Dynamicity and Multi-Commodity in Networks

Yaron Fairstein
Dynamicity and Multi-Commodity in Networks

Research Thesis

In Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Yaron Fairstein

Sumbitted to the Senate of
Technion - Israel Institute of Technology

Kislev, 5782 Haifa November, 2021
This research thesis was done under the supervision of Prof. Joseph (Seffi) Naor and Prof. Danny Raz in the Department of Computer Science.

**Acknowledgements**

I would like to thank both of my advisors, Prof. Seffi Naor and Prof. Danny Raz, for all the time they dedicated to teaching me the foundations of research, as well as all the help and support.

I would also like to thank my family for pushing me to pursue graduate studies and supporting me throughout the process.

The generous financial help of the Technion is gratefully acknowledged.
List of Publications:


3. Fairstein Yaron, Dor Harris, Joseph (Seffi) Naor, and Danny Raz. "NFV Placement in Resource-Scarce Edge Nodes." In 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), pp. 51-60. IEEE, 2020. [34]


# Table of Contents

## List of Figures

Abstract

Abbreviations and Notation

1 Introduction

1.1 The Dynamic NFV Problem

1.1.1 Our Results

1.2 The Submodular Multiple Knapsack Problem

1.2.1 Our Results

1.3 The Capacitated MEC Problem

1.3.1 Our Results

1.4 The VM Scheduling Problem

1.4.1 Our Results

1.5 The Generalized Multistage $d$-Knapsack Problem

1.5.1 Our Results

1.6 Organization of the Thesis

2 Related Work

3 Preliminaries

3.1 Submodular Functions

3.1.1 Basic Properties of Submodular Functions

3.2 Chernoff’s Bound

3.3 Approximation Factor Preserving Reduction

4 The Dynamic NFV Placement Problem

4.1 Formulation

Technion - Computer Science Department - Ph.D. Thesis PHD-2022-03 - 2022
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>Tools and Techniques</td>
<td>44</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Interval Graph List Coloring</td>
<td>44</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Useful Procedures</td>
<td>45</td>
</tr>
<tr>
<td>4.3</td>
<td>Dynamic Facility Location</td>
<td>48</td>
</tr>
<tr>
<td>4.4</td>
<td>Dynamic Uncapacitated NFV Placement</td>
<td>50</td>
</tr>
<tr>
<td>4.5</td>
<td>Dynamic Capacitated NFV Placement</td>
<td>52</td>
</tr>
<tr>
<td>4.6</td>
<td>Interval Graph List Coloring</td>
<td>55</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Clique Size Dependent Approximation</td>
<td>55</td>
</tr>
<tr>
<td>4.6.2</td>
<td>Horizon Length Dependent Approximation</td>
<td>57</td>
</tr>
<tr>
<td>4.6.3</td>
<td>Integrality Gap</td>
<td>58</td>
</tr>
<tr>
<td>4.7</td>
<td>Experiments</td>
<td>59</td>
</tr>
<tr>
<td>5</td>
<td>The Submodular Multiple Knapsack Problem</td>
<td>64</td>
</tr>
<tr>
<td>5.1</td>
<td>Tools and Techniques</td>
<td>64</td>
</tr>
<tr>
<td>5.2</td>
<td>The Approximation Algorithm</td>
<td>66</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Structuring the Instance</td>
<td>73</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Solving a Continuous Relaxation and Rounding</td>
<td>80</td>
</tr>
<tr>
<td>6</td>
<td>The Capacitated MEC Problem</td>
<td>88</td>
</tr>
<tr>
<td>6.1</td>
<td>Submodular Formulation</td>
<td>88</td>
</tr>
<tr>
<td>6.2</td>
<td>Algorithms</td>
<td>91</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Submodularity-Based Algorithms</td>
<td>91</td>
</tr>
<tr>
<td>6.2.2</td>
<td>LP based algorithms</td>
<td>93</td>
</tr>
<tr>
<td>6.3</td>
<td>Experimental Evaluation</td>
<td>95</td>
</tr>
<tr>
<td>7</td>
<td>The VM Scheduling Problem</td>
<td>105</td>
</tr>
<tr>
<td>7.1</td>
<td>Tools and Techniques</td>
<td>105</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Static Bin Packing Algorithms</td>
<td>107</td>
</tr>
<tr>
<td>7.2</td>
<td>Offline Scheduling Algorithms</td>
<td>110</td>
</tr>
<tr>
<td>7.2.1</td>
<td>The Covering Algorithm</td>
<td>111</td>
</tr>
<tr>
<td>7.2.2</td>
<td>The Density Algorithm</td>
<td>113</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Improving the Approximation For Non-Uniform Sizes</td>
<td>117</td>
</tr>
<tr>
<td>7.3</td>
<td>Online Algorithm Using Lifetime and Average Load Predictions</td>
<td>119</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Transforming Static Bounded Space Algorithms to Non-Clairvoyant Dynamic Algorithms</td>
<td>119</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Combined Algorithm</td>
<td>124</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Lower Bound: Dynamic Clairvoyant Bin Packing</td>
<td>129</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Handling Inaccurate Predictions</td>
<td>131</td>
</tr>
<tr>
<td>7.4</td>
<td>Online Algorithm Using Lifetime and Load Vector Predictions</td>
<td>133</td>
</tr>
</tbody>
</table>
8 The Generalized Multistage $d$-Knapsack Problem 140
8.1 Approximation Scheme for $d$-GMK 140
  8.1.1 Bounded Time Horizon 141
  8.1.2 General Time Horizon 145
8.2 Approximation Scheme for Submodular $d$-GMK 156
  8.2.1 Bounded Time Horizon 157
  8.2.2 General Time Horizon 162
8.3 Hardness Results 165
9 Discussion 170
Bibliography 171
# List of Figures

1.1 Multi-access edge system reference architecture variant for MEC in NFV from [33]. ......................................................... 5

4.1 An IGLC instance with an intergrality gap of 2 ............................... 58
4.2 Performance ratio of the dynamic UNFV algorithm compared to the static UNFV with respect to the ratio between the average connection cost and the change cost. Top presents the full graph while the bottom zooms in on the smaller values. ............................................. 60
4.3 Performance ratio of the dynamic UNFV algorithm compared to the static UNFV and the fractional solution with respect to theta. .... 61
4.4 Performance ratio of the dynamic UNFV algorithm compared to the static UNFV with respect to the facility size. ......................... 62

5.1 Input and output example for Algorithm 5 with $N = 2$. The original capacities, $W$, are represented by empty rectangles, whereas the hatched rectangles represent the new capacities $\tilde{W}$. Note that the last three bins are discarded by the algorithm as they do not form a full block. 74
5.2 Illustration of the steps in the proof of Lemma 5.2.6 for $N = 3$. Each row represents a level and each box represents either a block or a super-block. The number in the box is the number of bins in the block and a gray background implies that the block is non-empty. ............... 76

6.1 Central Park cellular antennas .................................................. 96
6.2 Algorithm performance with respect to the number of clients. Network settings: 20 infrastructure nodes, 5 network functions, uniform function capacity of 5  ......................................................... 97
6.3 Algorithm performance with respect to the number network functions. Network settings: 20 infrastructure nodes, 100 clients, uniform function capacity of 5  ......................................................... 98
6.4 Algorithm performance with respect to number of network functions.
Network settings: 200 infrastructure nodes, 1000 clients, uniform function capacity of 5

6.5 Algorithm performance with respect to number of infrastructure nodes.
Network settings: clients to infrastructure nodes ratio is 5, 3 network functions, uniform function capacity of 5

6.6 Algorithm performance with respect to total functions capacity. Network settings: 100 infrastructure nodes, 500 clients, 3 network functions, uniform functions capacity.

6.7 Algorithm performance with respect to the network load. Network settings: 100 infrastructure nodes, 500 clients, 3 network functions, random network function capacities.

6.8 Algorithm performance with respect to number of network function. Network settings: 50 infrastructure nodes, 500 clients. Random function capacities selected from an exponential distribution.
Abstract

The new network generations offer high bandwidth low-latency communication that provides better services and allows the creation of new businesses. This is enabled in part by the introduction of new emerging network paradigms such as Network Function Virtualization (NFV) and Multi-access Edge Computing (MEC).

The main idea behind NFV is decoupling functionality from hardware, leading to agile networks where service location may change swiftly. MEC, on the other hand, allows to move services from large centralized data centers to the edge of the network, where resource limitations are offset by the low latency that results from short physical distance to the clients.

However, in order to utilize these paradigms and tap into their potential, one needs to deploy new resource allocation algorithms. We identify two main categories of problems that should be addressed in this context: First, multi-commodity, which refers to dividing the limited resources between the available commodities (demanded by users). Second, dynamicity, which considers the dynamic nature of networks and the workload.

In this thesis we provide several algorithmic solutions which fall into at least one of those categories. For each problem domain we study, we define a rigorous model and present an algorithmic solution for the specific problem. We provide analytically proven performance bounds for these algorithms that are compared to the relevant lower bound. For some of the problems we also present a thorough performance evaluation via extensive simulation, indicating their advantage over other solutions in realistic scenarios.
## Abbreviations and Notation

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFV</td>
<td>Network Function Virtualization</td>
</tr>
<tr>
<td>MEC</td>
<td>Multi-access Edge Computing</td>
</tr>
<tr>
<td>DFL</td>
<td>Dynamic Facility Location</td>
</tr>
<tr>
<td>Dyn-UNFV</td>
<td>Dynamic Uncapacitated NFV Placement Problem</td>
</tr>
<tr>
<td>Dyn-CNFV</td>
<td>Dynamic Capacitated NFV Placement Problem</td>
</tr>
<tr>
<td>MKP</td>
<td>Multiple Knapsack Problem</td>
</tr>
<tr>
<td>SMKP</td>
<td>Submodular Multiple Knapsack Problem</td>
</tr>
<tr>
<td>CMAP</td>
<td>Capacitated MEC Problem</td>
</tr>
<tr>
<td>$d$-MKCP</td>
<td>Multiple $d$-Knapsack Constraint Problem</td>
</tr>
<tr>
<td>$d$-GMK</td>
<td>Generalized Multistage $d$-Knapsack Problem</td>
</tr>
<tr>
<td>GAP</td>
<td>Generalized Assignment Problem</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

New network generations offer low-latency communication with very high bandwidth. This allows for the deployment of new applications that were previously infeasible. The development of these networks is enabled by new emerging network paradigms such as Network Function Virtualization (NFV), which decouples network hardware from network functionality. This is achieved by implementing services in software and running them on Commercial-Off-the-Shelf (COTS) servers. NFV, together with Software Defined Network (SDN), allow operators to support even more complex in-network-services in a cost-effective way (see [12, 75]).

Multi-access Edge Computing (MEC) is another important network paradigm that enables the development of new networks, by allowing network operators to serve an increasing number of clients, demanding a growing number of services [33, 48, 58]. MEC dictates that clients are served by services that are placed at the edge of the network. Even though the resources at the edge are are much smaller compared to large data centers, shorter physical distance to the clients shortens latency as well. The NFV/SDN paradigm facilitates MEC by making the network design flexible and thus allowing network operators to modify the services they offer within seconds\(^1\).

These paradigms are important building blocks in the development of new networks. However, in order to utilize these paradigms and tap into their potential, one needs to

\(^1\)See https://www.sdxcentral.com/edge/definitions/mec-nfv/
deploy new resource allocation algorithms. We identify two main categories of problems that should be addressed in this context. First, multi-commodity, which refers to dividing the limited resources between the available commodities. Second, dynamicity, which considers the dynamic nature of the networks and the workload.

The first category of problems, multi-commodity, includes precisely those resource allocation problems raised by the MEC architecture. For example, Figure 1.1 depicts the recent MEC reference architecture for MEC and NFV from [33]. The MEC architecture Orchestrator (MEAO), the component making the function placement decisions, is highlighted in the figure. This component is described in Annex A.1: MEC host selection in [33]. The text there describes the input to the placement algorithm (requests can either come from the OSS, a third-party, or a device application) that contains information about the application to run, and possibly more information, such as the location where the application needs to be active, other application rules, and requirements.
Then it states:

“The multi-access edge orchestrator considers the requirements and information listed above and information on the resources currently available in the MEC system to select one or several MEC hosts within the MEC system, and requests the selected host(s) to instantiate the application. NOTE: The actual algorithm used to select the hosts depends on the implementation, configuration, and operator deployment and is not intended to be specified.”

The second category we address, dynamic problems, describe the dynamic nature of networks. Overtime network operators may open and close servers and offer new services. In addition, the set of clients might change and clients may change their location. This means that any static solution for a single time unit might be unsuitable as the network evolves. Thus, the solution must take into account the dynamic nature of the optimization problem.

Many of the initial NFV deployments concentrated on somewhat static applications like virtual customer premises equipment (vCPE), where demand is either generated in private customers’ homes or in business customers’ offices, tending to have fixed locations. However, there are new emerging applications and network functions that are much more dynamic in nature, due both to mobility and rapid change in demand. One such an example is vRAN, where the low layer wireless technology is implemented in NFV, and demand depends on the current load of the base stations. Another example is the area of self driving cars and drone controls; here, the mobility of the platform, together with fluctuations in demand, creates a highly dynamic service. A third class of fast growing applications having a very dynamic workload are enhanced video and virtual reality applications. In these applications there is a need for real time latency-bounded services, while the workload can rapidly change over time. This calls for an adjustment of the orchestration modules in order to allow for dynamic allocation of resources in smaller time frames and live migration of services, as needed.

The following lists the problems studied in this thesis.

**The Dynamic NFV Problem (DNFV).** This is a generalization of the Dynamic Facility Location Problem (DFL) and the Generalized Assignment Problem
(GAP). Similarly to DFL each client must be assigned to a facility which serves its demand, though different clients may require different commodities. In addition, limited resources at the facilities restrict the allocation of commodities as in GAP. The objective is to find a feasible assignment of clients to facilities that minimizes the distances between clients and the facility to which they are assigned, as well as the cost of installing commodities in facilities.

The Submodular Multiple Knapsack Problem (SMKP). This is a generalization of the classic Multiple Knapsack Problem, where the objective function is a monotone submodular set function. Given a set of items with weights and a set of bins with capacities, the goal is to find an assignment of items to bins that does not violate their capacities, while maximizing the objective function.

The Capacitated MEC Problem (CMAP) In CMAP we are given a set of clients, each demanding some network function. The goal is to assign clients to edge nodes in which the function they demand is allocated. Due to scarce resources at the edge of the network, the number of functions allocated to each node is restricted, as well as the number of clients each allocated function can serve. The objective is to find a feasible allocation of functions to edge nodes and a feasible assignment of clients to edge nodes which maximizes the total profit of satisfied clients.

The VM Scheduling Problem The VM Scheduling Problem solves the problem of allocating Virtual Machine (VM) requests into physical machines. The physical machines have limited resources, restricting the number of VM requests allocated to them. As each request is associated with a start and end time, the VM scheduling problem can be described by the Dynamic Bin Packing problem. In this work we consider both offline and online variants of the problem. In addition, two extensions of the online variant are considered, where additional noisy predictions of future requests are given.
The Generalized Multistage $d$-Knapsack Problem ($d$-GMK) The $d$-GMK problem is a multistage extension of a generalization of the Multiple Knapsack problem as well as the $d$-Dimensional Knapsack problem. Given a fixed set of items, at each time unit we must pack a subset of items in a bin. Continuous packing of an item accrues additional gains. On the other hand, a solution is taxed for changes in the packing (packing of a new item or a removal of a packed item).

1.1 The Dynamic NFV Problem

The Dynamic NFV placement problem is defined over a time horizon. We are given a set of demands, each composed of a set of flows, one for each time step, and a set of functions from which it requires service. The functions are to be installed on network servers. Each server has a size constraint; each function has both a size and cost for installing it at each server. In addition, each function has a capacity that bounds the number of flows it can serve. We are also given a change cost paid to change the assignment of a demand to different servers between two consecutive time steps. The objective is to find an assignment that minimizes the total cost of installing the functions at the servers, the sum of the distances of the flows from the servers and the sum of change costs. In our model, demands are represented by clients, servers by facilities, and network functions by network commodities.

This can be seen as a generalization of Dynamic Facility Location (DFL). In DFL we are given a time horizon $T$, and an undirected graph (or network) $G = (V, E)$, equipped with a distance function $d^t(\cdot, \cdot)$ between any pair of nodes at each time step $t \in T$. The distance functions induce a metric space over the graph. We are given a set $F \subseteq V$ of $m$ facilities and a set $C \subseteq V$ of $n$ clients. For a facility $i \in F$ and client $j \in C$, $i$ and $j$ indicate both facility and client, respectively, as well as where they reside at each time step. We assume that facilities remain at fixed locations in the network, while the evolution of the metric reflects the movement of clients over time, and thus change their distance to the facilities.

The installation cost of facility $i$ is denoted by $f_i$. We are also given a change cost $g$, paid for each change in assignment of a client to a facility. The objective in DFL
is to assign each client, at each time step, to an installed facility while minimizing the overall cost, comprising of the sum of installation costs, the sum of distances between the clients and the facilities to which they are assigned (paid for each commodity separately), which we also call connection costs, and the sum of the change costs.

In the DFL problem all clients require the same commodity. The Dynamic Uncapaciated NFV placement problem (Dyn-UNFV) extends it as each client demands a specific commodity. In particular, there is a set $S$ of $k$ network commodities; for each client $j \in C$, $\delta(j)$ denotes the subset of commodities that it requires. Each facility $i \in F$ has a total size $w_i$, and each commodity $s$ occupies size $w_{is}$ on facility $i$. The installation cost of a commodity $s$ at facility $i$ is denoted by $f_{is}$. In the Dynamic Capacitated NFV placement problem (Dync-CNFV), each commodity $s$ is also associated with a capacity $\mu_s$, a bound on the number of clients it can serve. To accommodate more clients, several copies of commodity $s$ can be installed at facility $i$, however, each copy occupies size $w_{is}$ and pays cost $f_{is}$.

In a feasible solution to the dynamic NFV placement problem we find an allocation of commodities to facilities, and an assignment of clients to facilities, such that at each time step each client $j \in C$ is assigned to a subset of facilities that can serve all the commodities in $\delta(j)$. To comply with the constraints, a solution must fulfill the requirement that the sum of the sizes of the commodities installed at a facility does not exceed its size. In Dync-CNFV it is also required that at each time step, the number of clients served by a (copy of a) commodity does not exceed its capacity. Similar to DFL, the objective is to find a feasible solution minimizing the overall cost, comprising of the sum of the installation costs, the sum of the connection costs, and the sum of the change costs.

1.1.1 Our Results

Our main contributions are the introduction of a new temporal model for dynamic NFV placement together with approximation algorithms with proven performance guarantees. Our new model contains both NFV placement [27] and dynamic facility location [32], capturing the most important aspects of dynamic NFV placement. We do
not make any assumptions on how our network evolves over time, however, we do assume that we are given (ahead of time) the changes in the network between time steps. We note that in the NFV setting it is reasonable to assume that servers remain at fixed locations over time, and only clients are dynamic and can change their location. This allows us to consider the NFV placement problem in a dynamic setting that takes into account the different stages a network goes through, thus developing algorithmic solutions that better fit the NFV setting.

The models we consider generalize facility location which is known to be NP-hard. Therefore, we turn to approximate solutions that can be computed efficiently. We first revisit the dynamic facility location problem studied by [32]. By assuming facilities are static, we obtain an elegant 7-approximation algorithm for this problem, improving over the $O(\log nT)$ approximation algorithm of Eisenstein et al. [32]. We note that this improvement is essential for constant factor approximations when extending the NFV setting to the dynamic setting.

We distinguish between two versions of the dynamic NFV placement problem: Dynamic Uncapacitated NFV (Dyn-UNFV) and Dynamic Capacitated NFV (Dyn-CNFV) which extends the former. In Dyn-CNFV there is a capacity constraint, a limit on the number of clients that can receive service from a network function. Our results for these problems are summarized in the following theorems.

**Theorem 1.1.1.** There exists an efficient algorithm for the Dyn-UNFV problem with an $(O(1), O(1))$ bi-criteria approximation algorithm, where we approximate the overall network cost by a constant factor, while exceeding the size constraints of the servers by at most a constant factor.

The next theorem contains our main contribution, which is also the most technically challenging part of our work on Dynamic NFV.

**Theorem 1.1.2.** There exists an efficient algorithm for the Dyn-CNFV problem with an $(O(1), O(1), \log(\min\{n, T\}))$ tri-criteria approximation algorithm, where we approximate the overall network cost by a constant factor, while exceeding the size constraints by at most a constant factor.

---

2In [32] it is assumed, in contrast, that the full metric (i.e., facilities and clients) changes in each time step.
of the servers by at most a constant factor, and the capacity constraints by at most a factor of $O(\log \min\{n, T\})$. Here, $n$ is the number of clients and $T$ is the number of time steps.

We also show that the problem of selecting the final assignment of clients to functions in the Dyn-CNFV problem reduces to the interval graph list coloring problem. In this problem we are given an interval graph and a palette of colors. For each interval a subset of the color set is given. The goal is to find a coloring of the intervals such that: (i) each interval is colored by a color from its allowed subset of colors; (ii) intersecting intervals are colored differently. As this problem is NP-hard, we turn to find an approximate solution with a bounded size set of intersecting intervals that receive the same color. We prove the following:

**Theorem 1.1.3.** There exists an efficient algorithm for the Interval Graph List Coloring problem that finds a coloring in which at most $O(\log \min\{k, T\})$ intersecting intervals receive the same color, where $k$ is the size of the largest clique in the interval graph, and $T$ is the number of distinct cliques (time steps) in the interval graph.

We accompany these results with a lower bound proof for the IGLC problem, showing our algorithm is almost tight.

In order to show that our new algorithms indeed improve network utilization under dynamic workloads, we conduct a real scenario simulation based performance evaluation. In this evaluation we compare the performance of the dynamic algorithm to two variants of the static algorithm (presented by [27]), and to the optimal fractional solution to the dynamic problem computed using an LP-solver. The results indicate that the dynamic algorithm preforms at least 40% better than the static algorithm (in some cases up to 2-3 times better), and the comparison to the fractional solution indicates that in the considered practical scenarios, our algorithm is at most twice the optimal (fractional) solution.
1.2 The Submodular Multiple Knapsack Problem

The Submodular Multiple Knapsack Problem is a generalization of the Multiple Knapsack Problem, where the objective function is monotone and submodular. The input consists of a set of \( n \) items \( I \) and \( m \) bins \( B \). Each item \( i \in I \) is associated with a weight \( w_i \geq 0 \), and each bin \( b \in B \) has a capacity \( W_b \geq 0 \). We are also given an oracle to a non-negative monotone submodular function \( f : 2^I \to \mathbb{R}_{\geq 0} \). A feasible solution to the problem is a tuple of \( m \) subsets \((A_b)_{b \in B}\) such that for every \( b \in B \) it holds that \( \sum_{i \in A_b} w_i \leq W_b \). The value of a solution \((A_b)_{b \in B}\) is \( f(S) \), where \( S = \bigcup_{b \in B} A_b \). The objective is to find a feasible solution of maximum value.

The problem is a natural generalization of both Multiple Knapsack [20] (where \( f \) is modular or linear), and the problem of monotone submodular maximization subject to a knapsack constraint [83] (where \( m = 1 \)).

1.2.1 Our Results

Our main result is stated in the next theorem.

**Theorem 1.2.1.** For any \( \varepsilon > 0 \), there is a randomized \((1 - e^{-1} - \varepsilon)\)-approximation algorithm for SMKP.

