Proof. Let $c(S_{OPT})$ be the minimum transition cost to an optimal set of centers, $S_{OPT}$, that achieves the radius $D_{OPT}$. Define the cost lower bound (CLB) of a ball $B_j$ to be the cost of the cheapest center that can be opened inside $B_j$. Clearly, since $r \geq D_{OPT}$, the
Algorithm 6.1 $A_{R(k-\text{Center})}$

1: Let $P = \{1, \ldots, n\}$ be the set of clients.
2: Run a 2-approximation algorithm that yields a solution of radius $r$.
3: $\mathcal{B} = \emptyset$.
4: for $1 \leq j \leq n$ do
5: Define a ball $B_j$ of radius $r$, whose center point is client $j$.
6: $\mathcal{B} = \mathcal{B} \cup \{j\}$.
7: end for
8: Let $S = \emptyset$ and $c(S) = 0$.
9: while $|S| < k$ do
10: 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., $c(j) = \min\{c(\ell) : \ell \in P, \ell \in \cup_{r \in \mathcal{B}} B_r\}$.
11: $S = S \cup \{j\}$
12: $c(S) = c(S) + c(j)$
13: Omit the balls intersecting $B_i$, i.e., $\mathcal{B} = \mathcal{B} \setminus \{\{r\} : B_r \cap B_i \neq \emptyset\}$
14: end while
15: Return $S, c(S)$

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.

We first argue that $A_{R(k-\text{Center})}$ incurs the optimal opening cost, i.e., the set $S$ of centers output by the algorithm satisfies $c(S) = c(S_{OPT})$, where $S_{OPT}$ is the set of centers opened by $OPT$. Note that $A_{R(k-\text{Center})}$ defines an independent set of clients (the set of clients whose balls are considered), and $c(S)$, the cost of $A_{R(k-\text{Center})}$, 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 $r$ from a center opened by $A_{R(k-\text{Center})}$. Furthermore, since $r \geq D_{OPT}$, any maximal independent set of clients is of size at most $k$. Since the independent set defined by $A_{R(k-\text{Center})}$ is maximal, after at most $k$ iterations $\mathcal{B}$ is empty. Therefore, any client not in the independent set is at distance at most $2 \cdot r$ from a client in the independent set, and thus at distance at most $3 \cdot r \leq 6 \cdot D_{OPT}$ from a center.

Finally, $A_{R(k-\text{Center})}$ has polynomial running time, since Step 2. can be implemented in polynomial time (see, e.g., [HS85, DF85]). Other steps can also be implemented in $O(n^2)$ time. \qed
6.2 Obtaining Reapproximation Algorithms via Our Frameworks

6.2.1 The Surgery Room Allocation Problem

We now show how we can use Algorithm 5.2 to obtain a \((1, \alpha)\)-reapproximation algorithm for the surgery room allocation problem (SRAP), for some \(\alpha \in (0, 1)\). Recall, that an input for the real-time scheduling problem consists of a set \(A = \{A_1, \ldots, A_m\}\) of activities, where each activity consists of a set of instances; an instance \(I \in A_j\) is defined by a half open time interval \([s(I), e(I))\) in which the instance can be scheduled (\(s(I)\) is the start time, and \(e(I)\) is the end time), and a profit \(p(I) \geq 0\). A schedule is feasible if it contains at most one instance of each activity, and for any \(t \geq 0\), at most one instance is scheduled at time \(t\). The goal is to find a feasible schedule of a subset of the activities that maximizes the total profit (see, e.g., [BBF+01]). Let II be the real-time scheduling problem. Then SRAP can be cast as \(R(\Pi)\), the reoptimization version of \(\Pi\).

Now, given budgeted SRAP, \(\Gamma_b = R(\Pi, b)\), in which the transition cost is bounded by \(b\), for some \(b \geq 0\), we can write \(\Gamma_b\) in the form

\[
\Gamma_b : \max_{S \in X} f(S)
\]

subject to: \(w(S) \leq b\), \hspace{1cm} (6.1)

where \(X = \{\text{all feasible operation schedules}\}\), and \(w(S) = \delta(S)\). The Lagrangian relaxation of \(\Gamma_b\) is \(\Gamma_b(\lambda) : \max_{S \in X} f(S) - \lambda \cdot w(S)\). We note that \(\Gamma_b(\lambda)\) yields an instance of the real-time scheduling problem, II. Our base problem, II, can be approximated within factor \(\frac{1}{3}\) [BBF+01]. By Theorem 2.4, budgeted real-time scheduling admits a \((\frac{1}{3} - \varepsilon)\)-approximation. The next result follows from Theorem 5.5.