A \((1 - e^{-1})\) hardness of approximation bound is known for the problem under \( P \neq \text{NP} \), due to the hardness of max-\( k \)-cover [39] which is a special case of SMKP. This is a vast improvement over previous results. Feldman presented in [42] a \((\frac{e-1}{e^{-1}} - o(1)) \approx 0.24\)-approximation for the special case of identical bin capacities, along with a \( \frac{1}{6} \)-approximation for general capacities. To the best of our knowledge, this is the best known approximation ratio for the problem.\(^3\)

Simultaneously and independently to our work, Sun et. al. [81] presented a deterministic greedy based \((1 - e^{-1} - \varepsilon)\)-approximation for the special case of identical bins. In a later version [82], which appeared after the publication of the preliminary version

\(^3\)Sun et. al. [81] indicate that a \((1 - e^{1-e^{-1}} - o(1)) \approx 0.468\)-approximation for the problem can be derived using the techniques of [18]. We note that this derivation is non-trivial (no details were given in [18]).
of this paper, Sun et. al. derived a randomized \((1 - e^{-1} - \varepsilon)\)-approximation for general SMKP instances, matching our result, by using a different approach.

1.3 The Capacitated MEC Problem

In the Capacitated MEC Allocation problem (CMAP) we are given a set of \(n\) clients \(C\), a set of \(m\) infrastructure nodes \(I\), and a set of links \(E \subseteq C \times I\) defining connections between clients and infrastructure nodes. In addition, there is a set of \(k\) network functions \(F\) that can be allocated to the infrastructure nodes. Thus, an allocation \(A\) of functions to infrastructure nodes satisfies \(A \subseteq I \times F\).

For each client \(c\) we are given a network function \(h(c) \in F\) it demands, as well as a profit \(p_c \in \mathbb{R}_{\geq 0}\) associated with it. A client is said to be satisfied if it is assigned to an infrastructure node on which network function \(h(c)\) resides, in which case the client’s profit is accrued. Note that the case of multiple network function demand is also captured here. This is true since a client that requires more than one network function can equivalently be viewed as multiple clients, each demanding a single function\(^4\).

We are also given size and capacity constraints. The size constraint of infrastructure node \(i\) limits the total number of network functions allocated to \(i\) to be at most \(w_i\). The capacity constraints limit the number of clients that can be assigned to network functions. That is, the number of clients that can be assigned to network function \(f\), residing on infrastructure node \(i\), is at most \(s_{i,f}\).

We note that even the uncapacitated version of CMAP is NP-hard. This can be proved using the following reduction from the NP-hard problem *Max SAT with no mixed clauses* [52]. For a given CNF formula \(\phi\) we define two functions \(F = \{t, f\}\) (true and false). For each literal \(x_i\) we define an infrastructure node \(i\). On each infrastructure node only a single function can be installed. In addition, a client \(c\) is defined for each CNF clause \(\phi_c\), and is connected to the infrastructure nodes that represent the literals in \(\phi_c\). If \(\phi_c\) is composed of positive literals, \(h(c) = t\); otherwise, \(h(c) = f\). It is easy to see that any solution to the uncapacitated CMAP instance defines a solution to formula

\(^4\)The much more complicated case, where a client is satisfied only if all the network functions it requires are provided, is beyond the scope of this thesis.
A solution to CMAP can be broken down into two parts. First, finding a feasible allocation of network functions to infrastructure nodes, i.e., an allocation that does not violate the size constraints. Second, finding a feasible assignment of clients to infrastructure nodes. An assignment is feasible if: (i) a client with demand \( h(c) \) is assigned only to an infrastructure node on which \( h(c) \) is allocated; (ii) capacity constraints are not violated. The goal in CMAP is to find a solution which maximizes the total profit accrued from satisfied clients.

We now formulate an integer program for CMAP. Let \( x_{ic} \) indicate whether client \( c \) is assigned to infrastructure node \( i \), and let \( y_{if} \) indicate whether function \( f \) is installed on infrastructure node \( i \). Integer program (IP-CMAP) is:

\[
\begin{align*}
\max & \sum_{c \in C} p_c \cdot x_{ic} \\
\text{subject to} & \\
(1) & \sum_{i \in I} x_{ic} \leq 1, \quad \forall c \in C \\
(2) & \sum_{c \in C} x_{ic} \leq y_{if} \cdot s_{if}, \quad \forall i \in I, f \in F \\
(3) & \sum_{f \in F} y_{if} \leq w_i, \quad \forall i \in I \\
(4) & x_{ic} \leq y_{if}, \quad \forall i \in I, c \in C, h(c) = f \\
(5) & x_{ic} = 0, \quad \forall c \in C, i \in I, (c, i) \notin E \\
(6) & x_{ic}, y_{if} \in \{0, 1\}, \quad \forall c \in C, i \in I, f \in F
\end{align*}
\]

The first constraint prohibits assigning a client more than once. The second and third constraints are the capacity and size (respectively) constraints of the infrastructure nodes. The fourth constraint prevents assigning a client to an infrastructure node on which the function it requires is not installed. The fifth constraint limits the assignment of clients to infrastructure nodes they are connected to. Lastly, the sixth constraint is an integrality requirement on the variables.
1.3.1 Our Results

We utilize two different mathematical formulations to study CMAP. The first formulation is based on a linear relaxation of an integer program that captures CMAP. In the second formulation we consider CMAP as a covering problem, where the goal is to maximize the coverage of sets that are function-infrastructure node pairs. This view allows us to formulate CMAP as a constrained submodular maximization problem.

We present four algorithms that approximate CMAP. The first two algorithms, presented in the first row of Table 1.1, are based on a linear formulation of CMAP. Both algorithms solve the linear relaxation of the integer program, and then the fractional solution is rounded to achieve an approximation ratio of $(1 - \epsilon)(1 - \frac{1}{2})$ (where $\epsilon$ depends on the capacities of the network functions). The last two algorithms, presented in the second row of Table 1.1, are based on a submodular formulation of CMAP. The first algorithm is the Discrete Greedy algorithm that achieves a $\frac{1}{2}$ approximation ratio by selecting in each step the function-infrastructure node pair with the largest profit margin. The second algorithm, the Continuous Greedy algorithm, uses a continuous extension of submodular functions, called the multilinear extension. This continuous extension allows for making small increments in the solution during the run of the algorithm. We show that Continuous Greedy has an approximation ratio of $1 - \frac{1}{e} - o(1)$.

In Table 1.2 the complexity of the four algorithms is given (where $n$ denotes the number of clients, $m$ denotes the number of infrastructure nodes, and $k$ is the number of different network functions). Note that the complexity of the LP rounding algorithms is dominated by the complexity of solving the LP. As for the greedy algorithms, even though the Continuous Greedy algorithm achieves a better approximation factor than the Discrete Greedy, it comes with a higher complexity price.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>LP Fully Random</th>
<th>LP Max Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approx. Ratio</td>
<td>$(1 - \epsilon)(1 - \frac{1}{2})$</td>
<td>$(1 - \epsilon)(1 - \frac{1}{2})$</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Discrete Greedy</td>
<td>Continuous Greedy</td>
</tr>
<tr>
<td>Approx. Ratio</td>
<td>$\frac{1}{2}$</td>
<td>$1 - \frac{1}{e} - o(1)$</td>
</tr>
</tbody>
</table>

Table 1.1: Approximation Results
### Table 1.2: Algorithms Complexity

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>LP Fully Random</th>
<th>LP Max Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>$O(nm + mk)^3$</td>
<td>$O(nm + mk)^3$</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Discrete Greedy</td>
<td>Continues Greedy</td>
</tr>
<tr>
<td>Complexity</td>
<td>$O(n^4m^3k)$</td>
<td>$O(n^2m^9k^4)$</td>
</tr>
</tbody>
</table>

To evaluate the practical performance of our algorithms we simulated them in realistic scenarios. We used New York cellular antennas located around Central Park as edge nodes\(^5\). In 5G networks these cellular locations will become edge nodes supporting compute, storage, and network resources. Randomly located clients are connected to these edge nodes in a fixed radius around their location. This simulates the cellular reception boundaries in realistic scenarios. A comprehensive simulation study is performed on this network layout with multiple network settings. The simulation results indicate that our algorithms perform better than currently used heuristics. In addition, our algorithms outperform their respective analytical bounds.

### 1.4 The VM Scheduling Problem

In the VM Scheduling problem we are given VM requests, each modeled as a time interval $I = [s, e)$; we often use the term interval when referring to a VM request. Each interval is associated with a start time, $s_I$, end time, $e_I$, and a size, $w_I \leq 1$. The intervals are scheduled on machines/bins whose size is normalized to 1. We say that $t \in I$ if $s_I \leq t < e_I$. Let $\ell_I = e_I - s_I$ be the length of interval $I$. We assume without loss of generality that the minimum length of an interval is 1, and denote by $\mu$ the maximum length of an interval (which is not necessarily known in advance). Let $\beta = \max_I w_I$ be the maximum size of an interval (which, again, is not necessarily known in advance).

In the \textit{uniform size model} the size of all intervals is $\frac{1}{g}$ for some integer value $g$. In the \textit{non-uniform size model} the size of each interval is arbitrary.

The \textit{static bin packing} problem is a classic NP-hard problem in which the goal is to pack a set of items of varying sizes, while minimizing the number of bins used. The

\(^5\)see https://opencellid.org
problem has been studied extensively in both offline and online settings [25, 56, 71].
The problem of allocating VMs to physical machines is equivalent to the dynamic bin
packing problem in which items (VMs) arrive over time and later depart [26]. The goal
is to minimize the total usage time of the bins which is the same as minimizing the total
time the machines are active [4, 67, 69, 78]. In the online setting items arrive over time,
and in the non-clairvoyant case no information is given to the scheduler upon arrival
of a new item, while in the clairvoyant setting the departure time (or duration) of an
item is revealed upon its arrival.

A machine is said to be active or open at time $t$ if at least one VM is running on
it. Our goal is to schedule the VMs so as to minimize the total (or equivalently the
average) number of active machines over the time horizon. We assume that the cloud
capacity is large enough, so that VM requests can always be accommodated. Without
loss of generality, we further assume that at each time $t$ there is at least one active
request (otherwise, the time horizon can be partitioned into separate time horizons).

Let $\mathcal{I}$ be a set of all intervals (VM requests). We define $\mathcal{I}(t) = \{I \in \mathcal{I} | t \in I\}$ as the
set of intervals that are active at time $t$, and let $N_t = |\mathcal{I}(t)|$. Let $v = (v_1, v_2, \ldots, v_T)$ be
the load vector over time, where $v_t = |\sum_{I \in \mathcal{I}(t)} w_I|$. In our analysis, we use several norms
of the load vector: $\|v\|_1 = \sum_{t=1}^{T} v_t$, $\|v\|_\infty = \max_{t=1}^{T} \{v_t\}$, and $\|v\|_0 = \sum_{t=1}^{T} 1_{(N_t > 0)}$ (i.e.,
the total number of time epochs in which there is at least one active VM request).
In addition, let $v_{\text{avg}} = \frac{\|v\|_1}{T}$ be the average value of the load vector (or the average
demand). Throughout the thesis, we will use load vector notions not only for the set $\mathcal{I}$
of all intervals, but also for different subsets $S \subseteq \mathcal{I}$. In every such use case, we describe
explicitly the corresponding subset. A simple (known) lower bound on the value of the
optimal solution, using our load vector notation, is the following:

**Observation 1.4.1.** The total active-machine time required by any scheduler is at least
$\|v\|_1$.

We next provide a useful lemma about intersecting intervals (intervals that are all
active at the same time $t$). We use this lemma frequently in our algorithms’ analyses
to guarantee that in each such set, there is one interval that sees a high load, i.e., the
total load is above a certain threshold, for its whole duration.
Lemma 1.4.2 (Intersecting intervals). Let $\mathcal{I}$ be a set of intervals with load vector $v$ that are all using time $t$ (i.e., $t \in I$ for all $I \in \mathcal{I}$). For any $\alpha > 0$, if $v_t > \alpha$, then there exists an interval $I \in \mathcal{I}$ such that $v_{t'} > \alpha/2$ for all $t' \in I$.

Proof. Let $\alpha > 0$ be a lower bound on the load at time $t$, i.e., $v_t > \alpha$. Since all intervals in $\mathcal{I}$ are active at time $t$, it can be seen that their load vector is non-decreasing until time $t$, and non-increasing after time $t$. Let $I_1, I_2, \ldots, I_J \in \mathcal{I}$ be the intervals sorted by their starting times (which are all prior to $t$). Let $A_1 = \{I_1, \ldots, I_j\} \subseteq \mathcal{I}$ be such that $\sum_{i=1}^{j} w_{I_i} \leq \alpha/2$, but $\sum_{i=1}^{j+1} w_{I_i} > \alpha/2$. For each interval in $\mathcal{I} \setminus A_1$ the load at its starting point is strictly more than $\alpha/2$. Similarly define $A_2 = \{I_k, \ldots, I_J\}$ such that $\sum_{i=k}^{J} w_{I_i} \leq \alpha/2$, but $\sum_{i=k-1}^{J} w_{I_i} > \alpha/2$. and let $\mathcal{I} \setminus A_2$ be the subset of intervals whose load in their endpoint is strictly more than $\alpha/2$. As the total load in $\mathcal{I}$ is strictly more than $\alpha$, and the load of intervals in $A_1 \cup A_2$ is at most $\alpha$, there must be an interval $I \in \mathcal{I} \setminus (A_1 \cup A_2)$. The load that an interval $I \in \mathcal{I} \setminus (A_1 \cup A_2)$ observes is strictly more than $\alpha/2$ at both its start and end times, and hence it is strictly more than $\alpha/2$ at any $t' \in I$. \hfill \Box

1.4.1 Our Results

As there is always at least one active VM in each time step, the optimal cost of the dynamic bin packing problem is at least $T$, the length of the time horizon. To facilitate the understanding of our results, we divide both the optimal cost, $OPT$, as well as our algorithm’s cost by $T$. Let $OPT_{avg}$ denote the optimal cost divided by $T$. The value $OPT_{avg}$ should thus be read as the average number of machines used by an optimal solution. This change, of course, does not affect the multiplicative factor in the approximation/competitive ratios we get. However, any additive term should now be read as the average number of additional machines our algorithm is using over the (average) number of machines an optimal solution is using. We note that in our cloud computing context the average number of machines is typically in the order of thousands. Finally, let $k$ denote the parameter of the known harmonic bin packing
algorithm, and let $\Pi_k$ be its asymptotic competitive ratio.\footnote{The harmonic algorithm is parameterized by $k$, which controls an additive term in its competitive ratio. $\Pi_k$ is a monotonically decreasing number that approaches $\Pi_\infty \approx 1.691$. $\Pi_k$ quickly becomes close to 1.691, for example, $\Pi_6 = 1.7$ and $\Pi_{12} \approx 1.692$.}

We study the VM scheduling problem in both offline and online settings.

**Offline Algorithms**

We first show the following result for the offline problem.

**Theorem 1.4.3.** For any integer $k \geq 3$, there exist offline scheduling algorithms whose average cost is at most:

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Non-uniform size case</th>
<th>Uniform size case</th>
</tr>
</thead>
<tbody>
<tr>
<td>14, 15</td>
<td>$\Pi_k \cdot OPT_{avg} + O\left(\sqrt{OPT_{avg} \cdot k \cdot \log \mu}\right)$</td>
<td>$OPT_{avg} + O\left(\sqrt{OPT_{avg} \cdot \log \mu}\right)$</td>
</tr>
<tr>
<td>13, 15</td>
<td>$2\Pi_k(1 + \frac{1}{k^2}) \cdot OPT_{avg} + k$</td>
<td>$2 \cdot OPT_{avg}$</td>
</tr>
<tr>
<td>Previous</td>
<td>$4 \cdot OPT_{avg}$ [78]</td>
<td>$2 \cdot OPT_{avg}$ [2, 62]</td>
</tr>
</tbody>
</table>

The $2 \cdot OPT_{avg}$ upper bound for the uniform size case is well known [2, 62, 78]. However, it is described here not only to compare against our other results, but also because the techniques used to prove it are later used in the online case; though fairly simple, these techniques are somewhat different from previous proofs. In the above theorem we obtain two improved new bounds for the offline case. These improvements are in the spirit of the asymptotic approximation ratio commonly used in the standard bin packing problem. Algorithm 13 has multiplicative approximation ratio $2\Pi_k(1 + \frac{1}{k^2})$, while using extra $k$ machines in each time step. Thus, its “asymptotic” approximation approaches $2\Pi_\infty \approx 3.38$, which is better than the best known (strict) 4-approximation for the problem.\footnote{We remark that Algorithm 13 also achieves the same 4-approximation (with no additive term).}

Algorithm 14 achieves an even better multiplicative approximation ratio of $\Pi_k$ (that approaches 1.69), albeit only when the average number of machines used by an optimal solution is relatively large. Specifically, it uses an extra $O\left(\sqrt{OPT_{avg} \cdot k \cdot \log \mu}\right)$
machines in each time step. If the average number of machines used in the optimal solution is much larger than $\log \mu$, the latter additive term becomes negligible compared to $\text{OPT}_{avg}$. In the uniform size case, the asymptotic approximation of the algorithm is 1, as may be expected when sizes are uniform. When the maximum demand of any VM is small (and also in some other scenarios), our performance guarantees are better than those outlined in Theorem 1.4.3 (actually, in both offline and online settings). We refer the reader to Section 7.2 and Section 7.4 for more details.

**Online Algorithms with Additional Information**

Our main contribution in this work is the construction of new online algorithms that have access to additional information on future demand, leading to improved competitive ratios. Interestingly, our online algorithms are inspired by their offline counterparts. Earlier results assumed that an online scheduler gets no extra information upon arrival of a VM request. The performance guarantee of these algorithms turned out to be very poor in many cases. Better results were later obtained for the clairvoyant model, in which duration of requests are revealed upon arrival.

Specifically, we explore two novel models in which the scheduler is provided with predictions about demand. In the first model, the average load is known to the scheduler (a single value), and in the second model the total load in each of the future time steps is known to the scheduler. We remark that predicting future cumulative demand is much simpler than obtaining the full structure of an instance, which requires predicting future arrivals of individual requests.

**Theorem 1.4.4.** For any integer $k \geq 2$, there exist online scheduling algorithms with average cost at most:
In the above table, we compare our results with the previously best known online result in the clairvoyant model, due to Azar and Vainstein [4]. As indicated in the table, [4] designed an algorithm whose total cost is at most \(O(\sqrt{\log \mu}) \cdot \text{OPT}_{\text{avg}}\), and proved that this ratio is optimal. Our results demonstrate that with more information the competitive ratio can be dramatically improved. Suppose that the only additional information provided is the average load (taken over the full time horizon), and that the average number of machines used is much larger than \(\log \mu\); then, we obtain a constant competitive ratio that approaches \(\Pi_{\infty} \approx 1.69\) in the non-uniform size case and an asymptotic ratio of 1 in the uniform size case. Thus, our performance guarantee is always better than [4], and it is the same when the average load is \(O(1)\).

If the load at all future times is known, we achieve a (strict) constant competitive ratio under no additional assumptions. This is in contrast to the \(\Omega(\sqrt{\log \mu})\) lower bound on the competitive ratio of any algorithm without this extra knowledge [4].

In Section 7.3.4 we complement our results and analyze the performance of our algorithms when the average load prediction, as well as interval lengths predictions, are inaccurate.

We complement the above results by generalizing the lower bound of [4] to take into account also \(\text{OPT}_{\text{avg}}\) showing that the additive term \(O(\sqrt{\text{OPT}_{\text{avg}} \cdot \log \mu})\) is indeed unavoidable, if only the average future load (and lifetime) is available to an online algorithm.

**Theorem 1.4.5.** The average cost of any online algorithm is at least \(\Omega(\sqrt{\text{OPT}_{\text{avg}} \cdot \log \mu})\). The bound holds even for the uniform size case and with prior knowledge of the average load and \(\mu\).

\(^8\)We note that similarly to the algorithm of [4], our algorithm also does not need to know the value of \(\mu\) upfront.
1.5 The Generalized Multistage $d$-Knapsack Problem

In this section we present the Generalized Multistage $d$-Knapsack problem ($d$-GMK). We begin with an informal description of the problem. An instance of the problem consists of $T$ stages, where in each stage we are given an instance of a generalization of the classic knapsack problem. While the instances differ between stages, in all stages the same set of items $I$ can be packed. The continuity of the solution is enforced by quantifying the similarity of consecutive solutions and integrating it into the objective function.

We quantify continuity by four types of values. The first two values specify gains earned for the similarity of solutions. For example, if an item $i$ is packed in stages $t-1$ and $t$, gain $g_{i,t}^+$ is awarded. Similarly, $g_{i,t}^-$ is awarded if $i$ is not packed in $t-1$ and $t$. The other two values define the cost of changes between consecutive solutions. For example, if an item $i$ was not packed in stage $t-1$, and it is decided to pack it in stage $t$, a change cost of $c_{i,t}^+$ is charged. Similarly, $c_{i,t}^-$ is charged if $i$ is packed in $t$, but not in $t+1$.

The packing problem at each stage generalizes the Multiple Knapsack problem, as well as the $d$-Dimensional Knapsack problem. In each instance of the problem we are given $d$ sets of bins, and the weight an item occupies in a bin depends on the set to which the bin belongs to. The profit of an item is accrued once it is assigned to some bin in all $d$ sets of bins. This problem is called $d$-Multiple Knapsack Constraints Problem and is formally defined below.

A Multiple Knapsack Constraint (MKC) is a tuple $K = (w, B, W)$ defined over a set of items $I$. The function $w: I \rightarrow \mathbb{R}_+$ defines the weight of the items, $B$ is a set of bins, each equipped with a capacity defined by the function $W: B \rightarrow \mathbb{R}_+$. An assignment is a function $A: B \rightarrow 2^I$, defining which items are assigned to each of the bins. An assignment is feasible if $w(A(b)) = \sum_{i \in A(b)} w_i \leq W(b)$ for each bin $b \in B$. Similarly, given a tuple of MKCs $K = (K_j)_{j=1}^d$ over $I$, a tuple of $d$ assignments $A = (A_j)_{j=1}^d$ is feasible for $K$ if for each $j = 1, \ldots, d$ assignment $A_j$ is a feasible assignment for $K_j$. We say $A$ is an assignment of set $S \subseteq I$ if $S = \bigcup_{b \in B} A_b$. 
In \textit{d-Multiple Knapsack Constraints Problem (d-MKCP)}, a problem first introduced in [37], we are given a tuple \((I, \mathcal{K}, p)\), where \(I\) is a set of items, \(\mathcal{K}\) is a tuple of \(d\) MKCs and \(p : I \rightarrow \mathbb{R}_{\geq 0}\) defines the profit of each item. A feasible solution for \(d\)-MKCP is a set \(S \subseteq I\) and a tuple of feasible assignments \(\mathcal{A}\) (w.r.t \(\mathcal{K}\)) of \(S\). The goal is to find a feasible solution that maximizes \(p(S) = \sum_{i \in S} p(i)\). We note that if there exists an item with negative profit it can be discarded in advance. This fact is used later on, in Section 8.1.1.

The \textit{Generalized Multistage} \(d\)-\textit{Knapsack problem (d-GMK)}, is the multistage model of \(d\)-MKCP. The problem is defined over a time horizon of \(T\) stages as follows. An instance of the problem is a tuple \(\left(\mathcal{P}_t\right)_{t=1}^T\), where \(\mathcal{P}_t = (I, \mathcal{K}_t, p_t)\) is a \(d_t\)-MKCP instance with \(d_t \leq d\) for \(t \in [T]\), \(g^+, g^- \in \mathbb{R}_+^{I\times[2,T]}\) are the gain vectors and \(c^+, c^- \in \mathbb{R}_+^{I\times[1,T]}\) are the change cost vectors.\(^9\) We use \(g_{i,t}^+\) and \(g_{i,t}^-\) to denote the gain of item \(i\) at stage \(t\). Similarly, we use \(c_{i,t}^+\) and \(c_{i,t}^-\) to denote the change cost of item \(i\) at stage \(t\).

A feasible solution for \(d\)-GMK is a tuple \((S_t, \mathcal{A}_t)_{t=1}^T\), where \((S_t, \mathcal{A}_t)\) is a feasible solution for \(\mathcal{P}_t\) (note that \(\mathcal{A}_t\) is a tuple of assignments of \(S_t\)). Throughout the thesis we assume \(S_0 = S_{T+1} = \emptyset\) and denote the objective function of instance \(Q\) by \(f_Q : I^T \rightarrow \mathbb{R}\), where

\[
f_Q\left((S_t)_{t=1}^T\right) = \sum_{t=1}^T \sum_{i \in S_t} p_t(i) + \sum_{t=2}^T \left( \sum_{i \in S_{t-1} \cap S_t} g_{i,t}^+ + \sum_{i \notin S_{t-1} \cup S_t} g_{i,t}^- \right)
- \sum_{t=1}^T \left( \sum_{i \in S_t \setminus S_{t-1}} c_{i,t}^+ + \sum_{i \in S_t \setminus S_{t+1}} c_{i,t}^- \right).
\]

The goal is to find a feasible solution that maximizes the objective function \(f_Q\).

A study of \(d\)-GMK reveals it does not admit a constant factor approximation algorithm (see Section 8.3). We found that in hard instances the change costs are much larger than the profits. Thus we consider an important parameter of the problem, the \textit{profit-cost ratio}. It is defined as the maximum ratio, over all items, between the change costs and the profits.

\(^9\)We use the notations \([n, m] = \{i \in \mathbb{N} \mid n \leq i \leq m\}\) and \([n] = [1, n]\) for \(n, m \in \mathbb{N}\).
cost \((c^+, c^-)\) and the profit of an item over all stages. It is denoted by \(\phi_Q\) for any instance \(Q\), and is formally defined as

\[
\phi_Q = \min \left\{ \infty \right\} \bigcup \left\{ r \geq 0 \mid \forall i \in I, t_1, t_2 \in [T] : \max \left\{ c^+_{i,t_1}, c^-_{i,t_1} \right\} \leq r \cdot p_{t_2}(i) \right\}
\]

We show that \(d\)-GMK instances where the profit-cost ratio is bounded by a constant admit a PTAS.

We also consider Submodular \(d\)-GMK, a submodular variant of \(d\)-GMK where the profit functions are replaced with monotone submodular set functions. It can also be viewed as the multistage model of Submodular \(d\)-MKCP. In Submodular \(d\)-MKCP we are given a tuple \((I, \mathcal{K}, p)\), where \(I\) is a set of items, \(\mathcal{K}\) is a tuple of \(d\) MKCs and \(p : 2^I \rightarrow \mathbb{R}_{\geq 0}\) is a non-negative, monotone and submodular set function, defining the profit of a subset \(S \subseteq I\). A feasible solution for Submodular \(d\)-MKCP is a set \(S \subseteq I\) and a tuple of feasible assignments \(A\) (w.r.t \(\mathcal{K}\)) of \(S\). The goal is to find a feasible solution that maximizes \(p(S)\).

The \(d\)-GMK problem is defined over a time horizon of \(T\) stages as follows. An instance of the problem is a tuple \(((\mathcal{P}_t))_{t=1}^T, g^+, g^-\), where \(\mathcal{P}_t = (I, \mathcal{K}_t, p_t)\) is a Submodular \(d_t\)-MKCP instance with \(d_t \leq d\) for \(t \in [T]\) and \(g^+, g^- \in \mathbb{R}_{\geq 0}^{I \times [2,T]}\) are the gains vectors.

We use \(g^+_{i,t}\) and \(g^-_{i,t}\) to denote the gain of item \(i\) at stage \(t\).

A feasible solution for Submodular \(d\)-GMK is a tuple \((S_t, A_t))_{t=1}^T\), where \((S_t, A_t)\) is a feasible solution for \(\mathcal{P}_t\) (note that \(A_t\) is a tuple of assignments of \(S_t\)). The objective function of instance \(Q\) is denoted by \(f_Q : I^T \rightarrow \mathbb{R}\), where

\[
f_Q \left( (S_t)_{t=1}^T \right) = \sum_{t=1}^T p_t(S_t) + \sum_{t=2}^T \left( \sum_{i \in S_{t-1} \cap S_t} g^+_{i,t} + \sum_{i \notin S_{t-1} \cup S_t} g^-_{i,t} \right)
\]

The goal is to find a feasible solution that maximizes the objective function \(f_Q\).

Both \(d\)-GMK and Submodular \(d\)-GMK generalize the Multistage Knapsack problem recently considered by Bampis et al. [8]. There are several aspects by which it is generalized. First, handling multiple knapsack constraints as well as \(d\)-dimensional knapsack vs a single knapsack in [8]. Second, the profit earned from assigning items
can be described as a submodular function, not only by a modular function. Third, [8] considered only symmetric gains, i.e., the same gain is earned whether an item is assigned or not assigned in consecutive stages. Lastly, change costs were not considered in [8].

1.5.1 Our Results

Our main result is stated in the following theorem.

Theorem 1.5.1. For any fixed $d \in \mathbb{N}$ and $\phi \geq 1$ there exists a randomized PTAS for $d$-GMK with a profit-cost ratio bounded by $\phi$.

The result uses the general framework of [8], in which the authors first presented an algorithm for instances with bounded time horizon, and then showed how it can be scaled for general instances. To handle bounded time horizons we show an approximation factor preserving reduction (as defined in [85]\textsuperscript{10}) from $d$-GMK to a generalization of $q$-MKCP. The reduction illuminates the relationship between $d$-GMK and $q$-MKCP. As $q$-MKCP admits a PTAS [37], this results in a PTAS for $d$-GMK instances with a bounded time horizon.

We note the reduction can be applied to the problem considered in [8] as well. In this case the target optimization problem is $d$-dimensional knapsack with a matroid constraint. As the latter problem is known to admit a PTAS [49], this suggests a simpler solution for bounded time horizon in comparison to the one given in [8].

To generalize the result to unbounded time horizon we use an approach similar to [8], though a more sophisticated analysis was required to handle the change costs. The generalization is achieved by cutting the time horizon into sub-instances with a fixed time horizon. Each sub-instance is solved separately, and then the solutions are combined to create a solution for the full instance. Handling change costs is trickier as cutting an instance may lead to an excessive charge of change costs at the cut points. We must compensate for these additional costs, or we will not be able to bound the value of the solution.

\textsuperscript{10}A formal definition is provided in Section 3.3 for completeness.
The results for the modular variant generalizes the PTAS for Multistage Knapsack [8]. For \( d \geq 2 \), we cannot expect better results as even \( d \)-KP, also generalized by \( d \)-GMK, does not admit an efficient PTAS (EPTAS). Theorem 1.5.2 shows an EPTAS cannot be obtained for 1-GMK as well.

**Theorem 1.5.2.** Unless \( W[1] = FPT \), there is no EPTAS for 1-GMK, even if the length of the time horizon is \( T = 2 \), the set of bins in each MKC contains one bin and there are no change costs.

The theorem is proved using a simple reduction from 2-dimensional knapsack. Bampis et al. [8] considered a similar withered down instance and proved that even if the gains are symmetric (i.e., \( g_{i,t}^+ = g_{i,t}^- \)) a Fully PTAS (FPTAS) does not exist for the problem.

Using a reduction from multidimensional knapsack we show that 1-GMK, in its general form, cannot be approximated to any constant factor.

**Theorem 1.5.3.** For any \( d \geq 1 \), there is no polynomial time approximation algorithm for \( d \)-GMK with a constant approximation ratio, unless \( NP = ZPP \).

This result justifies our study of the special cases of \( d \)-GMK in which the profit-cost ratio is bounded by a constant.

The techniques used to develop the algorithm for \( d \)-GMK can be adjusted slightly to produce an approximation algorithm for Submodular \( d \)-GMK.

**Theorem 1.5.4.** For any fixed \( d \in \mathbb{N} \) and \( \epsilon > 0 \) there exists a randomized \( 1 - \frac{1}{e} - \epsilon \) - approximation algorithm for Submodular \( d \)-GMK.

In the submodular variant one cannot hope for vast improvement over our results as the algorithm is almost tight. This is due to the hardness results for submodular maximization subject to a cardinality constraint presented by Nemhauser and Wolsey [73].
1.6 Organization of the Thesis

The rest of the thesis is organized as follows. In Chapter 2 we discuss related work. In Chapter 3 we present the needed definitions and notations. Chapter 4 is devoted to the study of the Dynamic NFV Placement Problem. Chapter 5 is devoted to the study of the Submodular Multiple Knapsack Problem. Chapter 6 is devoted to the study of the Capacitated MEC Problem. Chapter 7 is devoted to the study of the VM Scheduling Problem. Chapter 8 is devoted to the study of the Generalized Multistage $d$-Knapsack Problem. We conclude in Chapter 9 with a summary of the thesis.
Chapter 2

Related Work

**Dynamicity.** In the multistage model we are given a series of instances of an optimization problem, and we search for a solution which optimizes each instance while maintains some similarity between solutions. It was first introduced by Gupta et al. [51] and Eisenstat et al. [32] to address dynamic environments. Since its introduction, it has received growing attention and was used to address dynamicity of problems such as matching, clustering, subset sum, vertex cover and minimum \( s - t \) path [3, 6–8, 24, 31, 38, 44, 45].

Two different ideas were used to enforce a balance between single stage optimality and continuity. In [32, 51] a change cost is charged for the dissimilarity of consecutive solutions, while in [8] additional gains were given for their similarity. In the aforementioned cloud management problem, the change cost can be interpreted as either an installation cost, or an eviction cost charged when a client is initially served and then its service is discontinued. The gains can be modeled as increased costs the client is charged to guarantee the continuity of its service.

In the classic (and extensively studied) static bin packing problem the goal is to pack a set of items of varying sizes, while minimizing the number of bins used [25, 56, 71]. The dynamic bin packing problem in which items arrive over time and later depart [26] can be seen as a multistage model of static bin packing. The assignment of each item is selected upon its arrival and it cannot be changed throughout its lifetime. The objective is to Minimize the total usage time of the bins [4, 67, 69, 78]. The dynamic bin packing problem is of interest in both the uniform size case, in which all items have the same
size (and each bin can pack at most \( g \) items) \[43\], but especially under the more general setting, which we refer to as the \textit{non-uniform size} case. Coffman et al. \[26\], which first introduced the problem with the objective of minimizing the maximum number of active machines over the time horizon, and designed an 2.788-competitive algorithm. For this model, Wong et al. \[89\] obtain a lower bound of \( \frac{8}{3} \approx 2.666 \) on the competitive ratio.

The dynamic bin packing problem has also been studied in the online setting, where items arrive over time, giving rise to two different models. In the \textit{non-clairvoyant} case \[43, 67, 84\] no information is given to the scheduler upon arrival of a new item, and indeed only poor performance is obtained when there is a large variation in item duration times \[57\] (see additional discussion later). In the \textit{clairvoyant} setting \[4, 69, 78\] the departure time (or duration) of an item is revealed upon arrival, allowing for significant performance improvements.

The clairvoyant model assumes that highly accurate lifetime predictions are available to a scheduler. In the cloud context, this information has recently been obtained through Machine Learning (ML) tools \[10, 28, 68\], which are deployed to support resource management decisions for the underlying systems (see \[13, 28, 47, 66\] and references therein). ML is increasingly used, not only for lifetime prediction, but also to predict other metrics, such as machine health \[28\] and future demand \[50\]. Finally, we note that there has been growing interest in designing such resource management algorithms with ML-assisted (and potentially inaccurate) predictions; see, e.g., recent work on online caching \[70\], scheduling \[63\] and the ski-rental problem \[76\].

\textbf{Multi-Commodity.} Facility location is a well known family of problems that deals with selecting locations for facilities providing service to a set of clients. It is commonly assumed that each facility has an opening cost and there is a given metric space defining distances between clients and possible locations of facilities (see e.g., \[88\]). The goal is to open facilities, as well as assigning clients to facilities, minimizing the total cost of opening facilities and serving the clients. Of special relevance to us is the multi-commodity facility location (MCFL) problem, introduced by Ravi et al. \[77\]. In this problem there is a set of commodities that can be installed (opened) at the facilities,
incurring an installation cost for each commodity. Each client requires a subset of the commodities, and the goal is to satisfy the requirements of all the clients by connecting them to facilities that provide the commodities they need, while minimizing the total service and installation costs\(^1\).

Cohen et al. [27], following the paradigm of MCFL problems, defined a model for the NFV placement problem that takes into account special properties of the NFV setting. The input is a set of clients, representing network flows, each requiring service from a subset of the network functions, or commodities. The network functions are installed on servers at various parts of the network. Locating the functions is performed in a practical environment in which servers have limited space for allocating network functions. The overall cost is defined as the sum of the installation costs, reflecting the cost of having VMs that execute a function, and the cost of diverting traffic to these servers. The goal is to locate network functions in a way that minimizes overall network cost, while adhering to limited space (size) for installing functions on servers, and the constraint that each function can serve only a limited number (capacity) of clients. They presented a bicriteria approximation algorithm for the problem, approximating the value of an optimal solution within a constant factor and violating the size constraints by at most a constant factor.

Recently, the topic of VNF placement has been addressed in multiple studies. A simpler version of the MEC allocation problem was considered in [53], where capacity constraints are ignored. This, of course, does not capture the limitations due to having scarce resources at the edge nodes. Moens and De Turck [72] studied the problem of VNF placement in hybrid networks, where the available servers across the network are either physical or virtual. Indeed, their model takes into account many parameters, however, the proposed solutions are based on solving an ILP, and as such cannot be used on large scale networks due to time limitations. Basta et al. [9] described a model for VNF placement in network gateways which somewhat resemble MEC models, however, their proposed solution is also based on solving an ILP and thus is not scalable. Moreover, their solutions do not have any analytical guarantees.

\[^1\]The specific model considered in [77] captures the set cover problem, and therefore only a logarithmic factor approximation could be obtained for the problems studied therein.
Bahreini and Grosu [5] provided a time dependant model for MEC placement with server communication costs. They formulated a model based on MILP for both offline and online variants of their problem. Since their model is based on MILP, it cannot be scaled for large networks due to the complexity of solving such linear/integer programs. Yala et al. [90] presented a latency derived model for VNF placement in a MEC environment that also takes into account infrastructure failure probability. Their designed solution is based on a genetic algorithm and tries to maximize both network availability and low latency service. Both of these models do not consider edge limited resources in their respective solutions, and do not guarantee any analytical bounds on their solutions.

More recently, Sallam and Ji [79] showed that one can utilize submodular optimization techniques to approximate the single network function case (which is much simpler). We use a similar approach and show how submodular optimization coupled with a decomposition technique can be used to yield approximation algorithms for the multi-function case. Furthermore, it allows us to capture matroid constraints, multiple network functions, and a general profit objective function.

The multiple knapsack problem (MKP) is one of the most natural extensions of the classic Knapsack problem arising also in the context of Virtual Machine (VM) allocation in cloud computing. The practical task is to assign VM requests of clients to physical machines such that capacity constraints are satisfied, while maximizing the profit of the cloud provider. A submodular cost function allows cloud providers to offer complex cost models to high-volume clients. The price customers pay for each VM can depend on the overall number of machines used by the customer.

A polynomial time approximation scheme for MKP was first presented by Chekuri and Khanna [20]. The authors also ruled out the existence of a fully polynomial time approximation scheme for the problem. An efficient polynomial time approximation scheme was later developed by Jansen [54,55].
Chapter 3

Preliminaries

In this chapter we introduce the definitions and notation that are used throughout the thesis.

3.1 Submodular Functions

Submodular optimization has recently attracted much attention as it provides a unifying framework capturing many fundamental problems in combinatorial optimization, economics, algorithmic game theory, networking, and other areas. Furthermore, submodularity also captures many real-world practical applications where economy of scale is prevalent. Classic examples of submodular functions are coverage functions [39], matroid rank functions [17] and graph cut functions [40]. A recent survey on submodular functions can be found in [15].

Submodular functions are defined over sets. Given a ground set $I$, a function $f: 2^I \to \mathbb{R}_{\geq 0}$ is called submodular if for every $A \subseteq B \subseteq I$ and $i \in I \setminus B$, $f(A \cup \{i\}) - f(A) \geq f(B \cup \{i\}) - f(B)$.\footnote{Equivalently, for every $A, B \subseteq I$: $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$.} This reflects the diminishing returns property: the marginal value from adding $i \in I$ to a solution diminishes as the solution set becomes larger. A set function $f: 2^I \to \mathbb{R}$ is monotone if for any $A \subseteq B \subseteq I$ it holds that $f(A) \leq f(B)$. While in many cases, such as coverage and matroid rank function, the submodular function...
is monotone, this is not always the case (cut functions are a classic example). Given a
monotone submodular function \( f : 2^I \rightarrow \mathbb{R}_{\geq 0} \) and a set \( S \subseteq I \), we define \( f_S : 2^I \rightarrow \mathbb{R}_{\geq 0} \) by \( f_S(A) = f(S \cup A) - f(S) \). It follows that \( f_S \) is a monotone, non-negative submodular
function (see Claim 3.1.3 in 3.1.1).

In [73] Nemhauser and Wolsey presented a greedy based \((1 - e^{-1})\)-approximation for
maximizing a monotone submodular function subject to a cardinality constraint, along
with a matching lower bound in the oracle model. A \((1 - e^{-1})\) hardness of approximation
bound is also known for the problem under \( P \neq \text{NP} \), due to the hardness of max-
\( k \)-cover [39] which is a special case. The greedy algorithm of [73] was later generalized to
monotone submodular optimization with a knapsack constraint [59,83].

Many modern submodular optimization algorithms rely on the submodular Multi-
linear Extension (see, e.g., [16,17,22,41,61,65,86,87]). Given a function \( f : 2^I \rightarrow \mathbb{R}_{\geq 0} \),
its multilinear extension is \( F : [0,1]^I \rightarrow \mathbb{R}_{\geq 0} \) defined as:

\[
F(\bar{x}) = \sum_{S \subseteq I} f(S) \prod_{i \in S} \bar{x}_i \prod_{i \in I \setminus S} (1 - \bar{x}_i).
\]

The multilinear extension can be interpreted as an expectation of a random variable.
Given \( \bar{x} \in [0,1]^I \) we say that a random set \( X \) is distributed according to \( \bar{x} \), \( X \sim \bar{x} \),
if \( \Pr(i \in X) = \bar{x}_i \) and the events \( (i \in X)_{i \in I} \) are independent. It follows that \( F(\bar{x}) = E_{X \sim \bar{x}}[f(X)] \).

The unified greedy algorithm of [41] can be used to find approximate solution for
maximization problems of the form \( \max F(\bar{x}) \) s.t. \( \bar{x} \in P \), where \( F \) is the multilinear
extension of a monotone submodular function \( f \), and \( P \) is a down-monotone polytope.
The algorithm uses two oracles, one for \( f \) and another which given \( \bar{\lambda} \in \mathbb{R}^I \) returns a
vector \( \bar{x} \in P \) such that \( \bar{x} \cdot \bar{\lambda} \) is maximal. The algorithm returns \( \bar{x} \in P \) such that \( F(\bar{x}) \geq (1 - e^{-1} - o(1)) \max_{\bar{y} \in P \cap \{0,1\}^I} F(\bar{y}) \). The result can also obtained via the continuous
greedy of [18].

3.1.1 Basic Properties of Submodular Functions

We utilize several basic properties of submodular functions in our analysis.
Claim 3.1.1. Let \( f : 2^I \to \mathbb{R} \) be a set function and \( R \subseteq I \). Define \( g : 2^I \to \mathbb{R} \) by \( g(S) = f(S \cup R) \) for any \( S \subseteq I \). Then,

1. If \( f \) is submodular then \( g \) is submodular.
2. If \( f \) is monotone then \( g \) is monotone.

Proof.

1. Assume \( f \) is submodular. Let \( S, T \subseteq I \). Then,

\[
g(S) + g(T) = f(R \cup S) + f(R \cup T) \\
\geq f((R \cup S) \cup (R \cup T)) + f((R \cup S) \cap (R \cup T)) \\
= f(R \cup (S \cup T)) + f(R \cup (S \cap T)) \\
= g(S \cup T) + g(S \cap T)
\]

Hence, \( g \) is submodular.

2. Assume \( f \) is monotone, and let \( S \subseteq T \subseteq I \). Then \( R \cup S \subseteq R \cup T \), and therefore

\[
g(S) = f(R \cup S) \leq f(R \cup T) = g(T).
\]

Thus, \( g \) is monotone.

\[\square\]

Claim 3.1.2. Let \( f : 2^I \to \mathbb{R}_{\geq 0} \) be monotone and submodular function, then for any \( T_1 \subseteq T_2 \subseteq I \) and \( A \subseteq I \), it holds that \( f(T_1 \cup A) - f(T_1) \geq f(T_2 \cup A) - f(T_2) \).

Proof. By the submodularity of \( f \), we have

\[
f(T_1 \cup A) + f(T_2) \geq f(T_1 \cup A \cup T_2) + f((T_1 \cup A) \cap T_2) \geq f(T_2 \cup A) + f(T_1),
\]

(3.1)

where the second inequality follows from \( T_1 \subseteq (T_1 \cup A) \cap T_2 \) and the monotonicity of \( f \). By rearranging the terms in (3.1), we have

\[
f(T_1 \cup A) - f(T_1) \geq f(T_2 \cup A) - f(T_2)
\]
as required.

Claim 3.1.3. Let \( f : 2^I \rightarrow \mathbb{R}_{\geq 0} \) be a non-negative, monotone and submodular function, and let \( S \subseteq I \). Then, \( f_S \) is a submodular, monotone and non-negative function.

Proof. By Claim 3.1.1, it holds that the function \( g : 2^I \rightarrow \mathbb{R} \), defined by \( g(R) = f(S \cup R) \) for any \( R \subseteq I \), is monotone and submodular. Since \( f_S \) is the difference between \( g \) and a constant, it is submodular and monotone as well.

It remains to show \( f_S \) is non-negative. As \( f \) is monotone, for any \( R \subseteq I \) it holds that \( f(S \cup R) \geq f(S) \). Thus, \( f_S(R) = f(S \cup R) - f(S) \geq 0 \).

Claim 3.1.4. Let \( f : 2^I \rightarrow \mathbb{R}_{\geq 0} \) be a non-negative, monotone and submodular function, and let \( E \subseteq 2^I \times X \) for some set \( X \) (each element of \( E \) is a pair \((S, h)\) with \( S \subseteq I \) and \( h \in X \)). Then the function \( g : 2^E \rightarrow \mathbb{R}_{\geq 0} \) defined by \( g(A) = f(\cup_{(S, h) \in A} S) \) is non-negative, monotone and submodular.

It is important to emphasize that Claim 3.1.4 does not hold for non-monotone submodular functions.

Proof of Claim 3.1.4. It is easy to see that \( g \) is non-negative, as \( f \) is non-negative. In addition, for any two subsets \( A \subseteq B \subseteq E \), we have \( \cup_{(S, h) \in A} S \subseteq \cup_{(S, h) \in B} S \). Thus, since \( f \) is monotone, \( g \) is monotone as well.

We now show that \( g \) is submodular. Consider subsets \( A \subseteq B \subseteq E \) and \((S, h) \in E \setminus B\).

\[
g(A \cup \{(S, h)\}) - g(A) = f(\cup_{(S', h') \in A} S' \cup S) - f(\cup_{(S', h') \in A} S') \\
\leq f(\cup_{(S', h') \in B} S' \cup S) - f(\cup_{(S', h') \in B} S') \\
= g(B \cup \{(S, h)\}) - g(B).\]

The inequality follows from Claim 3.1.2 and \( \cup_{(S', h') \in A} S' \subseteq \cup_{(S', h') \in B} S' \).

The following is a similar claim, providing a slightly different extension for a submodular function.
Claim 3.1.5. Let \( f : 2^I \to \mathbb{R}_{\geq 0} \) be a non-negative, monotone and submodular function, and let \( E \subseteq I \times 2^X \) for some set \( X \) (each element of \( E \) is a pair \( (e, H) \) with \( e \in I \) and \( H \subseteq X \)). Then for any \( h \in X \) the function \( g : 2^E \to \mathbb{R}_{\geq 0} \) defined by \( g(A) = f \left( \{ i \mid \exists (i, H) \in A : h \in H \} \right) \) is non-negative, monotone and submodular.