**Theorem 6.2.** There is a polynomial-time \((1, \frac{1}{6} - \varepsilon)\)-reapproximation algorithm for SRAP.

6.2.2 The Cloud Provider Problem

Recall that, in the cloud provider problem (CPP), there are \(n\) hypervisors, each having computing power of \(C_i\), and \(m\) virtual machines. Each virtual machine \(1 \leq j \leq m\) requires \(r_j\) computing units and has a profit \(p_j\) and migration cost \(\delta_{i,j}\), for the reassignment of machine \(j\) to server (hypervisor) \(i\). An assignment is a function from a subset of the machines \(X\) to the hypervisor set, \(S : X \subseteq [m] \rightarrow [n]\). A feasible solution for the problem is an assignment \(S\) such that, for each hypervisor \(1 \leq i \leq n\) the total amount of computing units required for the assignment does not exceed the computing power of hypervisor \(i\), i.e.,

\[
\sum_{\{j | S(j) = i\}} r_j \leq C_i \hspace{1cm} \forall 1 \leq i \leq n.
\]
The profit of a solution $S$ is 

$$p(S) = \sum_{j \in X} p_j.$$ 

The transition cost of $S$ is 

$$\delta(S) = \sum_{\{i,j|S(j) = i\}} \delta_{i,j}.$$ 

In this section, we show how our framework for reapproximation via budgeted reoptimization (as given in Section 5) can be used to obtain a reapproximation algorithm for CPP.

**Theorem 6.3.** There is a randomized $(1, 1 - \frac{1}{e} - \varepsilon)$-reapproximation algorithm for CPP.

We prove the theorem using the next lemma.

**Lemma 6.1.** The base problem $\Pi$ for CPP is multiple knapsack (MKP).

**Proof.** In the base problem $\Pi$ corresponding to CPP, we have $n$ hypervisors, each having computing power $C_i$, $1 \leq i \leq n$, and $m$ virtual machines; each machine $1 \leq j \leq m$ requires $r_j$ computing units and has a profit $p_j > 0$. We can view each hypervisor $i$ as a bin of capacity $C_i$, $1 \leq i \leq n$, and each virtual machine $j$ as an item of size $r_j$ and profit $p_j$. Thus, packing feasibly a subset of the items of maximum total profit into the bins yields an optimal solution for the base problem $\Pi$, which is MKP. $\square$

In the budgeted version of CPP, we need to assign to the hypervisors a subset of virtual machines of maximal profit, such that the migration cost is at most $b$, for some $b > 0$. We can formulate this problem as the following integer program. Let $x_{ij} \in \{0, 1\}$ be an indicator for the assignment of machine $j$ to hypervisor $i$, for $1 \leq j \leq m$ and $1 \leq i \leq n$.

$$R(\Pi, b) : \max \quad \sum_i \sum_j x_{i,j} \cdot p_j$$

subject to:

- For all hypervisors $i$ 
  $$\sum_j x_{i,j} \cdot r_j \leq C_i$$

- For all virtual machines $j$ 
  $$\sum_i x_{i,j} \leq 1$$

- $$\sum_i \sum_j x_{i,j} \cdot \delta_{i,j} \leq b$$

- $$x_{i,j} \in \{0, 1\}$$

We note that budgeted CPP yields an instance of $BSAP_L$ (see Section 2.3).
Proof of Theorem 6.3. As shown by [CK05], MKP admits a PTAS. Also, Kulik presented in [Kul11] a randomized $(1, 1 - \frac{1}{e} - \varepsilon)$-approximation for $BSAP_L$. Given $\varepsilon > 0$, let $\varepsilon_1 = \frac{\varepsilon}{2 - \frac{1}{e} - \varepsilon}$, and $\varepsilon_2 = \frac{\varepsilon}{2}$. Applying Theorem 5.5 with $r_1 = 1 - \varepsilon_1$ and $r_2 = 1 - \frac{1}{e} - \varepsilon_2$, we have a $(1, r_1 \cdot r_2)$-reapproximation algorithm, where $r_1 \cdot r_2 = 1 - \frac{1}{e} - \varepsilon$. □

6.2.3 The Global Cloud Provider Problem

The global cloud provider problem (GCPP) is a generalization of CPP, in which the profits and computing units required by a virtual machines depend on the hypervisor to which it is assigned. Formally, each machine $j$ requires $r_{i,j}$ computing units and has a profit of $p_{i,j}$ when assigned to hypervisor $i$.

We show how our framework for reapproximation via budgeted reoptimization can be applied for this generalized version of CPP as well.

**Theorem 6.4.** There is a randomized $(1, 1 - \frac{2}{e} + \frac{1}{e^2} - \varepsilon)$-reapproximation algorithm for GCPP.

We prove the theorem using the next lemma.

**Lemma 6.2.** The base problem $\Pi$ for GCPP is the generalized assignment problem (GAP).

**Proof.** In the base problem $\Pi$ corresponding to GCPP, we have $n$ hypervisors, each having computing power $C_i$, $1 \leq i \leq n$, and $m$ virtual machines; each machine $1 \leq j \leq m$ requires $r_{i,j}$ computing units and has a profit $p_{i,j} > 0$ if assigned to hypervisor $i$. We can view each hypervisor $i$ as a bin of capacity $C_i$, $1 \leq i \leq n$, and each virtual machine $j$ as an item of size $r_{i,j}$ and profit $p_{i,j}$ when packed into bin $i$. Thus, packing feasibly a subset of the items of maximum total profit into the bins yields an optimal solution for the base problem $\Pi$, which is GAP. □

In the budgeted version of GCPP, we need to assign to the hypervisors a subset of virtual machines of maximal profit, such that the migration cost is at most $b$, for some $b > 0$. We note that budgeted GCPP yields an instance of $BSAP_L$.

**Proof of Theorem 6.4.** As shown by [FGMS11], GAP can be approximated within factor $(1 - \frac{1}{e} - \varepsilon)$. Also, Kulik presented in [Kul11] a randomized $(1, 1 - \frac{1}{e} - \varepsilon)$-approximation for $BSAP_L$. Thus, applying Theorem 5.5, we have the statement of the theorem. □
Chapter 7

Conclusions and Open Problems

In this thesis we studied the nature of optimization problems that need to be repeatedly solved in systems that change dynamically over time. We first considered the minimum cost reconfiguration problem, for which we developed algorithms that achieve the optimal cost, by using servers whose load capacity is increased by a small constant factor. This natural real-life scenario led to the development of a general model for combinatorial reoptimization, and the notions of reapproximation algorithm and fully polynomial time reapproximation scheme (FPTRS). We then introduced several frameworks for obtaining reoptimization and reapproximation algorithms for some well-studied classes of optimization problems, such as DP-Benevolent problems, and weighted subset selection problems. Our results distinguish 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 list below some open problems.

- Does the reoptimization version $R(\Pi)$, of any problem $\Pi$ that admits an FPTAS, has an FPTRS? We believe that it may be possible to extend our results (in Chapter 5) for problems in DP-B, to show that the answer to the above is: ‘Yes’. One direction is to obtain an FPTAS for the budgeted version of $\Pi$, and to apply the results in Section 5.2.

- How well can we approximate $R(\Pi)$ for a problem $\Pi$ that admits a PTAS? It would be interesting to develop a framework that yields polynomial time reapproximation schemes (PTRS) for such problems, or to show that such schemes may not exist.

- Lower bounds for reapproximability: We have left open the lower bounds on reapproximation ratios for the studied reoptimization problems. Clearly, such a trivial lower bound follows for $R(\Pi)$ from the lower bound on the approximability of the base problem, $\Pi$. We believe that tighter bounds exist for the reoptimization variants of these problems.

- Solving minimum cost reconfiguration as a reoptimization problem: Our study of
the minimum cost reconfiguration problem uses assumptions and measures that differ from those used in our reoptimization model. For example, we assume that the instance has a \textit{perfect placement}. Also, we allow the solution to use servers of increased load capacity. It would be of interest to study the problem within our model, for more general instances, in which only subset of the files can be placed on the servers. For such instances, a natural objective would be to maximize the amount of data placed on the servers at minimum cost.

- How efficiently can we handle local modifications? In reality, a new problem instance may result from a small modification of a previous instance. For example, suppose there is an optimal timetable for a railway network, and that a new timetable needs to be prepared, due to a close down of one station. Intuitively, in preparing the new timetable, we should benefit from having the old one.

In the \textit{cloud provider problem}, a change can be as small as the start of a new virtual machine by some user. This calls for the following question. Given a problem instance with an optimal (or, approximate) solution, and a new instance resulting from a small, local modification, how useful can the old solution be in constructing the new one? How much does it help for the runtime, how much for the quality of the output? Some previous results (see, e.g., [BFH+07, ABS10]) indicate that this additional information helps to generate solutions with better quality guarantees for some problems and does not help to find the solution more easily for other problems. For instance, it might be possible to guarantee a better approximation ratio for the reoptimization variant than for the original problem.