Proof. Let \( S, T \subseteq E \), then

\[
g(S) + g(T) = f \left( \{ i \mid \exists (i, H) \in S : h \in H \} \right) + f \left( \{ i \mid \exists (i, H) \in T : h \in H \} \right)
\geq f \left( \{ i \mid \exists (i, H) \in S : h \in H \} \cup \{ i \mid \exists (i, H) \in T : h \in H \} \right)
+ f \left( \{ i \mid \exists (i, H) \in S \cap T : h \in H \} \right)
\geq f \left( \{ i \mid \exists (i, H) \in S \cup T : h \in H \} \right) + f \left( \{ i \mid \exists (i, H) \in S \cap T : h \in H \} \right)
= g(S \cup T) + g(S \cap T)
\]

where the first inequality is by submodularity, and the second inequality uses

\[
\{ i \mid \exists (i, D) \in S : t \in D \} \cap \{ i \mid \exists (i, D) \in T : t \in D \} \subseteq \{ i \mid \exists (i, D) \in S \cup T : t \in D \}
\]

and the monotonicity of \( f \). Thus \( g \) is submodular.

Let \( S \subseteq T \subseteq E \), then

\[
g(S) = f \left( \{ i \mid \exists (i, H) \in S : h \in H \} \right) \leq f \left( \{ i \mid \exists (i, H) \in T : h \in H \} \right) = g(T),
\]

where the inequality holds since \( f \) is monotone. Thus \( g \) is monotone as well.

Finally, for any \( S \subseteq E \),

\[
g(S) = f \left( \{ i \mid \exists (i, H) \in S : h \in H \} \right) \geq 0,
\]

as \( f \) is non-negative. Thus \( g \) is non-negative. \( \square \)

Lemma 3.1.6. Let \( h : 2^I \to \mathbb{R} \) be a submodular function and let \( S_1, \ldots, S_N \subseteq I \) be disjoint sets. Then there is \( 1 \leq r^* \leq N \) such that

\[
h \left( \bigcup_{1 \leq r \leq N; \ r \neq r^*} S_r \right) \geq \left( 1 - \frac{1}{N} \right) h(S_1 \cup \ldots \cup S_N).
\]
Proof of Lemma 3.1.6. We can write

$$h(S_1 \cup \ldots \cup S_N) - h(\emptyset) = \sum_{r=1}^{N} (h(S_1 \cup \ldots \cup S_r) - h(S_1 \cup \ldots \cup S_{r-1})).$$

Therefore, there is $1 \leq r^* \leq N$ such that

$$h(S_1 \cup \ldots \cup S_{r^*}) - h(S_1 \cup \ldots \cup S_{r^*-1}) \leq \frac{1}{N} (h(S_1 \cup \ldots \cup S_N) - h(\emptyset)).$$

Thus,

$$\left(1 - \frac{1}{N}\right) (h(S_1 \cup \ldots \cup S_N) - h(\emptyset)) \leq h(S_1 \cup \ldots \cup S_N) - h(\emptyset) - h(S_1 \cup \ldots \cup S_{r^*}) + h(S_1 \cup \ldots \cup S_{r^*-1})$$

$$= h\left((S_1 \cup \ldots \cup S_N \setminus S_{r^*}) \cup (S_1 \cup \ldots \cup S_{r^*-1})\right)$$

$$+ h\left((S_1 \cup \ldots \cup S_N \setminus S_{r^*}) \cap (S_1 \cup \ldots \cup S_{r^*})\right)$$

$$- h(S_1 \cup \ldots \cup S_{r^*}) - h(\emptyset)$$

$$\leq h\left(S_1 \cup \ldots \cup S_N \setminus S_{r^*}\right) + h(S_1 \cup \ldots \cup S_{r^*}) - h(S_1 \cup \ldots \cup S_{r^*}) - h(\emptyset)$$

$$= h\left(\bigcup_{1 \leq r \leq N, r \neq r^*} S_r\right) - h(\emptyset),$$

where the second inequality follows from the submodularity of $h$. The first and last equalities use the property that $S_1, \ldots, S_N$ are disjoint. By rearranging the terms, we have

$$h\left(\bigcup_{1 \leq r \leq N, r \neq r^*} S_r\right) \geq \left(1 - \frac{1}{N}\right) h(S_1 \cup \ldots \cup S_N) + \frac{1}{N} \cdot h(\emptyset) \geq \left(1 - \frac{1}{N}\right) h(S_1 \cup \ldots \cup S_N).$$

\[\square\]
3.2 Chernoff’s Bound

In the analysis of Algorithm 7 we use the following Chernoff-like bounds.

Lemma 3.2.1 (Theorem 3.1 in [46]). Let $X = \sum_{i=1}^{n} X_i \cdot \lambda_i$ where $(X_i)_{i=1}^{n}$ is a sequence of independent Bernoulli random variable and $\lambda_i \in [0, 1]$ for $1 \leq i \leq n$. Then for any $\varepsilon \in (0, 1)$ and $\eta \geq \mathbb{E}[X]$ it holds that

$$\Pr (X > (1 + \varepsilon)\eta) < \exp \left(- \frac{\varepsilon^2}{3\eta} \right)$$

Lemma 3.2.2 (Theorem 1.3 in [21]). Let $I = \{1, \ldots, n\}$, $\nu > 0$ and $f : 2^I \rightarrow \mathbb{R}_+$ be a monotone submodular function such that $f(\{i\}) - f(\emptyset) \leq \nu$ for any $i \in I$. Let $X_1, \ldots, X_n$ be independent random variables and $\eta = \mathbb{E}[f(\{i \in I|X_i = 1\})]$. Then for any $\varepsilon > 0$ it holds that

$$\mathbb{E}[f(\{i \in I|X_i = 1\})] \leq (1 - \varepsilon)\eta \leq \exp \left(- \frac{\eta \cdot \varepsilon^2}{2\nu} \right)$$

3.3 Approximation Factor Preserving Reduction

Throughout the thesis an approximation factor preserving reduction is utilized to prove lower and upper bounds. The following provides a formal definition of this reduction.

Definition 3.3.1. Let $\Pi_1, \Pi_2$ be two maximization problems. An approximation factor preserving reduction from $\Pi_1$ to $\Pi_2$ consists of two polynomial time algorithms $f, g$ such that for any two instances $I_1$ of problem $\Pi_1$ and $I_2 = f(I_1)$ of problem $\Pi_2$ it holds that

- $I_2 \in \Pi_2$ and $OPT_{\Pi_2}(I_2) \geq OPT_{\Pi_1}(I_1)$.
- for any solution $s_2$ for $I_2$, solution $s_1 = g(I_1, s_2)$ is a solution for $I_1$ and $obj_{\Pi_1}(I_1, s_1) \geq obj_{\Pi_2}(I_2, s_2)$.

where $OPT_{\Pi}(I)$ is the value of an optimal solution for instance $I$ of problem $\Pi$, and $obj_{\Pi}(I, s)$ is the value of solution $s$ for instance $I$ of problem $\Pi$. 

40
Chapter 4

The Dynamic NFV Placement Problem

In this section we study a set of dynamic NFV placement problems, including the Dynamic Capacitated NFV Placement problem (Dyn-CNFW) and its special cases, the Dynamic Uncapacitated NFV Placement problem (Dyn-UNFW) and Dynamic Facility Location problem (DFL). We first start by formulating the problem as a linear program. This is followed by several important tools and techniques are presented. Next, approximation algorithms for DFL, Dyn-UNFW and Dyn-CNFW are presented. Then, a solution for the Interval Graph List Coloring problem is presented, on which the approximation algorithm for the Dyn-CNFW relies. Finally, we evaluate the performance of the algorithm for Dyn-UNFW via simulation.

4.1 Formulation

In the dynamic NFV problems we are given a time horizon $T$, and an undirected graph (or network) $G = (V, E)$, equipped with a distance function $d^t(\cdot, \cdot)$ between any pair of nodes at each time step $t \in [T]$. The distance functions induce a metric space over the graph. We are given a set $F \subseteq V$ of $m$ facilities and a set $C \subseteq V$ of $n$ clients. For a facility $i \in F$ and client $j \in C$, $i$ and $j$ indicate both facility and client, respectively,
as well as where they reside at each time step. We assume that facilities remain at
fixed locations in the network, while the evolution of the metric reflects the movement
of clients over time, and thus change their distance to the facilities. There is a set $S$ of
$k$ network commodities; for each client $j \in C$, $\delta(j)$ denotes the subset of commodities
that it requires. We are also given a change cost $g$, paid for each change in assignment
of a client to a facility.

Each facility $i \in F$ has a total size $w_i$, and each commodity $s$ occupies size $w_{is}$ on
facility $i$. The installation cost of a commodity $s$ at facility $i$ is denoted by $f_{is}$. In Dyn-
CNFV each commodity $s$ is also associated with a capacity $\mu_s$, a bound on the number
of clients it can serve. To accommodate more clients, several copies of commodity $s$
can be installed at facility $i$, however, each copy occupies size $w_{is}$ and pays cost $f_{is}$.

In a feasible solution to the dynamic NFV placement problem we find an allocation
of commodities to facilities, and an assignment of clients to facilities such that at each
time step, each client $j \in C$ is assigned to a subset of facilities that can serve all
the commodities in $\delta(j)$. To comply with the constraints, a solution must fulfill the
requirement that the sum of the sizes of the commodities installed at a facility does not
exceed its size. In Dyn-CNCFV it is also required that at each time step, the number
of clients served by a (copy of a) commodity does not exceed its capacity. The goal
is to find a feasible solution minimizing the overall cost, comprising of the sum of
installation costs, the sum of distances between the clients and the facilities to which
they are assigned (paid for each commodity separately), which we also call connection
costs, and the sum of change costs.

We formulate the dynamic NFV placement problem as a linear program (DNFV-
LP). We denote by $y_{is}$ the variable indicating the number of copies of commodity
$s$ installed in facility $i$, and by $x_{tij}$ the variable indicating whether facility $i$ serves
commodity $s$ to client $j$ at time step $t$. Variable $z_{tij}$ indicates a change in the assignment.

$$
\min \sum_{i \in F} \sum_{s \in S} f_{is} y_{is} + \sum_{t \in [T]} \sum_{j \in C} \sum_{s \in \delta(j)} d_{tij} x_{tij} + \sum_{t \in [T]} \sum_{j \in C} \sum_{s \in \delta(j)} z_{tij} \cdot g
$$

subject to:
Constraint (1) states that a facility cannot provide service of a commodity unless it is installed in it. Constraint (2) guarantees that each client is served all the commodities it requires. Constraint (3) bounds the total size of the commodities installed in each facility. Constraint (4) charges for connection changes between consecutive time steps. Constraint (5) limits the number of clients each commodity can serve.

### 4.2 Tools and Techniques

In this section we present the Interval Graph List Coloring problem as well as several useful procedures, all used in the development of an algorithm for Dyn-CNFV.

#### 4.2.1 Interval Graph List Coloring

We are given an interval graph \( G = (V, E) \), i.e., an intersection graph of intervals, where each vertex \( v \in V \) corresponds to an interval \( I_v \) on the real line, and there is an edge \( e = (u, v) \in E \) if and only if \( I_u \) and \( I_v \) intersect. In addition, there is a set of colors \( C \), and each vertex \( v \) is associated with a subset \( C_v \subseteq C \) of colors by which it can be legally colored. The goal is to find a legal coloring of the vertices such that neighboring vertices receive different colors. This is called the interval graph list coloring problem (IGLC).

Let \( T \) be the set of points on the real line containing all start and end points of the
intervals. Obviously, the set $T$ defines the set of maximal cliques (w.r.t. containment) in $G$. We denote the clique at point $t \in T$ by $I(t)$, i.e., the intervals that contain point $t$. Let $x_{vc}$ indicate whether vertex $v$ is colored by color $c$. We define an integer feasibility program for (IGLC) with three constraints:

(i) for $v \in V$, $\sum_{c \in C_v} x_{vc} \geq 1$, guaranteeing that vertex $v$ is assigned a color from $C_v$.

(ii) for $t \in T$, $c \in C$, $\sum_{I_v \in I(t)} x_{vc} \leq 1$, guaranteeing that at most a single interval in each clique $I(t)$ is assigned a particular color.

(iii) for $v \in V$, $c \in C$, $x_{vc} \in \{0, 1\}$: integrality constraints for the variables.

Clearly, a feasible solution implies a legal coloring for (IGLC).

In [11] it was shown that the IGLC problem is NP-hard, even though coloring interval graphs can be done efficiently. In Section 4.6 we present approximation algorithms that minimize the size of the largest clique with the same color, i.e., minimizing the violation factor of constraint (ii) in the feasibility program. This can also be viewed as bounding the number of copies of each color. We complement this result by showing that the violation achieved almost matches the gap between the feasibility of the integral and linear programs. We consider this gap as the integrality gap.

### 4.2.2 Useful Procedures

Our model generalizes several known problems. For example, [27] showed that the generalized assignment problem (below) and the uncapacitated facility location problem (see Introduction) are special cases of the NFV problem. Throughout this work we use several known procedures that are defined below.

**Cover-Growing Algorithm**

Here we describe a cover-growing algorithm for the uncapacitated facility location (UFL) problem. We present it together with its analysis, since we use it later and take advantage of its local properties. We assume that our input is a fractional solution
to the UFL problem. The output is an integral solution, i.e., a set of open facilities such that each client is assigned to an open facility.

We can view a fractional solution to the facility location problem as inducing a probability distribution over the facilities from which a client gets service. Thus, the fractional connection cost of a client is an expectation, since it is a sum of weighted distances (where the service fractions serve as weights). A cover (or ball) around a client, having radius twice the expected distance, contains at least half of the client’s fractional service. Thus, by doubling the fractions inside the cover, the client gets all of its service from it.

**Rounding Algorithm.**

1. Define a cover around each client with radius twice the expected distance.

2. Until all clients are satisfied:
   
   (a) among all unconnected clients, find client $i$ with minimum radius cover.
   
   (b) open facility $f$ that minimizes the installation cost in the cover.
   
   (c) for every client $i'$ whose cover intersects the cover of client $i$ (there exists a facility that serves both): connect it to facility $f$.

It follows from the rounding algorithm that every client is either connected to a facility in its own cover or connected to a facility in an intersecting cover (which does not have a larger radius). In this case we say that a client $j$ is connected to a facility in its representative cover. Thus, the connection costs are at most 6 times the sum of the expected distances, and the installation costs are at most twice the fractional installation costs. In total, the approximation factor achieved is 6 for the uncapacitated facility location.

**Generalized Assignment Problem**

In the generalized assignment problem (GAP) we are given $m$ machines and $n$ jobs that need to be assigned to the machines. Job $j$ has cost $c_{ij}$ and size $w_{ij}$ on machine $i$; machine $i$ has total size $w_i$. Our goal is to assign each job to a machine, without
violating machine size constraints, while minimizing the total assignment cost. Assume we are given a feasible fractional solution to GAP. In our algorithms we apply a rounding procedure to the given fractional solution due to Shmoys and Tardos (see [88]). The output of the rounding procedure is an integral solution whose cost is at most the cost of the fractional solution, and the size of every machine is violated by at most the maximum size of a job assigned fractionally to the machine, i.e., by at most a factor of two. GAP is a special case of the NFV placement problem in which all distances in the metric are set to be zero. In this case at most one copy of each function (i.e., a job) is installed, yielding a GAP instance.

**Interval Selection**

In the linear relaxation (presented in Section 4.3) of the DFL problem we pay for fractional changes in the assignment of a client to a facility between consecutive time steps. Eisenstat et al. [32] gave a procedure that breaks the time horizon into intervals, separately for each client, such that in each interval the fractional connection is static. The fractional change in each interval is bounded. The idea behind the procedure is to iteratively construct the intervals for each client. An interval terminates at the latest time step $t$ in which the fractional changes that were accumulated through until $t$ are bounded. The procedure for client $j$ is as follows:

1. Set $t^j_0 = 1$ and $\ell = 1$.

2. Next interval starts at the maximal $t$, $t \in (t^j_\ell, T + 1]$, such that $\sum_{i \in F} (\min_{t^j_\ell \leq u < t} x^u_{ij}) \geq \theta$ (where $\theta \in [0, 1]$).

3. If $t = T + 1$, all intervals are selected; otherwise, set $\ell \leftarrow \ell + 1$ and select next interval.

For each interval we set the new static fractions $\hat{x}$ as follows. For each $t$ in the $\ell$th interval,

$$\hat{x}^t_{ij} = \frac{\min_{t^j_\ell \leq u < t^j_{\ell+1}} x^u_{ij}}{\sum_{t' \in F} \min_{t^j_\ell \leq u < t^j_{\ell+1}} x^{u^*}_{i't_j}}.$$
It is straightforward to verify that the fractions in the solution are at most multiplied by $\frac{1}{\theta}$, since the numerator is smaller than all fractions in its interval and the denominator is at least $\frac{1}{\theta}$. If we multiply the installation fractions by $\frac{1}{\theta}$, then the solution is feasible. Next, we want to show that fractional changes in each interval are at least $1 - \theta$. If so, since we only change the assignment at the end of each interval, we pay at most $\frac{1}{1-\theta}$ times the change cost. If we consider the $\ell$th interval, the total fractional change in assignment in the interval is

$$\sum_{i \in F} \sum_{t_i' \leq u \leq t_i' + 1} z^u_{ij} \geq \sum_{i \in F} (x_{ij}^t - \min_{t_i' \leq u \leq t_i' + 1} x_{ij}^u)$$

$$= 1 - \sum_{i \in F} \min_{t_i' \leq u \leq t_i' + 1} x_{ij}^u \geq 1 - \theta.$$

The first inequality follows since the change is at least the first fractional connection minus the min fractional connection.

4.3 Dynamic Facility Location

We consider here the dynamic facility location problem (i.e., single commodity) under the assumption that facilities/servers are static (as in the NFV setting). We obtain a 7-approximation algorithm for this problem, improving over the $O(\log n T)$ approximation algorithm of Eisenstat et al. [32]. This improvement forms the basis for obtaining constant factor approximations when extending the NFV setting to the dynamic setting. Our algorithm uses the interval selection procedure of Eisenstat et al. [32], however, our approach significantly departs from [32], thus enabling us to improve on their results. The linear program is\(^1\):

$$\begin{align*}
\min & \sum_{i \in F} f_i y_i + \sum_{t \in [T]} \sum_{j \in C} d_t(i, j) x_{ij}^t + \sum_{t \in [T]} \sum_{j \in C} z_{ij}^t \cdot g \\
\text{subject to:}
\end{align*}$$

\(^1\)The variables are the same as in the linear program for the dynamic NFV problem in Section 4.1.
\[
x_{ij}^t \leq y_i \quad \forall i \in F, j \in C, t \in [T]
\]
\[
\sum_{i \in F} x_{ij}^t = 1 \quad \forall j \in C, t \in [T]
\]
\[
x_{ij}^t - x_{ij}^{t+1} \leq z_{ij}^t \quad \forall i \in F, j \in C, t \in [T]
\]
\[
y_i, x_{ij}^t, z_{ij}^t \geq 0 \quad \forall i \in F, j \in C, t \in [T]
\]

We first solve the LP for the problem. Given a fractional solution, we run the interval selection procedure and then apply the cover-growing algorithm. The details are as follows.

**Algorithm 1: DFL-Algorithm**

**Input:** Instance of Dynamic Facility Location.

1. Solve the LP and construct the intervals for each client (see Section 4.2.2).
2. For each client and interval, find the average distance from the client to each facility (over the time steps of the interval), and define a cover for the client containing only facilities that, on average, are within distance of at most twice the fractional connection cost.
3. Run the cover-growing algorithm over the covers from the previous step as in Section 4.2.2.

**Lemma 4.3.1.** Algorithm 1 provides a 7-approximation factor for the DFL problem.

**Proof.** As mentioned in Section 4.2.2, the interval selection procedure can at most double the cost of the solution, yet it allows us to change the assignment (only) between consecutive intervals, while paying at most the fractional change cost. Thus, the total change cost is twice the fractional change cost. The next claim is rather easy to prove:

**Claim 4.3.2.** For every interval of a client, if two facilities are in the cover of the interval, i.e., the average distance to each of them is smaller than twice the fractional connection cost, 2r, then the distance between them is at most 4r.

In Step (3) we run the cover-growing algorithm over the constructed covers. Each client in each interval is either assigned to a facility in its own cover, or in its representative cover. Recall that all facilities in its own cover are at distance of at most twice
its fractional connection cost, thus summing up over the interval, the total connection
cost is at most twice the fractional connection cost times the number of time steps in
the interval. If a client is assigned to a facility in its representative cover, we pay an
additional connection cost, from the intersection of the covers to the opened facility.
As seen in the above claim, the distance between each pair of facilities inside a cover is
at most four times the fractional connection cost over the interval it is defined for, so
this additional cost is at most four times the fractional connection cost which defines
the cover. Since the representative cover has a smaller fractional connection cost, the
overall distance from the client to the opened facility in its representative cover is at
most six times its own fractional connection cost. In addition, since we open at most
a single facility in each cover, from Markov inequality the total installation cost is at
most twice the fractional installation cost. Concluding, after accounting for the $\frac{1}{\theta}$ fac-
tor of the interval selection procedure (see Section 4.2.2 for the role of $\theta$), we get an
approximation ratio of $\frac{6}{\theta}$ for the connection costs, $\frac{2}{\theta}$ for the installation costs, and $\frac{1}{1-\theta}$
for the change costs. By choosing $\theta = \frac{6}{7}$ we get an approximation ratio of 7.

4.4 Dynamic Uncapacitated NFV Placement

In this section we consider the Dynamic Uncapacitated NFV Placement (Dyn-UNFV)
problem, an uncapacitated variant of Dyn-CNFW and a generalization of DFL (see
Section 4.1). In Dyn-UNFV, for each time step we need to solve an instance of the
uncapacitated NFV problem, while paying for changes in the assignment of clients to
facilities. We solve this problem by extending the Algorithm 1. The algorithm first
constructs covers for each commodity separately, but then it uses a rounding algorithm
for GAP (instead of the cover-growing algorithm) to decide on the final location of the
commodities. The next algorithm together with Lemma 4.4.1 proves Theorem 1.1.1.
Algorithm 2: Dynamic UNFV Algorithm

Input: Instance of Dyn-UNFV.

1. Solve DNФV-LP (with infinite capacities) and construct the intervals for each client (see Section 4.2.2).
2. For each client and interval, calculate the average distance from the client to each facility (over the time steps of the interval), and define a cover for the client containing only facilities that, on average, are within distance of at most twice the fractional connection cost. The radius of a cover is defined as twice its client’s fractional connection cost.
3. Separately, for each commodity, pick a non-intersecting set of representative covers as follows: select a cover with minimum radius and delete all covers that intersect with it. Continue until there are no more covers. Each client whose cover was deleted receives service from the representative cover that ”caused” the deletion.
4. Apply the GAP rounding algorithm to the fractional solution defined by the representative covers (see Section 4.2.2): each cover is considered a separate job and each facility a machine.
5. Assign each client at each interval to the facility in its representative cover in which the commodities were installed according to the GAP rounding.

Lemma 4.4.1. Algorithm 2 provides \((O(1), O(1))\) bi-criteria approximation factor for the Dyn-UNFV problem.