- In the study of reoptimization variants of NP-hard problems, suppose that there exists an $\alpha$-approximation algorithm for such optimization problem $\Pi$. Is there a polynomial time $(r, \alpha)$-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 reoptimization algorithm. The existence of such optimal algorithms for a wider class of problems remains open.
Bibliography


[GLYW08] Xiuyan Guo, Jun Li, Jian Yang, and Jinlin Wang. The research on dynamic replication and placement of file using dual-threshold


עכ. עבורי חלקי הפרקים בתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחום פיתוח בים חוף, עבורי את החומרים לתחומ

iii

Technion - Computer Science Department - Ph.D. Thesis PHD-2016-06 - 2016
 última página, el texto es un estudio de caso en el que se presenta un modelo de optimización combinatoria y se presentan herramientas para resolver problemas en este modelo. Para el desarrollo de algoritmos para optimización, se revisa la idea de un algoritmo cercano.

En particular, para funciones de objetivo NP, incluso en problemas de optimización clásicos, donde la optimización se basa en funciones de objetivo. Se indica que el uso de un algoritmo cercano se debe al hecho de que no existe un algoritmo que resuelva todos los problemas de optimización.

Para el parámetro \((r, \rho)\): se define el concepto de optimización cercana, que incluye dos parámetros \(r\) y \(\rho\). Para resolver el problema de optimización a un algoritmo cercano, se describen varias técnicas de optimización, incluyendo (FPTAS).

Elaboración de algoritmos cercanos: presenta una serie de algoritmos para resolver problemas de optimización, incluyendo el problema de optimalización de la función de objetivo. Estos algoritmos son eficientes en términos de tiempo de ejecución, especialmente si se comparan con otros algoritmos que no utilizan técnicas de optimización.

Algoritmo de optimización cercana: muestra el uso de un algoritmo cercano para resolver problemas de optimización, incluyendo el problema de optimalización de la función de objetivo. Estos algoritmos son eficientes en términos de tiempo de ejecución, especialmente si se comparan con otros algoritmos que no utilizan técnicas de optimización.

Elaboración de algoritmos cercanos: presenta una serie de algoritmos para resolver problemas de optimización, incluyendo el problema de optimalización de la función de objetivo. Estos algoritmos son eficientes en términos de tiempo de ejecución, especialmente si se comparan con otros algoritmos que no utilizan técnicas de optimización.
תקציר

המחקר תחומי של אופטימיזציה קומבינטורית המתקדד, באופיו MISS, שביעית התוכנית, וביעילות המโนוט הפורמלי. בערבי קלח הייחודי, ייעד התוכן תחומי אחד ליעל על תורת התוכנה של מטרות מוכבות, המתוחמים בין אם קיימת ליניאריית קבוצות, או בין מטרות תמורות. וביניהם, התחלתיות תקנות התוכנה ותחומיה כאשר התוכנה מתאימה מבצע של כריך מתאימה לאורח זמני. מטרה בין התוכנות של התוכנה להנדיס, על המרחב ומתקנים מגדלים בכדי להיבנות גם בתוכנה שהופכת לתחומיה. זה נועד את התוכנה ב年底前, מדענים מתאימים מדריך את הבינים של התוכנה במדינ את התוכנה נעל במתן התוכנה. למיפוי התוכנה ישאול את התוכנה במדד בתוכנה של התוכנה במדウィ קבוצות בין התוכנה. בין התוכנה,恶心 במדウィ קבוצות בין התוכנה במדウィ קבוצות בין התוכנה. למיפוי התוכנה ישאול את התוכנה במדד בתוכנה של התוכנה במדウィ קבוצות בין התוכנה.


1. הנבולה מוקדם, $C_j$ זהי מוספר התוכנה אשר ניינ יניב השם לשתי.

2. הנבולה צולת השורש, $L_j$ זהי מוספר התוכנה אשר ניינ יניב השם לשתי מערכות.
אלגוריתמים
לרה-אופטימיזציה קומבינטורית

гибור על מחקר

לשם مليי חקיקי של הר StringField לשכת ההון
דוקטור לפילוסופיה

גל טמיר

הוות להטננ הטכניים — מפורסם טכנולוגי לישראל
איך התשע”ל עיון 2016
אלגוריתמים
לרה-אופטימיזציה קומבינטורית

ג'ל תמיי