Proof. We first notice that the analysis of the connection and change costs remains the same as for the Algorithm 1 (Section 4.3), assuming the same choice of \(\theta = \frac{6}{7}\). This means that the connection costs of the rounded solution are at most 7 times the fractional connection costs, and the change costs are at most 7 times the fractional change costs. For each commodity \(s\), the representative covers do not intersect. This allows us to install a copy of \(s\) in each representative cover. It follows that we can treat each cover, a fractional allocation of a commodity to a facility, as a fractional assignment of a job to a set of machines. Each job has a cost equal to its installation cost and a size equal to the commodity’s size, and each machine has a size bound equal
to the facilities size bound. Thus, the fractional allocation of representative covers to facilities define an instance of GAP. The rounding algorithm for GAP returns an integral solution whose cost is at most the fractional cost and the size constraints are violated by at most a factor of 2. Thus, since we multiplied the installation fractions by $\frac{2}{\theta} = \frac{7}{3}$ (in the interval selection and in the covers construction), the integral solution returned from it has an installation cost of at most $\frac{7}{3}$ times the fractional installation costs, and the size constraints are violated by at most a factor of $\frac{14}{3}$. To conclude, we get a bi-criteria approximation. The approximation ratio of the cost is 7 for the connection costs, $\frac{7}{3}$ for the installation costs and 7 for the change costs, and the size constraints are violated by at most a factor of $\frac{14}{3}$.

4.5 Dynamic Capacitated NFV Placement

In this section we consider the Dynamic Capacitated NFV Placement (Dyn-CNFV) problem. In this problem (see Section 4.1) we need to solve an instance of the capacitated NFV placement problem at each time step, while taking into account the cost of changing the assignment of clients to facilities between time steps. Just like the previous algorithm (for Dyn-UNFV) we select the intervals and find their covers for each client. The difficulty we encounter here is that we may connect too many covers to the same representative cover, resulting in a violation of the commodities’ capacities. We choose the representative covers similarly to Cohen et al. [27], and then reduce the problem to an instance of the interval graph list coloring problem. The next algorithm together with Lemma 4.5.1 prove Theorem 1.1.2.
Algorithm 3: Dyn-CNFV Algorithm

**Input:** Instance of Dyn-CNFV.

1. Solve DNFV-LP and select intervals.
2. For each client and interval, calculate the average distance from the client to each facility (over the time steps of the interval), and define a cover for the client containing only facilities that, on average, are within distance of at most twice the fractional connection cost. The radius of a cover is defined as twice its client’s fractional connection cost.
3. Select a cover with smallest radius and assign all intersecting covers to it. Remove covers that were already assigned at least $\frac{1}{4}$ of their service. Continue till all covers are removed.
4. For each cover, normalize to 1 the total service it gets from the representative covers.
5. Apply the interval graph list coloring rounding algorithm (see Section 4.6) to the problem defined by the covers and representative cover: each cover is considered an interval and each representative cover defines a set of colors (of size equal to its capacity).
6. Apply the GAP rounding algorithm to the fractional solution defined by the representative covers (see Section 4.2.2): each cover is considered a separate job and each facility a machine.
7. Assign each client, at each interval, to the facility in its representative cover in which the commodities were installed according to the GAP rounding.

**Lemma 4.5.1.** Algorithm 3 provides $(O(1), O(1)), O(\log \min\{n, T\}))$ tri-criteria approximation factor for the Dynamic Capacitated NFV problem.

**Proof.** Like previous algorithms, we first select the intervals, paying a factor of $\frac{1}{1-\theta}$ over the change cost, and $\frac{1}{\theta}$ over all other terms. Next we find a cover for each interval and pay another factor of 2 for doubling its radius (for all terms, but the change cost). Afterwards, we assign each cover to a set of representative covers of smaller radius, supplying at least $\frac{1}{4}$ of its service. By normalizing the total service of each cover we multiply all installation fractions by at most 4, increasing the installation cost and size.
constraint violation.

We are left with two tasks. First, assigning each cover to a single representative cover, and second, choosing in which facility to install each cover, which, like the previous (uncapacitated) algorithm, we solve by running a GAP rounding algorithm. We solve the first task by reducing the problem it defines to the IGLC problem. For each representative cover with a total capacity of \( \mu \), we create \( \mu \) different colors. Each cover is defined for a time interval \([t_1, t_2]\) and is represented by a set of representative covers. Thus, for each cover we create an interval \( I = [t_1, t_2] \), associated with the subset of colors created for its representative covers. The fractional service assignment of covers to representative covers can be seen as a fractional coloring of intervals, in which a single copy of each color suffices. This defines an instance of IGLC. From the algorithm for IGLC we get a coloring of the intervals. If the interval defined for cover \( j \) is colored by color \( c \), and \( c \) is the color created for representative cover \( i \), we set cover \( i \) as the final representative of cover \( j \).

What are the properties of the fractional coloring? From the construction of the (IGLC) instance, together with the fact that each interval is fully colored, we can infer that there is a feasible solution for the linear program for (IGLC). This means that every subset of \( \ell \) intersecting intervals has at least \( \ell \) colors available for it, which is exactly the local condition our rounding algorithms requires. Thus, the rounding algorithms for (IGLC) return a solution in which the number of copies of each color is at most \( \log(\min\{n, T\}) \), leading to a violation of the capacity constraints of the commodities by the same factor. After taking into account the additional violation from the GAP rounding we get that the size constraints are violated by a factor of \( \frac{16}{17} \). By choosing \( \theta = \frac{16}{17} \) we get that the total cost is at most 17 times the fractional cost, and that the size constraints are violated by a factor of 17. Overall we get an approximation of \((17, 17, O(\log(\min\{n, T\})))\): 17-approximation for the costs and size constraints, and \(O(\log(\min\{n, T\}))-approximation for capacities. \(\square\)
4.6 Interval Graph List Coloring

We provide two approximation algorithms for the (IGLC) problem. Given a legal fractional solution for the feasibility program (see Section 4.2.1), the first algorithm (see Section 4.6.1) finds a solution in which the maximum number of intersecting intervals that get the same color is of size $O(\log k)$, where $k$ is the size of the largest clique. The second algorithm (see Section 4.6.2) finds a solution in which the maximum number of intersecting intervals that get the same color is of size $O(\log T)$. We emphasize that $T$ and $k$ are independent parameters. Combining the two algorithms proves Theorem 1.1.3.

We note that the fractional feasibility of the (IGLC) instance guarantees the following: the size of the union of the allowed color sets of any set $S$ of intersecting intervals is at least as large as $|S|$. Thus, the conditions of Hall’s theorem are satisfied in the bipartite graph of intersecting intervals vs. colors. This property turns out to be very useful in our algorithms.

We complement the above results and show that the integrality gap of the linear feasibility program is $\Omega(\frac{\log k}{\log \log k})$. This means that given a feasible fractional solution, any integral solution might have to color a clique of size at least $\Omega(\frac{\log k}{\log \log k})$ with the same color.

4.6.1 Clique Size Dependent Approximation

We can achieve an $O(\log k)$ approximation if we find a coloring of at least half the intervals in each clique (with a constant number of copies of each color). This is achieved by utilizing Hall’s theorem. Then, by repeating iteratively, we end up with a full coloring of the intervals.

**Claim 4.6.1.** There exists a subset of intervals $S$ that can be colored with color $c$, such that in each clique $I(t)$ having an interval that can be colored by color $c$, $1 \leq |I(t) \cap S| \leq 2$.

**Proof.** Let us assume the claim is false and there is no such subset $S$. Define $S = \{I_v | c \in C_v\}$. Obviously, each clique with an interval that can be colored by color $c$ has
an interval in this subset $S$. There is a point $t$ with at least three intervals (otherwise, we are done). Let $I_1 = [a_1, b_1], I_2[a_2, b_2]$ and $I_3 = [a_3, b_3]$ be three intervals in $S$ that intersect at $t$. First we notice that all starting points $a_i$ are smaller than $t$, and all ending points $b_i$ are bigger than $t$. Without loss of generality let us assume $a_1$ is the smallest starting point. Now there are two options, either $b_1$ is bigger than $b_2$ and $b_3$, which in this case, obviously, $I_1$ contains both of $I_2$ and $I_3$. The other option is that either $b_2$ or $b_3$ has the biggest ending point. W.l.o.g let us assume that $a_2 \geq a_3$. So the range $[a_1, b_2]$ contains $I_3$. In either case we notice that we can remove at least one of the intervals without uncovering points on the line. We can continue with this process until there are no points with more than two intervals in $S$.

Using this claim we can iterate through the colors, and color intervals. In each iteration we can guarantee that in each clique $I(t)$ containing an interval that can be colored by color $c$ (the color of the current iteration), some interval $I \in I(t)$ is colored.

**Clique-Coloring Algorithm.**

1. For each color $c$:
   
   (a) Create a subset of intervals that can be colored $c$

   (b) Until there are no cliques of size three or more: select three intervals in such a clique and remove the interval contained in the other two intervals from the subset.

   (c) Color all intervals in the subset with color $c$.

2. If there are uncolored intervals, go back to (1).

**Lemma 4.6.2.** *Clique-Coloring Algorithm provides $O(\log k)$ approximation for IGLC.*

*Proof.* Each clique can be seen as a matching of intervals to colors. We are guaranteed by the fractional solution (to IGLC in Section 4.2.1) that the number of colors available for any set of $\ell$ intersecting intervals is at least $\ell$. Therefore, by Hall’s theorem, there is a perfect matching in each clique from intervals to colors.

Whenever we color a subset of intervals by color $c$, we color at least one interval in every clique with an interval that can be colored with $c$. Consider a clique $I(t)$ of
size $s$ with an interval that received color $c$. As already mentioned, $I(t)$ has a perfect matching before coloring the intervals. After we color a subset of intervals by color $c$, at least one of the intervals in $I(t)$ was colored by $c$, and no other interval in $I(t)$ will receive color $c$ in the same iteration. It may be the case that coloring the interval by color $c$ does not agree with the coloring in the perfect matching of $I(t)$. So, by coloring it we may have lost a potential match, and are left with only $s - 2$ possible matches. In the worst case, for every interval colored, each clique has two less color matches available, one for the colored interval, and one for the interval that was matched to color $c$ in the perfect matching. In addition, any clique without an interval that can be colored by $c$ can still be matched perfectly since the bipartite graph of intervals and colors does not change.

Since we use a single color each time, if an interval can be matched to color $c$ in one clique, it can be matched to it in all cliques (that contain it). Overall we lose a single match in each clique for an interval we color, so we are left without colors only after half of the intervals in each clique were colored. Since this applies to all cliques, after $\log k$ iterations, where $k$ is the size of the largest clique, all cliques are fully colored. Each iteration uses only two copies of each color, thus the result coloring uses at most $O(\log k)$ copies of each color.

\[ \square \]

### 4.6.2 Horizon Length Dependent Approximation

Assuming there is a feasible fractional solution to the (IGLC) program, we split the time horizon into two independent parts which we color separately, achieving an $O(\log T)$ approximation. Similarly to the previous algorithm, we exploit the fact that each clique can be legally colored.

**Time-Split Algorithm.**

1. Let $t$ be the middle point in $T$.

2. Color the intervals in clique $I(t)$.

57
3. Continue recursively on both halves of $T$. At the $i$th iteration use the $i$th copy of the set of colors.

**Lemma 4.6.3.** *Time-Split Algorithm provides $O(\log T)$ approximation for the IGLC problem.*

**Proof.** Each clique can be seen as a matching of intervals to colors. We are guaranteed by the fractional solution (to IGLC in Section 4.2.1) that the number of colors available for any set of $\ell$ intersecting intervals is at least $\ell$. Therefore, by Hall's theorem, there is a perfect matching in each clique from intervals to colors. Next, at each step we color the middle point with a new set of colors. After removing all colored intervals we are left with two ranges that do not share any intervals thus, we can use the next copy of the set of colors at both of them. Each time the we color a clique we create two ranges that are at most half the size of the original range, so it is obvious we will not need more than $O(\log T)$ copies of the colors to color all intervals. \(\square\)

### 4.6.3 Integrality Gap

We define an instance of (IGLC) where we are given 4 colors denoted by 1, 2, 3, 4. There are six intervals, $I_1, \ldots, I_6$, as seen in Figure 4.1. The intervals' subsets of allowed colors are: $C_1 = \{1, 2\}$, $C_2 = \{3, 4\}$, $C_3 = \{1, 3\}$, $C_4 = \{2, 3\}$, $C_5 = \{1, 4\}$, and $C_6 = \{2, 4\}$.

We define a feasible fractional solution in which each interval is colored half-half by its allowed colors. For any integral coloring of $I_1$ and $I_2$ there is no legal coloring for one of the other intervals (intervals $I_3, I_4, I_5$ and $I_6$, cover all coloring combination of intervals $I_1$ and $I_2$). This means we will need at least two copies of some color, and the integrality gap is 2.

This example can be expanded along the same lines. First, we define a set of $x$ intervals, $B_1$, each colored equally with its own distinct set of $x$ colors. Next we define
$B_2$, a set of $x^x$ disjoint intervals. Each interval in $B_2$ requires a different color set, one from each of the intervals in $B_1$, and is colored equally by them. As seen in the example above, for every coloring of the intervals in $B_1$, one of the intervals in $B_2$ will be left without a color it can be legally colored with. This results with a clique of two intervals that receive the same color.

Next, we create a copy of the intervals in $B_1$, each with its own set of distinct colors, and add them to $B_1$. For each interval $I \in B_2$ we create $x^x$ disjoint intervals in its range. Their colors are chosen in the same way as the colors of the intervals in $B_2$, but with the colors of the new intervals of $B_1$. The new intervals are added to $B_2$ as well. As a result of this process, in any integral coloring, a point $t$ exists such that $I(t)$ contains at least two different pairs of intervals that received the same color.

Applying this process $x$ times results with an instance for which:

(i) in any integral solution, there exists a clique with $x$ different pairs of intervals that received the same color.

(ii) in every clique we used at most $\frac{2}{x}$ of every color, $\frac{1}{x}$ for every level.

An interval that requires the $x$ colors described in (i), that are already used by $x$ pairs of intervals, forces us to use a third copy of some color. We can create the third level $B_3$ appropriately to guarantee that some interval receives the third color. We can create at most $x$ levels in this manner, as each level requires at most $\frac{1}{x}$ of every color. If the size of the largest clique is $k = O(x^x)$, the integrality gap is $\Omega(\frac{\log k}{\log \log k})$.

### 4.7 Experiments

We devote this section to test the uncapacitated dynamic NFV algorithm. To this end we consider a subnet of the physical network of Cogent, a tier 1 ISP, which offers us a realistic facilities’ deployment (using its publicly available data center locations). We choose ten data centers placed in Europe and defined facilities at their location. Next, we added a hundred clients in random positions, and for each client defined a random walk, taking a step at a random direction of random size. In addition, each client was
assigned a random commodity vector describing which commodities, out of a list of five different commodities, it requires. Finally, a random size was assigned to each facility, together with a size and cost for each commodity (at each facility).

In order to evaluate the algorithm, we compare it with previous solutions for the problem, the uncapacitated (static) NFV from [27]. Since the static algorithm does not optimize over the time horizon of the dynamic problem, we define two different variants for using it. The first one uses the clients’ position over the time horizon to find the average position of each client. Using this average position we obtain a single time step instance. The second one runs the static algorithm at each time step separately, paying change costs accordingly. These two options define the two extreme options of a fully static solution, in which we do not allow any changes, and a fully dynamic one, which does not integrate between solutions to avoid overpaying for change costs and opening costs.

In Figure 4.2 shows the cost percentage of the two versions of the static algorithm compared to the cost of the dynamic algorithm. It can be seen that the intuition for the performance of the algorithm is correct, that is, for small change costs (relative to the connection cost), ignoring it results in higher costs, and for high change costs, it does not necessarily come with a cost. Still, giving consideration to the dynamic nature of the problem does give advantage to the dynamic algorithm, and for high change costs,
Figure 4.3: Performance ratio of the dynamic UNFV algorithm compared to the static UNFV and the fractional solution with respect to theta.

the performance of the static algorithm and the dynamic algorithm, converge to one another.

In Section 4.2.2 we discussed the interval selection procedure. For each client, we found the time steps so that in between substantial fractional changes in the assignment were accumulated, and we split the time horizon into intervals at these time steps. The question of at which point the "right amount" of fractional change has accumulated arises. We ran the experiments with a small change cost (0.04 of the average connection cost) and a large change cost (which equals the average connection cost). As seen in Figure 4.3, this value, denoted by $\theta$, may have significant impact on the performance of the algorithm. If we choose a value too big, we may induce too many assignment changes as we break the time horizon into too many intervals. And for small values of $\theta$, we may end up with a static solution which may lose the advantage of the dynamic algorithm. Another evidence for this can be found in the comparison with the fractional solution. For values of $\theta$ in the range $[0.4, 0.6]$, the ratio between the fractional solution and the algorithm’s solution is bigger. Usually the optimal solution and the fractional solution are not close. This leads us to assume that in practice, the ratio between the algorithm’s solution and the optimal solution is better than 2.

Lastly, in Figure 4.4 we can see the performance ratio of the dynamic algorithm as a function of the expected number of commodities that can be installed in each facility. The performance of the algorithm peaks as the size constraint loosens. The
Figure 4.4: Performance ratio of the dynamic UNFV algorithm compared to the static UNFV with respect to the facility size.

lower performance ratio may be the result of tight size constraint which creates a hard problem without room for much improvement. On the other hand, when the sizes of the facilities are very large, we may install each commodity at several facilities to allow more assignment changes.
Chapter 5

The Submodular Multiple Knapsack Problem

In this section we study the Submodular Multiple Knapsack Problem (SMKP), a natural generalization of the classic Multiple Knapsack Problem. First, we present the tools and techniques used by our algorithm. Next, by an overview of the approximation algorithm. It is followed by the presentation and analysis of the structuring and rounding steps used by the algorithm.

We use $\mathcal{I} = (I, w, B, W, f)$ to denote an SMKP instance consisting of a set of items $I$ with weights $w_i$ for $i \in I$, a set of bins $B$ with capacities $W_b$ for $b \in B$, and objective function $f$. Given a set $A \subseteq I$, let $w(A) = \sum_{i \in A} w_i$. We denote by $OPT(\mathcal{I})$ the optimal solution value for the instance $\mathcal{I}$.

5.1 Tools and Techniques

Our algorithm relies on a refined analysis of techniques for submodular optimization subject to $d$-dimensional knapsack constraints [18,22,61], combined with sophisticated application of tools used in the development of approximation schemes for packing problems [30].

At the heart of our algorithm lies the observation that SMKP for a large number
of identical bins (i.e., \( \forall b \in B, \ W_b = W \) for some \( W \geq 0 \)) can be easily approximated via a reduction to the problem of maximizing a submodular function subject to a 2-dimensional knapsack constraint (see, e.g., [61]). Given such an SMKP instance and \( \varepsilon > 0 \), we partition the items to \textit{small} and \textit{large}, where an item \( i \in I \) is small if \( w_i \leq \varepsilon W \) and large otherwise. We further define a \textit{configuration} to be a subset of large items which fits into a single bin, and let \( C \) be the set of all configurations. It follows that for fixed \( \varepsilon > 0 \), the number of configurations is polynomial.

Using the above we define a new submodular optimization problem, to which we refer as the \textit{block-constraint} problem. We define a new universe \( E \) which consists of all configurations \( C \) and all small items, \( E = C \cup \{|i| \text{ is small}\} \). We also define a new submodular function \( g : 2^E \to \mathbb{R}_{\geq 0} \) by \( g(T) = f(\bigcup_{A \in T} A) \). Now, we seek a subset of elements \( T \subseteq E \) such that \( T \) has at most \( m = |B| \) configurations, i.e., \( |T \cap C| \leq m \), and the total weight of sets selected is at most \( m \cdot W \); namely, \( \sum_{A \in T} w(A) \leq m \cdot W \), where \( w(A) = \sum_{i \in A} w_i \).

It is easy to see that the optimal value of the block-constraint problem is at least the value of the optimum for the original instance. Moreover, a solution \( T \) for the block-constraint problem can be used to generate a solution for the SMKP instance with only a small loss in value. As there are no more than \( m \) configurations, and all other items are small, the items in \( T \) can be easily packed into \((1 + \varepsilon)m + 1\) bins of capacity \( W \) using First Fit. Then, it is possible to remove \( \varepsilon m + 1 \) of the bins while maintaining at least

\[
\frac{m}{(1+\varepsilon)m+1} \geq \frac{1}{1+2\varepsilon}
\]

of the solution value, for \( m \geq \frac{1}{\varepsilon} \). Once these \( \varepsilon m + 1 \) bins are removed, we have a feasible solution for the SMKP instance. The block-constraint problem can be viewed as monotone submodular optimization subject to a 2-dimensional knapsack constraint. Thus, a \((1 - e^{-1} - \varepsilon)\)-approximate solution can be found efficiently [61].

Our approximation algorithm for SMKP is based on a generalization of the above. We refer to a set of bins of identical capacity as a \textit{block}, and show how to reduce an SMKP instance into a submodular optimization problem with a \( d \)-dimensional knapsack constraint, in which \( d \) is twice the number of blocks plus a constant. While, generally, this problem cannot be solved for non-constant \( d \), we use a refined analysis of known algorithms [22,61] to show that the problem can be efficiently solved if the blocks admit a certain structure, to which we refer as \textit{leveled}.

65
We utilize a grouping technique, inspired by the work of Fernandez de la Vega and Lueker [30], to convert a general SMKP instance to a leveled instance. We sort the bins in decreasing order by capacity and then partition them into levels, where level \( t, t \geq 0 \), has \( N^{2+t} \) bins, divided into \( N^2 \) consecutive blocks, each containing \( N^t \) bins. We decrease the capacity of each bin to the smallest capacity of a bin in the same block. While the decrease in capacity generates the leveled structure required for our algorithm to work, it only slightly decreases the optimal solution value. The main idea is that given an optimal solution, each block of decreased capacity can now be used to store the items assigned to the subsequent block on the same level. Also, the items assigned to \( N \) blocks from each level can be evicted, while only causing a reduction of \( \frac{1}{N} \) to the profit (as only \( N \) of the \( N^2 \) blocks of the level are evicted). These evicted blocks are then used for the items assigned to the first block in the next level.

### 5.2 The Approximation Algorithm

In this section we present our approximation algorithm for SMKP. Given an instance \( \mathcal{I} \) of the problem, let \( A^* = \bigcup_{b \in B} A^*_b \) be an optimal solution of value \( \text{OPT}(\mathcal{I}) \). We first observe that there exists a constant size subset \( A = \bigcup_{b \in B} A_b \), where \( A_b \subseteq A^*_b \), satisfying the following property: the value gained from any item in \( i \in A^* \setminus A \) is small relative to \( \text{OPT}(\mathcal{I}) \). Thus, our algorithm initially enumerates over all possible partial assignments of constant size. Each assignment is then extended to an approximate solution for \( \mathcal{I} \). Among all possible partial assignments and the respective extensions the algorithm returns the best solution. Thus, from now on we restrict our attention to finding a solution for the residual problem, obtained by fixing the initial partial assignment.

Formally, given an SMKP instance, \( \mathcal{I} = (I, w, B, W, f) \), a feasible partial solution \((A_b)_{b \in B}\) and \( \xi \in \mathbb{N} \), we define the residual instance \( \mathcal{I}' = (I', w, B, W', f') \) with respect to \((A_b)_{b \in B}\) and \( \xi \) as follows. Let \( A = \bigcup_{b \in B} A_b \) and set \( I' = \left\{ i \in I \setminus A \big| f_A(\{i\}) \leq \frac{f(A)}{\xi} \right\} \). The weights of the items remain the same and so is the set of bins. For every \( b \in B \) we set \( W'_b = W_b - w(A_b) \). Finally, the objective function of the residual instance is \( f' = f_A \).
Lemma 5.2.1. Let $I$ be an SMKP instance, $\xi \in \mathbb{N}$, and $(A_b^*)_{b \in B}$ an optimal solution for $I$ such that $A_{b_1}^* \cap A_{b_2}^* = \emptyset$ for any $b_1, b_2 \in B$, $b_1 \neq b_2$. If $\sum_{b \in B} |A_b^*| \geq \xi$ there is a feasible solution $(A_b)_{b \in B}$ for $I$ such that $A_b \subseteq A_b^*$ for any $b \in B$, $\sum_{b \in B} |A_b| = \xi$, and $(A_b^* \setminus A_b)_{b \in B}$ is a feasible solution for the residual instance of $I$ w.r.t. $(A_b)_{b \in B}$ and $\xi$.

Proof. Let $(A_b^*)_{b \in B}$ be an optimal solution for the SMKP instance. Define $A^* = \bigcup_{b \in B} A_b^*$ and order the items of $A^*$ by their marginal values. That is, $A^* = \{a_1, \ldots, a_r\}$ where $f_{T_{\ell-1}}(\{a_\ell\}) = \max_{a \in A^{\star}\setminus T_{\ell-1}} f_{T_{\ell-1}}(\{a\})$ with $T_\ell = \{a_1, \ldots, a_\ell\}$ for every $1 \leq \ell \leq r$ (also, $T_0 = \emptyset$). Define $(A_b)_{b \in B}$ by $A_b = A_b^* \cap \{a_1, \ldots, a_\xi\}$ for every $b \in B$ and $A = \bigcup_{b \in B} A_b$. We therefore have $A = \{a_1, \ldots, a_\xi\}$.

For any $b \in B$, it holds that $w(A_b) \leq w(A_b^*) \leq W_b$, and thus $(A_b)_{b \in B}$ is a feasible solution for $I$. Furthermore, for any $b \in B$ it holds that $A_b \subseteq A_b^*$ by definition. As the sets $(A_b^*)_{b \in B}$ are disjoint it follows that $\sum_{b \in B} |A_b| = \xi$.

Let $I' = (I', w, B, W', f')$ be the residual instance of $I$ w.r.t. $(A_b)_{b \in B}$ and $\xi$. It remains to show that $(A_b^* \setminus A_b)_{b \in B}$ is a feasible solution for $I'$. For every $\xi < i \leq r$ and $1 \leq \ell \leq \xi$ it holds that $f_A(\{a_i\}) \leq f_{T_{\ell-1}}(\{a_i\}) \leq f_{T_{\ell-1}}(\{a_\ell\})$ where the first inequality follows from the submodularity of $f$ and the second by the definition of $a_\ell$. Combining the last inequality with $f' = f_A$ we obtain,

$$\xi \cdot f'(\{a_i\}) = \xi \cdot f_A(\{a_i\}) \leq \sum_{\ell=1}^{\xi} f_{T_{\ell-1}}(\{a_\ell\}) = f(A) - f(\emptyset) \leq f(A).$$

Thus, $a_i \in I'$, implying that $A_b^* \setminus A_b \subseteq I'$ for any $b \in B$. Furthermore, for any $b \in B$,

$$w(A_b^* \setminus A_b) = w(A_b^*) - w(A_b) \leq W_b - w(A_b) = W'_b.$$ 

It follows that $(A_b^* \setminus A_b)_{b \in B}$ is a solution for the residual instance. \qed \qed

Next, we observe that instances of SMKP are easier to solve when the number of distinct bin capacities is small (e.g., uniform bin capacities), leading us to consider bin blocks:

Definition 5.2.1. For a given instance of SMKP we say that a subset of bins $\bar{B} \subseteq B$
is a block if all the bins in $\tilde{B}$ have the same capacity, i.e., for bins $b_1$ and $b_2$ belonging to the same block it holds that $W_{b_1} = W_{b_2}$.

Following an enumeration over partial assignments, our algorithm reduces the number of blocks by altering the bin capacities. To this end, we use a specific structure that we call leveled, defined as follows.

**Definition 5.2.2.** For any $N \in \mathbb{N}$, we say that a partition $(B_j)_{j=0}^{k}$ of a set $B$ of bins with capacities $(W_b)_{b \in B}$ is $N$-leveled if $B_j$ is a block, and $|B_j| = N \lfloor \frac{j}{N^2} \rfloor$ for all $0 \leq j \leq k$.

By the above definition, we can view each set of consecutive blocks of the same size as a level. For $0 \leq j \leq k$, block $j$ belongs to level $\ell = \lfloor \frac{j}{N^2} \rfloor$. Thus, for level $\ell > 0$ the number of bins in each block of level $\ell$ is $N$ times the number of bins in each block of level $\ell - 1$.

In Section 5.2.1 we give Algorithm 5 which generates an $N$-leveled partition of the bins, $\tilde{B} = \cup_{j=0}^{k} \tilde{B}_j$ with the capacities of the bins $(W_b)_{b \in B}$ modified to $(\tilde{W}_b)_{b \in \tilde{B}}$. We show that solving the problem with these new bin capacities may cause only a small harm to the optimal solution value. In particular, we prove (in Section 5.2.1) the following.

**Lemma 5.2.2.** Algorithm 5 is a polynomial time algorithm which given $N \in \mathbb{N}$, a set of bins $B$ and capacities $(W_b)_{b \in B}$, returns a subset of bins $\tilde{B} \subseteq B$, capacities $(\tilde{W}_b)_{b \in \tilde{B}}$, and an $N$-leveled partition $(\tilde{B}_j)_{j=0}^{k}$ of $\tilde{B}$, such that

1. The bin capacities satisfy $\tilde{W}_b \leq W_b$, for every $b \in \tilde{B}$.

2. For any set of items $I$, weights $(w_i)_{i \in I}$, a submodular non-negative function $f : 2^I \rightarrow \mathbb{R}_{\geq 0}$, and a feasible assignment $(S_b)_{b \in B}$ for the instance $(I, w, B, W, f)$, there exists a feasible assignment $(\tilde{S}_b)_{b \in \tilde{B}}$ for the instance $(I, w, \tilde{B}, \tilde{W}, f)$ such that $f \left( \bigcup_{b \in \tilde{B}} \tilde{S}_b \right) \geq (1 - \frac{1}{N}) f \left( \bigcup_{b \in B} S_b \right)$ and $\bigcup_{b \in \tilde{B}} \tilde{S}_b \subseteq \bigcup_{b \in B} S_b$.

We refer to $\tilde{B}$ and $\tilde{W}$ as the $N$-leveled constraint of $B$ and $W$.

Once the instance is $N$-leveled, we proceed to solve the problem (fractionally) and apply randomized rounding to obtain an integral solution (see Section 5.2.2). Algorithm 7 utilizes efficiently the leveled structure of the instance. Instead of having a
separate constraint for each bin in a block — to bound the total size of the items packed in this bin — we use only two constraints for each block. The first constraint is a knapsack constraint referring to the total capacity of a block, and the second constraint restricts the number of configurations assigned to the block.\footnote{We defined a configuration in Section 5.1.} Thus, the number of constraints significantly decreases if the blocks are large. Since leveled instances also have a constant number of blocks consisting of a single bin, those are handled separately via the notion of $\delta$-restricted SMKP.

Given $\delta > 0$, the input for $\delta$-restricted SMKP includes the same parameters as an input for SMKP, and also a subset $B^r \subseteq B$ of restricted bins. A solution for $\delta$-restricted SMKP is a feasible assignment $(A_b)_{b \in B}$ satisfying also the property that $\forall b \in B^r$ the items assigned to $b$ are relatively small; namely, for any $b \in B^r$ and $i \in A_b$ it holds that $w_i \leq \delta W_b$.

Given the $N$-leveled instance of our problem, we turn the blocks of a single bin (that is, blocks $\tilde{B}_j$ such that $|\tilde{B}_j| = 1$) to be restricted. We note that while items of weight greater than $\delta W_b$ may be assigned to these blocks in some optimal solution, the overall number of such items is bounded by a constant. Indeed, our initial enumeration guarantees that evicting these items from an optimal solution may cause only small harm to the optimal solution value, allowing us to consider the instance as $\delta$-restricted.

In Section 5.2.2 we show the following bound on the performance guarantee of Algorithm 7, which uses randomized rounding. The algorithm is parameterized by $\mu \in (0, 0.1)$ (to be determined). Suppose we are given a $\delta$-restricted SMKP instance $I$, such that the unrestricted bins are partitioned into blocks, i.e., $B \setminus B^r = B_1 \cup \ldots \cup B_k$, and

$$v = \max_{i \in I} f(\{i\}) - f(\emptyset).$$

**Lemma 5.2.3.** For $\mu \in (0, 0.1)$, Algorithm 7 returns a feasible solution $(S_b)_{b \in B}$ such that...
\[ \mathbb{E}[f(\cup_{b \in B} S_b)] \geq (1 - e^{-1})(1 - \mu)^{\frac{1}{1+\mu}}(1 - \gamma)OPT(\mathcal{I}), \text{ where} \]

\[ \gamma = \exp \left( -\frac{\mu^3}{16} \cdot \frac{OPT(\mathcal{I})}{v} \right) + |B'| \exp \left( -\frac{\mu^2}{12} \cdot \frac{1}{\delta} \right) + 2 \cdot \sum_{j=1}^{k} \exp \left( -\frac{\mu^2}{12} |B_j| \right). \]

Algorithm 4 gives the pseudocode of our approximation algorithm for general SMKP instances. The algorithm uses several configuration parameters that will be set in the proof of Lemma 5.2.4.

**Algorithm 4: Algorithm for SMKP**

**Input**: An SMKP instance \( \mathcal{I} = (I, w, B, W, f) \) and the parameters \( N, \xi, \delta \) and \( \mu \).

1. **for all** feasible assignments \( A = (A_b)_{b \in B} \) such that \( \sum_{b \in B} |A_b| \leq \xi \) do
2. Let \( \mathcal{I}' = (I', w, B, W', f') \) be the residual instance of \( \mathcal{I} \) w.r.t \( (A_b)_{b \in B} \) and \( \xi \).
3. Run Algorithm 5 with the bins \( B \) and capacities \( (W'_b)_{b \in B} \). Let \( \tilde{B} \) and \( (\tilde{W}_b)_{b \in \tilde{B}} \) be the output, and \( \tilde{B} = \bigcup_{j=0}^{k} \tilde{B}_j \) the partition of \( \tilde{B} \) to leveled blocks. Let \( \tilde{\mathcal{I}} = (I', w, \tilde{B}, \tilde{W}, f') \) be the resulting instance.
4. Let \( \tilde{\mathcal{I}}_R \) be the \( \delta \)-restricted SMKP instance of \( \tilde{\mathcal{I}} \) with the restricted bins \( \tilde{B}' = \bigcup_{j=0}^{\min\{N^2-1,k\}} \tilde{B}_j \).
5. Solve \( \tilde{\mathcal{I}}_R \) using Algorithm 7 with parameter \( \mu \), and the partition \( \tilde{B} \setminus \tilde{B}' = \bigcup_{j=N^2}^{k} \tilde{B}_j \). Denote the returned assignment by \( (\tilde{S}_b)_{b \in \tilde{B}} \), and let \( S_b = \tilde{S}_b \) for \( b \in \tilde{B} \) and \( S_b = \emptyset \) for \( b \in B \setminus \tilde{B} \).
6. If \( f(\cup_{b \in B} (A_b \cup S_b)) \) is higher than the value of the current best solution, set \( (A_b \cup S_b)_{b \in B} \) as the current best solution.
7. Return the best solution found.

**Lemma 5.2.4.** For any \( \varepsilon > 0 \), there are parameters \( N, \xi, \delta, \mu \) such that, for any SMKP instance \( \mathcal{I} \), Algorithm 4 returns a solution of expected value at least \((1 - e^{-1} - \varepsilon)OPT(\mathcal{I})\). \]

**Proof.** We start by setting the parameter values. The reason for selecting these values will become clear later. Given a fixed \( \varepsilon \in (0,0.1) \), there is \( \mu \in (0,0.1) \) such that
\[
\frac{(1-\mu)^3}{1+\mu} \geq (1 - \varepsilon^2). \quad \text{By the Monotone Convergence Theorem,}
\]
\[
\lim_{N \to \infty} 2N^2 \cdot \sum_{t=1}^{\infty} \exp \left( -\frac{\mu^2 \cdot N^t}{12} \right) = \sum_{t=1}^{\infty} \lim_{N \to \infty} 2N^2 \exp \left( -\frac{\mu^2 \cdot N^t}{12} \right) = 0.
\]

It follows that there are \( N > \frac{1}{\varepsilon^2} \) and \( \delta > 0 \) such that
\[
N^2 \exp \left( -\frac{\mu^2}{12} \cdot \frac{1}{\delta} \right) + 2N^2 \cdot \sum_{t=1}^{\infty} \exp \left( -\frac{\mu^2}{12} \cdot N^t \right) \leq \frac{\varepsilon^2}{2}.
\]  

Finally, we select \( \xi \) such that \( \xi \geq \frac{N^2}{\varepsilon^2} \) and \( \exp \left( -\frac{\mu^2}{16} \cdot \frac{\xi}{\delta} \right) \leq \frac{\varepsilon^2}{2} \).

Let \( \mathcal{I} = (I, w, B, W, f) \) be an SMKP instance, and let \( (A^*_b)_{b \in B} \) be an optimal solution for \( \mathcal{I} \). Assume w.l.o.g. that \( A^*_b \cap A^*_b' = \emptyset \) for any \( b_1, b_2 \in B, b_1 \neq b_2 \). Define \( A^* = \bigcup_{b \in B} A^*_b \). If \( |A^*| \leq \xi \), there is an iteration of Line 1 in which \( A^*_b = A_b \) for all \( b \in B \). Therefore, in this iteration we have at Line 6 \( f(\bigcup_{b \in B}(A_b \cup S_b)) \geq f(A^*) \), and the algorithm returns a solution of value at least \( f(A^*) \). Otherwise, by Lemma 5.2.1, there is a feasible solution \( (A_b)_{b \in B} \) such that \( A_b \subseteq A^*_b \), \( \sum_{b \in B} |A^*_b| = \xi \) and \( (A^*_b \setminus A_b)_{b \in B} \) is a feasible solution for \( \mathcal{I}' \), the residual instance of \( \mathcal{I} \) w.r.t. \( (A_b)_{b \in B} \) and \( \xi \). It follows that there is an iteration of Line 1 which considers this solution \( (A_b)_{b \in B} \). We focus on this iteration for the rest of the analysis.

Let \( A = \bigcup_{b \in B} A_b \). If \( f(A) \geq (1 - e^{-1}) f(A^*) \) then when the algorithm reaches Line 6 it holds that \( f(\bigcup_{b \in B}(A_b \cup S_b)) \geq f(A) \geq (1 - e^{-1}) f(A^*) \); therefore, the algorithm returns a \( (1 - e^{-1}) \)-approximation in this case. Henceforth, we can assume that \( f(A) \leq (1 - e^{-1}) f(A^*) \). Then,
\[
f'(\bigcup_{b \in B}(A^*_b \setminus A_b)) = f'(A^*_b \setminus A) = f(A^*) - f(A) \geq \frac{f(A)}{1 - e^{-1}} - f(A) = \frac{1}{e - 1} f(A).
\]

Since \( (A^*_b \setminus A_b)_{b \in B} \) is a feasible solution for \( \mathcal{I}' \), and by Lemma 5.2.2, it holds that
\[
OPT(\tilde{\mathcal{I}}) \geq \left( 1 - \frac{1}{N} \right) f'(A^*_b \setminus A) \geq (1 - \varepsilon^2) f'(A^*_b \setminus A),
\]  

71
where $\tilde{I}$ is the instance defined in Step 3. The last inequality follows from the definition of $N$. Let $(D_b)_{b \in \tilde{B}}$ be an optimal solution for $\tilde{I}$. Consider $(D'_b)_{b \in \tilde{B}}$ where $D'_b = D_b \setminus \{i \in D_b | w_i > \delta \cdot \tilde{W}_b\}$ for $b \in \tilde{B}^c$ (the set $\tilde{B}^c$ is defined in Line 4) and $D'_b = D_b$ for $b \in \tilde{B} \setminus \tilde{B}^c$. It follows that $D'_b$ is a solution for the $\delta$-restricted SMKP instance $\tilde{I}_R$. As for any $b \in \tilde{B}^c$ it holds that $|\{i \in D_b | w_i > \delta \cdot \tilde{W}_b\}| \leq \frac{1}{7}$, we have

$$OPT(\tilde{I}_R) \geq f'(\cup_{b \in \tilde{B}} D'_b) \geq OPT(\tilde{I}) - \frac{N^2}{\delta \cdot \xi} f(A) \geq (1 - \varepsilon^2) f'(A^* \setminus A) - \varepsilon^2 \cdot f(A).$$

(5.4)

The second inequality follows from the definition of residual instance, and the third inequality from (5.3) and the choice of $\xi$. Since $f'(A^* \setminus A) \geq \frac{1-\varepsilon}{\varepsilon} f(A)$ and $\varepsilon \in (0, 0.1)$, it follows that $OPT(\tilde{I}_R) \geq \frac{f(A)}{5}$.

Let $v = \max_{i \in I'} f'(\{i\})$. By Lemma 5.2.3, we have that

$$\mathbb{E} \left[ f'(\cup_{b \in \tilde{B}^c} \tilde{S}_b) \right] \geq (1 - e^{-1}) \left( \frac{1-\mu}{1+\mu} \right)^3 (1-\gamma) OPT(\tilde{I}_R) \geq (1- e^{-1})(1-\varepsilon^2)(1-\gamma) OPT(\tilde{I}_R),$$

(5.5)

where

$$\gamma = \exp \left( -\frac{\mu^3}{16} \cdot \frac{OPT(\tilde{I}_R)}{v} \right) + |\tilde{B}^c| \exp \left( -\frac{\mu^2}{12} \cdot \frac{1}{5} \right) + 2 \cdot \sum_{j=N^2}^{k} \exp \left( -\frac{\mu^2}{12} |\tilde{B}_j| \right) \leq \exp \left( -\frac{\mu^3}{16} \cdot \frac{\xi}{5} \right) + N^2 \exp \left( -\frac{\mu^2}{12} \cdot \frac{1}{5} \right) + 2 \cdot N^2 \cdot \sum_{i=1}^{\infty} \exp \left( -\frac{\mu^2}{12} N^i \right) \leq \varepsilon^2. \quad (5.6)$$

The first inequality uses $v = \max_{i \in I'} f'(\{i\}) \leq \xi^{-1} f(A)$ (by the definition of $I'$). The second inequality holds since $OPT(\tilde{I}_R) \geq \frac{f(A)}{5}, |\tilde{B}^c| \leq N^2$ and there are at most $N^2$ blocks $\tilde{B}_j$ of size $N^i$. The last inequality uses (5.2) and the choice of $\xi$. Combining (5.6) with (5.5) and (5.4), we obtain

$$\mathbb{E} \left[ f(\cup_{b \in \tilde{B}} (A_b \cup S_b)) \right] \geq f(A) + \mathbb{E} \left[ f'(\cup_{b \in \tilde{B}} \tilde{S}_b) \right] \geq f(A) + (1 - e^{-1})(1 - \varepsilon^2)^2 OPT(\tilde{I}_R) \geq f(A) + (1 - e^{-1})(1 - \varepsilon^2)^3 f'(A^* \setminus A) - \varepsilon^2 f(A) \geq (1 - e^{-1} - \varepsilon) f(A^*).$$

Hence, in this iteration the solution considered in Line 6 has expected value at least
(1 − e^{-1} − \varepsilon) f(A^*). This completes the proof of the lemma.

\[ \square \]

\[ \square \]

**Lemma 5.2.5.** For any constant parameters \( N, \xi, \delta \) and \( \mu \), Algorithm 4 returns a feasible solution for the input instance in polynomial time.

**Proof.** We first note that for any fixed parameter values the algorithm has a polynomial running time. The number of assignments considered in Line 1 can be trivially bounded by \( (n \cdot m)\xi \). As Algorithms 5 and 7 are polynomial in their input size, the operations in each iteration are also done in polynomial time.

For each iteration of Line 1, by Lemma 5.2.3, \((\tilde{S}_b)_{b \in \tilde{B}}\) is a feasible solution to \( \tilde{I}_R \). Therefore, for any \( b \in B \) either \( w(S_b) = w(\emptyset) \leq W'_b \) or \( w(S_b) = w(\tilde{S}_b) \leq \tilde{W}_b \leq W'_b \), where the last equality follows from Lemma 5.2.2. Therefore, \( w(A_b \cup S_b) \leq w(A_b) + W'_b \leq \tilde{W}_b \). Hence, the solution considered in each iteration is feasible for the input instance.

\[ \square \]

Theorem 1.2.1 follows from Lemmas 5.2.4 and 5.2.5.

### 5.2.1 Structuring the Instance

In this section we present Algorithm 5 and prove Lemma 5.2.2. Our technique for generating an \( N \)-leveled partition can be viewed as a variant of the linear grouping technique of [30]. We start with a brief overview of the classical concepts of grouping and shifting in the context of a multiple knapsack constraint.

Let \( B = \{1, 2, \ldots, m\} \) be a set of bins with capacities \((W_b)_{b \in B}\), where \( W_1 \geq W_2 \geq \ldots \geq W_m \) and \( m = q \cdot N^2 \) for some integer \( q \geq 1 \). We can partition \( B \) into \( N^2 \) groups (sets) \( B_1, \ldots, B_{N^2} \), each consists of \( q \) consecutive bins, i.e., \( B_j = \{(j-1) \cdot q+1, \ldots, j \cdot q\} \) for \( 1 \leq j \leq N^2 \). Thus, the capacity of a bin in \( B_j \) is greater or equal to the capacity of a bin in \( B_{j+1} \).

We use the partition to define new capacities for the bins. The new capacity of a bin \( b \in B_j \) is \( \tilde{W}_b = \min_{b' \in B_j} W_{b'} = W_{q \cdot j} \), the minimal (original) capacity of a bin in its group. Clearly, given an SMKP instance \( I = (I, w, B, W, f) \) and a feasible assignment \((S_b)_{b \in B}\) for the instance, it may be that \((S_b)_{b \in B}\) is infeasible for the instance with the new capacities \( \tilde{I} = (I, w, B, \tilde{W}, f) \).
Figure 5.1: Input and output example for Algorithm 5 with $N = 2$. The original capacities, $W$, are represented by empty rectangles, whereas the hatched rectangles represent the new capacities $\tilde{W}$. Note that the last three bins are discarded by the algorithm as they do not form a full block.

We can apply shifting to partially circumvent this hurdle. Given $b \in B_{j+1}$, $j \neq N^2$, the set $S_b$ complies with the new capacity constraint of any bin $b' \in B_j$, i.e., $w(S_b) \leq W_b \leq \tilde{W}_{b'}$. Define a new assignment $(\tilde{S}_b)_{b \in B}$ by $\tilde{S}_b = S_{b+q}$ for $b \in B \setminus B_{N^2}$ and $\tilde{S}_b = \emptyset$ for $b \in B_{N^2}$. As $b+q \in B_{j+1}$ for any $b \in B_j$, $j \neq N^2$, it follows that $(\tilde{S}_b)_{b \in B}$ is a feasible assignment for $\tilde{I}$. Furthermore, $(\tilde{S}_b)_{b \in B}$ is an assignment of all the items in $(S_b)_{b \in B}$, except for the items $\bigcup_{b \in B_1} S_b$ assigned to the first group in $(S_b)_{b \in B}$. The assignment of these items is handled by different techniques.

Algorithm 5 applies grouping with non-uniform group size to generate the $N$-leveled partition. The algorithm assumes w.l.o.g that the set of bins is $B = \{1, 2, \ldots, m\}$ and that the bins are ordered by capacity, $W_1 \geq W_2 \geq \ldots \geq W_m$. It defines groups (or blocks) of bins, where group $j$ consists of $N^\lfloor j/N^2 \rfloor$ consecutive bins, for $j \geq 0$. The capacity of the bins in each group is reduced to the minimal capacity of a bin in this group. This procedure is formalized in Algorithm 5. A simple illustration for a small instance is given in Figure 5.1.
Algorithm 5: Structure in Blocks

Input: A set of bins $B$, capacities $(W_b)_{b \in B}$ and $N \in \mathbb{N}$.

1. Let $B = \{1, \ldots, m\}$ where $W_1 \geq W_2 \geq \ldots \geq W_m$.
2. Let $k = \max \{\ell \in \mathbb{N} \mid \sum_{r=0}^{\ell} N^{\left\lfloor \frac{r}{N} \right\rfloor} \leq m\}$.
3. Define $\tilde{B}_j = \left\{ b \mid \sum_{r=0}^{j-1} N^{\left\lfloor \frac{r}{N} \right\rfloor} < b \leq \sum_{r=0}^{j} N^{\left\lfloor \frac{r}{N} \right\rfloor} \right\}$ for $0 \leq j \leq k$.
4. Let $\tilde{B} = \bigcup_{j=0}^{k} \tilde{B}_j$, and $\tilde{W}_b = \min_{b' \in \tilde{B}_j} W_{b'}$ for all $0 \leq j \leq k$ and $b \in \tilde{B}_j$.
5. Return $\tilde{B}$, $(\tilde{W}_b)_{b \in \tilde{B}}$ and the partition $(\tilde{B}_j)_{j=0}^{k}$.

By construction, we have that $(\tilde{B}_j)_{j=0}^{k}$ is an $N$-leveled partition of $\tilde{B}$. Furthermore, $\tilde{B} \subseteq B$ and $\tilde{W}_b \leq W_b$ for any $b \in \tilde{B}$. Finally, it can be easily observed that Algorithm 5 has a polynomial running time. Thus, to complete the proof of Lemma 5.2.2 we need to show that property 2 holds as well. To this end, we use a variant of the shifting argument outlined in the above overview.

Lemma 5.2.6. Let $N \in \mathbb{N}$, $B$ be a set of bins with capacities $(W_b)_{b \in B}$ and let $\tilde{B}$, $\tilde{W}$ be the output of Algorithm 5 for the input $B$, $W$ and $N$. Furthermore, let $I$ be a set of items with weights $(w_i)_{i \in I}$, $f : 2^I \to \mathbb{R}_{\geq 0}$ be a submodular non-negative function, and $(S_b)_{b \in B}$ be a feasible assignment for $(I, w, B, W, f)$. Then there is $(\tilde{S}_b)_{b \in \tilde{B}}$ feasible for $(I, w, \tilde{B}, \tilde{W}, f)$ such that $f\left(\bigcup_{b \in \tilde{B}} \tilde{S}_b\right) \geq (1 - \frac{1}{N}) f\left(\bigcup_{b \in B} S_b\right)$ and $\bigcup_{b \in \tilde{B}} \tilde{S}_b \subseteq \bigcup_{b \in B} S_b$.

Clearly, Lemma 5.2.6 completes the proof of Lemma 5.2.2. To prove Lemma 5.2.6 we use the property of submodular functions stated in Lemma 3.1.6.

Proof of Lemma 5.2.6. W.l.o.g assume that $B = \{1, 2, \ldots, m\}$ and $W_1 \geq W_2 \geq \ldots \geq W_m$. Furthermore, assume the sets $(S_b)_{b \in B}$ are disjoint. We modify $(S_b)_{b \in B}$ using a sequence of steps, eventually obtaining a feasible assignment $(\tilde{S}_b)_{b \in \tilde{B}}$ for $(I, w, \tilde{B}, \tilde{W}, f)$. Figure 5.2 gives an illustration of these steps.

Define $\tilde{B}_{k+1} = B \setminus \tilde{B}$. We note that $\tilde{B}_{k+1}$ may be empty. We partition $\{\tilde{B}_j\}_{0 \leq j \leq k+1}$ into levels and super-blocks. We consider each $N^2$ consecutive blocks to be a level, and each $N$ consecutive blocks within a level to be a super-block. Formally, level $t$ is

$$\mathcal{L}_t = \{j \mid t \cdot N^2 \leq j < \min\{(t+1)N^2, k+2\}\}$$
Figure 5.2: Illustration of the steps in the proof of Lemma 5.2.6 for $N = 3$. Each row represents a level and each box represents either a block or a super-block. The number in the box is the number of bins in the block and a gray background implies that the block is non-empty.
for $0 \leq t \leq \ell$ with $\ell = \left\lfloor \frac{k + 1}{N^2} \right\rfloor$. We note that if $\ell = 0$, then $(S_b)_{b \in B}$ is a feasible assignment for the instance $(I, w, \tilde{B}, \tilde{W}, f)$ and the claim hold. Thus, we may assume that $\ell \geq 1$. The super-block $r$ of level $t$ is

$$S_{t,r} = \{ j \mid t \cdot N^2 + r \cdot N \leq j < t \cdot N^2 + (r + 1) \cdot N \}$$

for $0 \leq r < N$ and level $0 \leq t < \ell$ (we do not partition the last level into super-blocks). It follows that $B = \bigcup_{t=0}^{\ell} \bigcup_{j \in L_t} \tilde{B}_j$ and $L_t = \bigcup_{r=0}^{N-1} S_{t,r}$ for $0 \leq t < \ell$. Furthermore, for any $j \in L_t$, $j \neq k + 1$ it holds that $|\tilde{B}_j| = N^t$ and $|\tilde{B}_{k+1}| < N^t$. Essentially, all the blocks of level $t$ are of the same size, and the number of bins in a super-block in level $t - 1$ is the number of bins in a single block of level $t$. We use this property in the shifting process where the assignments of items to bins in the first blocks of level $t$ are shifted to the last super-block of level $t - 1$. The eviction and shuffle steps, described below, are used to ensure that all bins in the last super-block of each level are empty when shifting is applied.

We modify the assignment $(S_b)_{b \in B}$ using the following steps.

**Eviction:** We first evict a super-block of bins from each level (except the last one). Let $R = \bigcup_{j \in L_t} \bigcup_{b \in B_j} S_b$ be the subset of items assigned to the last level, and let $g : 2^I \to \mathbb{R}_{\geq 0}$ defined by $g(Q) = f(Q \cup R)$. Note that $g$ is submodular and non-negative (see Claim 3.1.1). Also, let $V_{t,r} = \bigcup_{j \in S_{t,r}} \bigcup_{b \in \tilde{B}_j} S_b$ be the set of items assigned to super-block $r$ of level $t$ for $0 \leq r < N$ and $0 \leq t < \ell$. Then, by definition, we have

$$f \left( \bigcup_{b \in B} S_b \right) = g \left( \bigcup_{r=0}^{N-1} \bigcup_{t=0}^{\ell-1} V_{t,r} \right).$$

Furthermore, we note that the sets $\left( \bigcup_{t=0}^{\ell-1} V_{t,r} \right)_{r=0}^{N-1}$ are disjoint. Hence, by Lemma 3.1.6 there is $0 \leq r^* < N$ such that

$$g \left( \bigcup_{0 \leq r < N, r \neq r^*} \bigcup_{t=0}^{\ell-1} V_{t,r} \right) \geq \left( 1 - \frac{1}{N} \right) \cdot g \left( \bigcup_{r=0}^{N-1} \bigcup_{t=0}^{\ell-1} V_{t,r} \right) = \left( 1 - \frac{1}{N} \right) f \left( \bigcup_{b \in B} S_b \right).$$

77
We define a new assignment \((T_b)_{b \in B}\) by

\[
T_b = \begin{cases} 
\emptyset & b \in B_j \text{ for } j \in S_{t,r^*} \text{ and } 0 \leq t < \ell \\
S_b & \text{otherwise}
\end{cases}
\]

for any \(b \in B\). Thus,

\[
f\left( \bigcup_{b \in B} T_b \right) = g \left( \bigcup_{0 \leq r < N-1, \ r \neq r^*} \bigcup_{j \in S_{t,r}} \bigcup_{i=0}^{\ell-1} V_{t,r} \right) \geq \left( 1 - \frac{1}{N} \right) f \left( \bigcup_{b \in B} S_b \right). \quad (5.7)
\]

It also holds that \(T\) is a feasible assignment for the instance \((I, w, B, W, f)\), since \(T_b \in \{S_b, \emptyset\}\) for any \(b \in B\). By the same argument, it follows that \(S_b \subseteq \bigcup_{b \in B} T_b \subseteq \bigcup_{b \in B} S_b\).

**Shuffling:** We now generate a new assignment \(\tilde{T}\) such that \(S_b \in \bigcup_{b \in B} \tilde{T}_b = \bigcup_{b \in B} T_b\), and the last super-block in each level (except the last one) is empty. This property is obtained by moving the assignments of the bins in super-block \(N-1\) to the bins of super-block \(r^*\) for every \(0 \leq t < \ell\).

In case \(r^* = N-1\) we simply define \((\tilde{T}_b)_{b \in B}\) by \(\tilde{T}_b = T_b\) for any \(b \in B\). Otherwise we have \(r^* \neq N-1\). For any \(0 \leq t < \ell\), let \(\varphi_t : \bigcup_{j \in S_{t,r^*}} \tilde{B}_j \to \bigcup_{j \in S_{t,N-1}} \tilde{B}_j\) be a bijection from the bins of super-block \(r^*\) to the bins of the last super-block of level \(t\) (note that both sets have the same cardinality, thus such a bijection exists). We define \((\tilde{T}_b)_{b \in B}\) by

\[
\tilde{T}_b = \begin{cases} 
\emptyset & b \in \tilde{B}_j \text{ for } j \in S_{t,N-1}, 0 \leq t < \ell \\
T_{\varphi_t(b)} & b \in \tilde{B}_j \text{ for } j \in S_{t,r^*}, 0 \leq t < \ell \\
T_b & \text{otherwise}
\end{cases}
\]

for any \(b \in B\). For \(b \in \tilde{B}_j\) with \(j \in S_{t,r^*}, 0 \leq t < \ell\) it holds that

\[
w(\tilde{T}_b) = w(T_{\varphi_t(b)}) \leq W_{\varphi_t(b)} \leq W_b
\]

where the last inequality follows from \(\varphi_t(b) > b\). Also, for any other bin \(b \in B\) it holds that \(\tilde{T}_b \in \{\emptyset, T_b\}\) thus \(w(\tilde{T}_b) \leq W_b\).

In both cases it holds that \(\bigcup_{b \in B} \tilde{T}_b = \bigcup_{b \in B} T_b\) and \(\tilde{T}\) is a feasible assignment for
\((I, w, B, W, f)\).

**Shifting:** In this step we generate a feasible assignment \((\tilde{S}_b)_{b \in \tilde{B}}\) for the instance \((I, w, \tilde{B}, \tilde{W}, f)\). As the bins of the last super-block in each level (except the last level) are vacant in \((\tilde{T})_{b \in \tilde{B}}\), we use them for the assignment of the first block of the next level. This can be done since \(N\) blocks of level \(t\) contain the same number of bins as a single block of level \(t+1\). We also use blocks in levels greater than 0 which are not in the last super-block to store the assignment of the subsequent block in the same level.

Formally, define \((\tilde{S}_b)_{b \in \tilde{B}}\) by

\[
\tilde{S}_b = \begin{cases} 
\tilde{T}_{b+N+1} & b \in \tilde{B}_j \text{ with } j \in S_{t,N-1}, 0 \leq t < \ell \\
\tilde{T}_{b+N} & b \in \tilde{B}_j \text{ with } j \in L_t \setminus S_{t,N-1}, 0 < t < \ell \\
\tilde{T}_{b+N} & b \in \tilde{B}_j \text{ with } j \in L_{\ell}, b + N^{\ell} \leq m \\
\emptyset & b \in \tilde{B}_j \text{ with } j \in L_{\ell}, b + N^{\ell} > m \\
\tilde{T}_b & b \leq N^2 - N 
\end{cases}
\]

for any \(b \in \tilde{B}\). The first case defines the shifting of the assignments of the first block of level \(t+1\) to the last super-block of level \(t\). The second and third cases define the shifting of assignments of a block to the previous block. The forth case handles the last block in \(\tilde{B}\), and the last case indicates that the assignments of the first \(N^2 - N\) bins remain in place.

Let \(b \in \tilde{B}\). If \(\tilde{S}_b = \emptyset\) then \(w(\tilde{S}_b) = 0 \leq \tilde{W}_b\). If \(b \leq N^2 - N\) then \(\{b\} = \tilde{B}_{b-1}\). Hence, \(w(\tilde{S}_b) = w(\tilde{T}_b) \leq W_b = \tilde{W}_b\). In any other case, there are \(0 < j < j' \leq k+1\) and \(b' \in \tilde{B}_{j'}\) such that \(b \in \tilde{B}_j\) and \(\tilde{S}_b = \tilde{T}_{b'}\). It follows that \(W_{b'} \leq \tilde{W}_b\) by the definition of \(\tilde{W}_b\). Thus, \(w(\tilde{S}_b) = w(\tilde{T}_{b'}) \leq W_{b'} \leq \tilde{W}_b\). That is, \((\tilde{S}_b)_{b \in \tilde{B}}\) is feasible for \((I, w, \tilde{B}, \tilde{W}, f)\).

Clearly, \(\bigcup_{b \in \tilde{B}} \tilde{S}_b \subseteq \bigcup_{b \in \tilde{B}} \tilde{T}_b\). Let \(i \in \bigcup_{b \in \tilde{B}} \tilde{T}_b\); thus, there is \(b \in B\) such that \(i \in T_b\). There is \(0 \leq j \leq k+1\) such that \(b \in \tilde{B}_j\). Also, since \(T_b \neq \emptyset\) it holds that \(b \not\in S_{t,N-1}\) for all \(0 \leq t \leq \ell\). If \(j \in L_0\) then \(\tilde{S}_b = \tilde{T}_b\) as \(b \not\in S_{0,N-1}\), hence \(i \in \bigcup_{b \in \tilde{B}} \tilde{S}_b\). Otherwise, either \(j \in L_t \setminus S_{t,N-1}\) for \(1 \leq t < \ell\), or \(j \in L_{\ell}\) for \(t = \ell\), it can be verified that in both cases \(\tilde{S}_{b-N} = \tilde{T}_b\); thus, \(i \in \bigcup_{b \in \tilde{B}} \tilde{S}_b\). Therefore, it also holds that

79
\[ \bigcup_{b \in \hat{B}} \hat{S}_b = \bigcup_{b \in B} \tilde{T}_b = \bigcup_{b \in B} T_b \subseteq \bigcup_{b \in B} S_b, \text{ and by (5.7)} \]

\[ f \left( \bigcup_{b \in \hat{B}} \hat{S}_b \right) = f \left( \bigcup_{b \in B} \tilde{T}_b \right) \geq \left( 1 - \frac{1}{N} \right) \cdot f \left( \bigcup_{b \in B} S_b \right). \]

\[ \square \]

5.2.2 Solving a Continuous Relaxation and Rounding

In this section we give Algorithm 7 which outputs a solution satisfying Lemma 5.2.3. The input for the algorithm is a \( \delta \)-restricted SMKP instance along with a partition \( B \setminus B^r = B_1 \cup \ldots \cup B_k \) of the bins, where \( B_j \) is a block for all \( 1 \leq j \leq k \). The algorithm utilizes the block-constraint instance defined in Section 5.2.2. We give the algorithm in Section 6.

The Block-Constraint Instance

Recall that a \( \delta \)-restricted SMKP instance is defined by an SMKP instance \( I = (I, w, B, W, f) \) and a set of restricted bins \( B^r \subseteq B \). Given such an instance, a partition \( B_1, \ldots, B_k \) of \( B \setminus B^r \) to blocks and \( \mu > 0 \), we define their associated block constraint instance as a triplet \((E, P, g)\), where \( E \) is a set, \( P \subseteq [0, 1]^E \) is a polytope and \( g : 2^E \to \mathbb{R}_{\geq 0} \) is a monotone non-negative submodular function. The instance \((E, P, g)\) defines the optimization problem \( \max_{T \subseteq E} : \bar{x}_T \in P \ g(T) \); however, this point of view is only used indirectly.\(^2\) In the following we give the formal definition of the block constraint instance \((E, P, g)\).

For simplicity, let \( \{B_{k+1}, \ldots, B_\ell\} = \{\{b\} \mid b \in B^r\} \) be a set of blocks, each consisting of a single bin. Thus, \( B = \bigcup_{j=1}^{\ell} B_j \). Denote the (uniform) capacity of the bins in block \( B_j \) by \( W^*_j \), for \( 1 \leq j \leq \ell \). That is, for any \( b \in B_j \) it holds that \( W^*_j = W_b \). For \( 1 \leq j \leq k \), we say that an item \( i \in I \) is \( j \)-small if \( w_i \leq \mu \cdot W^*_j \); otherwise \( i \) is \( j \)-large. Let \( I_j = \{\{i\} \mid i \text{ is } j \text{-small}\} \) for \( 1 \leq j \leq k \). For \( k < j \leq \ell \) define \( I_j = \{\{i\} \mid w_i \leq \delta \cdot W^*_j \} \).

\(^2\)For a set \( T \subseteq E \), we use \( \bar{x}^T \) to denote the vector \( \bar{x}^T \in \{0, 1\}^E \) defined by \( \bar{x}^T_e = 1 \) for \( e \in T \), and \( \bar{x}^T_e = 0 \) for \( e \in E \setminus T \).
A \textit{j-configuration} is a subset of \textit{j}-large items which can be packed into a single bin in \(B_j\). That is, \(C \subseteq I\) is a \textit{j}-configuration if every item \(i \in C\) is \textit{j}-large and \(w(C) \leq W_j^*\). Let \(C_j\) be the set of all \textit{j}-configurations for \(1 \leq j \leq k\) and \(C_j = \emptyset\) for \(k < j \leq \ell\). As any \textit{j}-configuration has at most \(\mu - 1\) items, it follows that \(|C_j| \leq |I|^{\mu - 1}\), i.e., the number of configurations is polynomial in the size of \(I\). Furthermore, for \(A \subseteq I\) such that \(w(A) \leq W_j^*\), \(1 \leq j \leq k\), there are \(C \in C_j\) and \(S \subseteq I\) such that all the items in \(S\) are \textit{j}-small and \(A = C \cup S\). Our algorithm exploits this property.

The set \(E\) is defined by \(E = \{(S, j) | S \in C_j \cup I_j, 1 \leq j \leq \ell\}\). Informally, the element \((S, j) \in E\) represents an assignment of all the items in \(S\) to a single bin \(b \in B_j\). The function \(g : 2^E \to \mathbb{R}_{\geq 0}\) is defined by \(g(T) = f\left(\bigcup_{(S, j) \in T} S\right)\). By Claim 3.1.4, \(g\) is a submodular, monotone and non-negative function.

We define the polytope \(P\) as follows.

\[
P = \left\{ \bar{x} \in [0,1]^E \mid \sum_{C \in C_j} \bar{x}(C, j) \leq |B_j| \quad \forall 1 \leq j \leq k, \sum_{S \in C_j \cup I_j} w(S) \cdot \bar{x}(S, j) \leq |B_j| \cdot W_j^* \quad \forall 1 \leq j \leq \ell \right\} \tag{5.8}
\]

The polytope represents a relaxed version of the capacity constraints over the bins. For each block \(B_j, 1 \leq j \leq k\), we only require that the total weight of items assigned to bins in \(B_j\) does not exceed the total capacity of the bins in this block. We also require that the number of \textit{j}-configurations selected for \(B_j\) is no greater than the number of bins in this block.

Our algorithm for solving \(\delta\)-restricted SMKP uses the unified greedy algorithm of [41] to find \(\bar{x} \in P\) such that \(G(\bar{x})\) is of high value, where \(G\) is the multilinear extension of \(g\). Subsequently, a random set \(T\) is sampled based on \(\bar{x}\). The set \(T\) is then converted to a solution for the original instance using Algorithm 6. The approximation guarantee of the above process relies on the following connection between the \(\delta\)-restricted SMKP instance and the block constraint instance.

**Lemma 5.2.7.** Let \(I = (I, w, B, W, f)\) and \(B^*\) be an instance of \(\delta\)-restricted SMKP, \(B_1, \ldots, B_k\) a partition of \(B \setminus B^*\) to blocks and \(\mu > 0\). Furthermore, let \((E, P, g)\) be the block constraint instance of the above. Then the following hold:
1. There is $T \subseteq E$, $\bar{x}^T \in P$ such that $g(T) \geq \text{OPT}(T)$, where $\text{OPT}(T)$ is the optimal solution value for $\delta$-restricted SMKP instance $\mathcal{I}$ and $B^r$.

2. Given $T \subseteq E$ such that $\bar{x}^T \in (1 - \mu) \cdot P$, Algorithm 6 returns in polynomial time a feasible solution $(A_b)_{b \in B}$ for $\delta$-restricted SMKP instance $\mathcal{I}$ and $B^r$ satisfying $f(\cup_{b \in B}A_b) = g(T)$.

**Algorithm 6:** Employ a Block-Constraint Solution for SMKP

**Input:** A $\delta$-restricted SMKP instance $\mathcal{I} = (I, w, B, W, f)$ and $B^r$, a partition $B_1, \ldots, B_k$ of $B \setminus B^r$ to blocks and $T \subseteq E$.

1. Let $\{B_{k+1}, \ldots, B_l\} = \{\{b\} | b \in B^r\}$.
2. Set $A_b = \emptyset$ for every $b \in B$.
3. Sort the elements $(S, j)$ in $T$ in decreasing order by the $w(S)$ values.
4. for each $(S, j) \in T$ in the sorted order do
   5. Set $A_b \leftarrow A_b \cup S$ where $b = \arg\min_{b \in B_j} w(A_b)$.
6. Return $(A_b)_{b \in B}$.

**Proof.** We start by proving part 1. Let $(A^*_b)_{b \in B}$ be an optimal solution for the $\delta$-restricted SMKP instance, and let $L_j$ be the set of all $j$-large items for $1 \leq j \leq k$, and $L_j = \emptyset$ for $k < j \leq \ell$ (recall we use $\{B_{k+1}, \ldots, B_l\} = \{\{b\} | b \in B^r\}$). Define

$$T = \left( \bigcup_{j=1}^k \{(A^*_b \cap L_j, j) | b \in B_j\} \right) \cup \left( \bigcup_{j=1}^\ell \bigcup_{b \in B_j} \{(i, j) | i \in A^*_b \setminus L_j\} \right).$$

It can be easily shown that $g(T) = f(\cup_{b \in B}A^*_b)$. Furthermore, as $(A^*_b)_{b \in B}$ is a feasible solution, it holds that $\bar{x}^T \in P$.

We now prove part 2. Let $(A_b)_{b \in B}$ be the output of Algorithm 6 for the given input. We first note that $\cup_{b \in B}A_b = \cup_{(S, j) \in T}S$, and thus $g(T) = f(\cup_{b \in B}A_b)$.

For any $b \in B^r$, there is $k < j \leq \ell$ such that $B_j = \{b\}$. Therefore $A_b = \{i \mid (i, j) \in T\}$, and since $\bar{x}^T \in (1 - \mu)P$, it follows that $w(A_b) \leq W^*_j = W_b$.

Let $1 \leq j \leq k$ and $b \in B_j$. Assume by negation that $w(A_b) > W_b = W^*_j$. Let $(S, j) \in T$ be the last element in $T$ such that $S \neq \emptyset$ and $S$ was added to $A_b$ in Line 5.

---

3Given a polytope $Q$ and $\eta \geq 0$ we use the notation $\eta \cdot Q = \{\eta \bar{x} | \bar{x} \in Q\}$. 

82
We conclude that $w(A_b \setminus S) > 0$, as otherwise $w(A_b) = w(S) \leq W_b$, by the definition of $E$. Therefore, there are at least $|B_j|$ elements $(S', j) \in T$ such that $w(S') \geq w(S)$ (else, in the iteration of $(S, j)$ there must be $b \in B_j$ with $A_b = \emptyset$). If $S \in C_j$ then $w(S) > \mu \cdot W^*_j$, and thus

$$\{|S' \neq \emptyset| (S', j) \in T, \ S' \in C_j\| \geq \{|S'| (S', j) \in T, \ w(S') \geq w(S)\| > |B_j|,$$

contradicting $\bar{x}^T \in (1 - \mu)P$.

Therefore $S \notin C_j$, and we conclude that $S = \{i\}$ with $w_i \leq \mu \cdot W^*_j$. Thus, $w(A_b \setminus S) > (1 - \mu) \cdot W^*_j$. Here, $S$ was allocated to $A_b$ (which is a set of minimum weight). Then, for any $b' \in B_j$, we have $w(A_{b'}) \geq w(A_b) > (1 - \mu) \cdot W^*_j$. Thus,

$$\sum_{(S', j) \in T} w(S') \geq \sum_{b' \in B_j} w(A_{b'}) > |B_j|(1 - \mu) \cdot W^*_j,$$

contradicting $\bar{x}^T \in (1 - \mu)P$. We conclude that $w(A_b) \leq W_b$.

Also, by definition, we have that for any $b \in B^c$ and $i \in A_b$ it holds that $w_i \leq \delta \cdot W_b$. Hence, $(A_b)_{b \in B}$ is a solution for the restricted SMKP instance.

An Algorithm for $\delta$-restricted SMKP

We are now ready to present our algorithm for $\delta$-restricted SMKP. We note that in Line 3 of Algorithm 7 we use sampling by a solution vector $\bar{x}^*$, as defined in Section 3.1.
Algorithm 7: Solve and Round

Input: A $\delta$-restricted SMKP instance $\mathcal{I}$ and $B'$, a partition $B_1, \ldots, B_k$ of $B \setminus B'$ to blocks, and a parameter $\mu > 0$.

1. Let $(E, P, g)$ be the block-constraint instance of $\mathcal{I}$, $B'$, $(B_j)_{j=1}^k$ and $\mu$.

2. Let $G : [0, 1]^E \rightarrow \mathbb{R}_{\geq 0}$ be the multilinear extension of $g$. Find a solution $\bar{y}^*$ for $\max_{\bar{x} \in P} G(\bar{x})$ using the unified greedy of [41].

3. Let $\bar{x}^* = \frac{1-\mu}{1+\mu} \cdot \bar{y}^*$ and sample a random set $T \sim \bar{x}^*$.

4. if $T \in (1-\mu)P$ then
   5. Use Algorithm 6 to convert $T$ into a solution $(A_b)_{b \in B}$ for $\delta$-restricted SMKP instance $\mathcal{I}$ and $B'$. Return $(A_b)_{b \in B}$.

6. else
   7. Return $(A_b)_{b \in B}$ with $A_b = \emptyset$ for every $b \in B$.

For the analysis, consider first the running time. We note that, for any $\bar{\lambda} \in \mathbb{R}^E$, a vector $\bar{x} \in P$ which maximizes $\bar{x} \cdot \bar{\lambda}$ can be found in polynomial time. Therefore, the continuous greedy in Line 2 runs in polynomial time. Thus, Algorithm 7 has a polynomial running time.

It remains to show that the algorithm returns a solution of expected value as stated in Lemma 5.2.3. Similarly to [21], we use submodular concentration bounds within the proof. We note it is possible to prove a variant of this lemma using an approach of [61]. While eliminating the dependence on $\upsilon$, this will result in a more involved proof (recall that $\upsilon$ is defined in (5.1)).

Proof of Lemma 5.2.3. For any $e \in E$ define $X_e$ to be a random variable such that $X_e = 1$ if $e \in T$ and $X_e = 0$ otherwise. It follows that $(X_e)_{e \in E}$ are independent Bernoulli random variables, $\mathbb{E}[X_e] = \bar{x}^*_e$ and $T = \{e \in E | X_e = 1\}$.

We first consider blocks $k < j \leq \ell$. Let $k < j \leq \ell$ and $B_j = \{b\}$. Since $\bar{x}^* \in \frac{1-\mu}{1+\mu} P$, it follows that $\mathbb{E} \left[ \sum_{(S,j) \in E} w(S) \cdot X_{(S,j)} \right] \leq \frac{1-\mu}{1+\mu} \cdot W_b$. Also, $w_{(S,j)} \leq \delta \cdot W_b$ for every $(S,j) \in E$. Using Chernoff’s bound (Theorem 3.1 in [46], see also Lemma 3.2.1), we
have

\[
\Pr \left( \sum_{(S,j) \in T} w(S) > (1 - \mu)W_b \right) \leq \exp \left( -\frac{\mu^2}{3} \cdot \frac{1 - \mu}{1 + \mu} \cdot \frac{1}{\delta} \right) \leq \exp \left( -\frac{\mu^2}{12} \cdot \frac{1}{\delta} \right),
\]

(5.9)

where the last inequality follows from \( \mu \in (0, 0.1) \).

Now, let \( 1 \leq j \leq k \). For every \((S,j) \in E\) it holds that \( w(S) \leq W^*_{j} \). Also, since \( \bar{x}^* \in \frac{1 - \mu}{1 + \mu}P \), it holds that \( \mathbb{E} \left[ \sum_{(S,j) \in E} w(S) \cdot X_{(S,j)} \right] \leq \frac{1 - \mu}{1 + \mu} \cdot |B_j| \cdot W^*_{j} \), and \( \mathbb{E} \left[ \sum_{(S,j) \in E, \ s \in C_j} 1 \cdot X_{(S,j)} \right] \leq \frac{1 - \mu}{1 + \mu} \cdot |B_j| \). Therefore, by Chernoff’s bound (Theorem 3.1 in [46] and Lemma 3.2.1), we have

\[
\Pr \left( \sum_{(S,j) \in T} w(S) > (1 - \mu)|B_j|W^*_{j} \right) \leq \exp \left( -\frac{\mu^2}{3} \cdot \frac{1 - \mu}{1 + \mu} \cdot |B_j| \right) \leq \exp \left( -\frac{\mu^2}{12} \cdot |B_j| \right),
\]

(5.10)

\[
\Pr \left( \sum_{(S,j) \in T, \ s \in C_j} 1 > (1 - \mu)|B_j| \right) \leq \exp \left( -\frac{\mu^2}{3} \cdot \frac{1 - \mu}{1 + \mu} \cdot |B_j| \right) \leq \exp \left( -\frac{\mu^2}{12} \cdot |B_j| \right).
\]

(5.11)

By Lemma 5.2.7, \( \max_{z \in PC(\{0,1\}^E) G(z) \geq OPT(\mathcal{T}) } \). As the unified greedy of [41] yields a \((1 - e^{-1} - o(1))\)-approximation for the problem of maximizing the multilinear extension subject to a polytope constraint, it follows that \( G(\bar{y}^*) \geq (1 - e^{-1})(1 - \mu)OPT(\mathcal{T}) \) (under the assumption that the number of items is sufficiently large). Since the second derivatives of \( G \) are non-positive (see [18]) it follows that

\[
G(\bar{x}^*) \geq \frac{1 - \mu}{1 + \mu} G(\bar{y}^*) \geq (1 - e^{-1}) \frac{(1 - \mu)^2}{1 + \mu} OPT(\mathcal{T}).
\]

(5.12)

For any \((S,j) \in E\) we have \(|S| \leq \mu^{-1}\), and by the submodularity of \( f, g(\{(S,j)\}) -

85
\( g(\emptyset) \leq \mu^{-1}v. \) Therefore, by the concentration bound of [21] (see Lemma 3.2.2), we have

\[
\Pr \left( g(T) \leq (1 - e^{-1}) \frac{(1 - \mu)^3}{1 + \mu} \text{OPT}(\mathcal{I}) \right) \leq \Pr \left( g(\{e \in E \mid X_e = 1\}) \leq (1 - \mu)G(\bar{x}^*) \right)
\]

\[
\leq \exp \left( -\frac{\mu^3 \cdot G(\bar{x}^*)}{2v} \right) \leq \exp \left( -\frac{\mu^3(1 - e^{-1})}{2v} \frac{(1 - \mu)^2}{1 + \mu} \text{OPT}(\mathcal{I}) \right) \leq \exp \left( -\frac{\mu^3 \cdot \text{OPT}(\mathcal{I})}{16 \cdot v} \right)
\]

The first and third inequality are due to (5.12).

Let \( \omega \) be the event \( \bar{x}^T \in (1 - \mu)P \) and \( g(T) \geq \frac{(1 - \mu)^3}{1 + \mu}(1 - e^{-1})\text{OPT}(\mathcal{I}) \). By applying the union bound over (5.9), (5.10), (5.11) and (5.13), we have

\[
\Pr(\omega) \geq 1 - \left( |B^r| \exp \left( -\frac{\mu^2}{12} \frac{1}{\delta} \right) - 2 \sum_{j=1}^{k} \exp \left( -\frac{\mu^2}{12} |B_j| \right) - \exp \left( -\frac{\mu^2 \cdot \text{OPT}(\mathcal{I})}{16 \cdot v} \right) \right) = 1 - \gamma.
\]

In case the event \( \omega \) occurs, the algorithm executes Line 5, and by Lemma 5.2.7, \( f(\cup_{b \in B} A_b) = f(T) \). Hence,

\[
\mathbb{E} [f(\cup_{b \in B} A_b)] = \Pr (\omega) \cdot \mathbb{E} [f(\cup_{b \in B} A_b) | \omega] \geq (1 - \gamma) \frac{(1 - \mu)^3}{1 + \mu}(1 - e^{-1})\text{OPT}(\mathcal{I}).
\]

Also, the algorithm either returns an empty solution when Line 7 executes, or Line 5 executes. In the latter case the solution is feasible by Lemma 5.2.7. Therefore, the algorithm always returns a feasible solution.

\[ \square \]
Chapter 6

The Capacitated MEC Problem

In this section we study the Capacitated MEC Problem. First, we present a formulation of the problem as a submodular optimization problem. Next, two algorithms are presented based on this formulation and, in addition, we present two randomized rounding algorithms based on a linear relaxation of the integer program for the problem. Finally, extensive simulation is presented, evaluating the different algorithms and comparing their performance with natural heuristics.

6.1 Submodular Formulation

An alternative way of formulating the CMAP problem is via a submodular optimization problem over the allocation of functions to infrastructure nodes. To this end, we define a new function, $R_N : 2^{I \times F} \rightarrow \mathbb{R}_{\geq 0}$; given an allocation $A \subseteq I \times F$, $R_N(A)$ is the maximum profit that can be accrued from a feasible assignment of clients to infrastructure nodes. Let $C_A$ denote a set of satisfied clients determining the value of $R_N(A)$ (clearly $C_A$ is not unique).

Function $R_N$ has several interesting properties. First, it is non-decreasing. This is easy to see as installing additional network functions on infrastructure nodes can only satisfy additional clients and increase the profit. Second, $R_N$ is a submodular function. The following claim is essential in proving the submodularity of $R_N$. 
Claim 6.1.1. For $A \subseteq B \subseteq I \times F$, for any choice of $C_A \subseteq C$, there exists a subset $C_B \subseteq C$, such that $C_A \subseteq C_B$.

Proof. For a given $C_A \subseteq C$, let us choose $C_B$ (among all choices) such that it minimizes $C_B \setminus C_A$ and assume $C_A \not\subseteq C_B$. Then, there exists a client $c \in C_A \setminus C_B$ which is assigned to $(i,f) \in A$. Let $I_A \times f$ be the infrastructure nodes to which client $c$ can be assigned to in allocation $A$. Of course $I_A \times f \subseteq B$. We denote the clients that are assigned to $I_A \times f$ in allocation $B$ by $D$. If $D \subseteq C_A$ then there exists $(i,f) \in I_A \times f$ that still has additional capacity left, and $c$ can be assigned to it in allocation $B$. This, of course, contradicts the optimality of $C_B$. Otherwise, all clients in $D \setminus C_A$ cannot have a smaller profit than $c$, otherwise we can replace one of them by client $c$ and get a larger profit. This, again, contradicts the optimality of $C_B$. Lastly, let us consider the case in which all clients in $D \setminus C_A$ have a larger profit. For each client $c'$ in $D \setminus C_A$ there is a path from $c'$ to $c$ through clients, infrastructure nodes, and edges that are used either in the allocation to $A$ or to $B$ (else there is an infrastructure node which is not fully utilized and a client we can assign to it in either $A$ or $B$). This path defines an alternating path of even length, in which odd edges belong to the allocation to $B$, and even edges belong to the allocation to $A$. We can use this path to update the allocation to $A$ such that $c'$ is assigned in $A$, all clients along the path from $c'$ to $c$ are still assigned, and $c$ is not assigned any more. This assignment has a higher profit than the original assignment to $A$, contradicting the optimality of $C_A$. Thus, we can assign $c$ to allocation $B$ instead of a client $c' \in D \setminus C_A$, producing a new assignment $C_B'$, such that $|C_B \setminus C_A| > |C_B' \setminus C_A|$, contradicting our optimality assumption. \hfill $\square$

Lemma 6.1.2. Function $R_N$ is a submodular function.

Proof. Let $A \subseteq B \subseteq I \times F$, and let $e = (i,f) \notin B$. Let $C_A, C_A', C_B$ and $C_B'$ be the subsets of clients (respectively) such that $C_A \subseteq C_A' \subseteq C_B \subseteq C_B'$ as guaranteed by Claim 6.1.1. We denote by $P(S)$ the sum of the profits of the clients in $S$. It follows that

$$R_N(A \cup \{e\}) - R_N(A) = P(C_A' \setminus C_A)$$

$$R_N(B \cup \{e\}) - R_N(B) = P(C_B' \setminus C_B).$$
Since $C_A' \subseteq C_B$, $(C_B \setminus C_B) \cap C_A' = \emptyset$. In addition, $|C_B' \setminus C_B| \leq s_{i_f}$, since this is the surplus capacity from adding $e = (i, f)$ to $B$. This means that $(C_B' \setminus C_B) \cup C_A$ can be assigned to allocation $A'$ in a feasible solution. It immediately follows that $R_N$ is submodular as

$$R_N(A \cup \{e\}) - R_N(A) \geq R_N(B \cup \{e\}) - R_N(B)$$

The last important property of $R_N$ is that it can be computed exactly in polynomial time. Given an allocation of network functions to infrastructure nodes, we need to compute the maximum achievable profit from a feasible assignment of clients to infrastructure nodes. This problem is an instance of the maximum weight $b$-matching problem in a bipartite graph, and can be solved exactly in polynomial time ([29]).

The next property is essential to providing a submodular formulation of CMAP.

**Claim 6.1.3.** The size constraints of the infrastructure nodes in CMAP are captured by a partition matroid.

**Proof.** Let $N = I \times F$ and $\mathcal{I}$ contain all subsets of legal assignments. For each infrastructure node $i$ we define $N_i = \{i\} \times F$. Notice that $\bigcup_{i \in I} N_i = N$. Further, for every subset

$$A \in \mathcal{I}, |A \cap N_i| \leq w_i.$$

It is obvious that $\mathcal{M}$ satisfies all the properties of a matroid, hence it is a partition matroid.

Now, using $R_N$ we can define a new submodular formulation for CMAP. Let $\mathcal{M} = (I \times F, \mathcal{I})$, where

$$\mathcal{I} = \{X \subseteq I \times F \mid \forall i \in I: |X \cap \{i\} \times F| \leq w_i\}$$

contains all legal allocations of network functions to infrastructure nodes. We get the following formulation for CMAP, which we denote by (Submodular-CMAP):

\[90\]
Formulating CMAP as a submodular maximization problem subject to a matroid constraint turns out to be very useful. This problem has received a lot of attention in the literature and several approximation algorithms have been developed for it.

## 6.2 Algorithms

In this section we present four approximation algorithms for CMAP. Two algorithms are based on (Submodular-CMAP) as formulated in Section 6.1, and two LP rounding algorithms are based on (IP-CMAP), as formulated in Section 1.3.

### 6.2.1 Submodularity-Based Algorithms

We consider two variants of the greedy algorithm, the Discrete Greedy and the Continuous Greedy. The Discrete Greedy algorithm (see Fig 8) evaluates the additive profit from allocating each network function on each infrastructure node (such that the resulting allocation remains feasible), and selects the allocation yielding the highest increase in profit. This is done iteratively until the allocation cannot be further extended.

**Algorithm 8:** Discrete Greedy algorithm

<table>
<thead>
<tr>
<th>Input:</th>
<th>An instance of CMAP.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$B = \emptyset, \hat{B} = I \times F.$;</td>
</tr>
<tr>
<td>2 while $\hat{B} \neq \emptyset$ do</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$(i, f) = \arg \max_{(i', f') \in \hat{B}} { R_N(B \cup {(i', f')}) };$</td>
</tr>
<tr>
<td>4</td>
<td>$B = B \cup {(i, f)};$</td>
</tr>
<tr>
<td>5</td>
<td>$T = {(i', f') \in \hat{B}</td>
</tr>
<tr>
<td>6</td>
<td>$\hat{B} = \hat{B} \setminus (T \cup {(i, f)}).$</td>
</tr>
<tr>
<td>7 Set $A$ as a maximum assignment to allocation $B.$;</td>
<td></td>
</tr>
<tr>
<td>8 Allocation $B$, assignment $A.$;</td>
<td></td>
</tr>
</tbody>
</table>
Nemhauser et al. [74] showed that the greedy algorithm returns a $\frac{1}{2}$-approximation solution for maximizing a submodular function over a matroid constraint. As we saw in Section 6.1, CMAP can be formulated as a submodular maximization problem over a partition matroid constraint. Thus, we can deduce the following.

**Theorem 6.2.1.** The Discrete Greedy algorithm returns a $\frac{1}{2}$-approximation for CMAP.

The following simple example shows that this analysis is tight. We define an instance of CMAP in which we are given two infrastructure nodes $i_1, i_2$, two network functions $f_1, f_2$, and two clients $c_1, c_2$, such that $h(c_1) = f_1, h(c_2) = f_2$ and $p(c_1) = p(c_2) = 1$. In addition, $w_{i_1} = w_{i_2} = 1$ and client $c_1$ can connect to both $i_1$ and $i_2$, but client $c_2$ can only connect to $i_1$. Initially, when running the Discrete Greedy algorithm, the profit earned from selecting one of the pairs $(i_1, f_1), (i_1, f_2)$ and $(i_2, f_1)$ is exactly 1. Let us assume that the algorithm selected $(i_1, f_1)$. Then, the algorithm cannot serve client $c_2$ as it can only be served by infrastructure node $i_1$, and it cannot allocate $f_2$ on $i_1$, since that would violate its size constraint. Thus, the algorithm earns a profit of 1. However, the optimal solution will select $(i_1, f_2), (i_2, f_1)$, and is able to serve both clients. This results in a profit of 2, thus the ratio between the Discrete Greedy solution and the optimal solution is $\frac{1}{2}$.

We denote the multilinear extension of $R_N$ by $R_{ML}$ and note that the value of $R_{ML}$ of a fractional allocation vector $x \in [0,1]^{I \times F}$ can be efficiently computed. The Continuous Greedy algorithm starts with an empty solution vector $x = \vec{0}$. It then evaluates using $R_{ML}$ the profit of adding a small fraction $\delta \in (0,1)$ of each node pair of (function, infrastructure). Considering small increments allows the algorithm to “ignore” the submodularity of the objective function and thus solve each step as if the objective function is linear. This process results in a fractional solution that approximates the optimal integral solution. Finally, the fractional solution is rounded with no loss.
**Algorithm 9: Continuous Greedy algorithm**

**Input:** An instance of CMAP, $\delta \in (0, 1)$.

1. $x = \vec{0}$;
2. for $j \in 1, \ldots, \frac{1}{\delta}$ do
   3. $\forall i \in I, f \in F : p_{if} = \frac{R_{ML}(x + \delta \cdot x_{if}) - R_{ML}(x)}{\delta}$;
   4. $\forall i \in I : B_i = \arg \max_{T \subseteq \{i\} \times F, |T| \leq w_i} \left\{ \sum_{(i,f) \in T} p_{if} \right\}$;
   5. $B_\delta = \bigcup_{i \in I} B_i$;
   6. $x = x + \delta \cdot \sum_{(i,f) \in B_\delta} e_{if}$;
   7. $\forall i \in I, f \in F$: add $(i, f)$ to $B$ with probability $x_{if}$;
8. Set $A$ as a maximum assignment to allocation $B$;

It was shown by Calinescu et al. [17] that the continuous greedy algorithm returns a $1 - \frac{1}{e} - \frac{1}{\pi^2}$ approximation fractional solution for maximizing a submodular function over a matroid constraint when $\delta = \frac{1}{\pi^2}$. The fractional solution can be rounded into an integral solution with no further loss at all, using the pipage rounding technique that was introduced by Ageev et al. [1]. As we have already shown, CMAP can be formulated as a submodular maximization problem over a matroid constraint. This means that the Continuous Greedy algorithm returns a $1 - \frac{1}{e} - o(1)$ approximate (integral) solution for CMAP. To summarize,

**Theorem 6.2.2.** The Continuous Greedy algorithm returns a $1 - \frac{1}{e} - o(1)$ approximation for CMAP.

### 6.2.2 LP based algorithms

By relaxing the integer constraints of (IP-CMAP) (see Section 1.3) and replacing them with linear constraints, $x_{ic} \geq 0$, we obtain a linear program that is solvable in polynomial time. The fractional solution, which upper bounds the optimal integral solution, can be rounded in order to attain an approximate solution with proven bounds.

We present two rounding algorithms for the LP. The first algorithm, LP Fully Random, a fully randomized rounding algorithm. The algorithm randomly selects both the allocation of functions to infrastructure nodes, and the assignment of clients.
Algorithm 10: LP Fully Random algorithm

**Input:** An instance of CMAP, $\delta \in (0, 1)$.

1. Solve the linear relaxation of (IP-CMAP).;
2. $\forall i \in I, f \in F$: add $(i, f)$ to $B$ with probability $y_{if}$;
3. $\forall (i, f) \in B, c \in C$: $A(c) = i$ with probability $(1 - \delta) \frac{x_{ic}}{y_{if}}$;
4. Remove all functions for which the capacity constraint is violated. Throw away clients assigned to removed functions.;
5. Allocation $B$, assignment $A$.

**Theorem 6.2.3.** For every $\epsilon \in (0, 1)$, LP Fully Random algorithm returns a $(1 - \epsilon)(1 - \frac{1}{e})$ approximation solution under the assumption that $\forall i \in I, f \in F : s_{if} \geq \epsilon^{-2}(1 - \epsilon)^{-1}\ln(\epsilon^{-1})$.

**Proof.** We first ignore the last step in which network functions are removed, and show that each client is assigned to an infrastructure node with high probability. Then we prove that an assigned client is thrown away with low probability.

The algorithm only assigns clients to infrastructure nodes in which the function they demand is allocated. Thus, the probability client $c$ is assigned to infrastructure node $i$ is $(1 - \delta) \frac{x_{ic}}{y_{if}}$, and if $\sum_i x_{ic} = x_c$, then the probability that client $c$ was assigned to some infrastructure node is at least

$$1 - \prod_i (1 - (1 - \delta)x_{ic}) \geq (1 - \frac{1}{e})(1 - \delta)x_c$$

From this we can conclude that the total expected profit earned prior to the removal of clients and functions is $(1 - \delta)(1 - \frac{1}{e})OPT$.

In the removal step the profit may decrease if we remove an allocated function to which clients were assigned. Let us denote the event that client $c$ with demand $h(c) = f$ is assigned to infrastructure node $i$ by $I_{icf}$. Using standard Chernoff bounds [23] we can bound the probability in which network function $f$ is removed from infrastructure node $i$ and client $c$ was assigned to it by:

$$\Pr[\sum_c (1 - \delta)x_{ic} \geq s_{if}|I_{icf}] \leq \delta$$

94
It is easy to see that we lose at most another factor of $1 - \delta$ in the removal step as we lose the profit gained from a satisfied client with probability $\delta$. Overall we get an approximation ratio of $(1 - \delta)^2(1 - \frac{1}{\epsilon}) = (1 - \epsilon)(1 - \frac{1}{\epsilon})$.

The LP Fully Random algorithm ignores the fact that given an allocation of network functions to infrastructure nodes we can find an optimal assignment of clients to the allocation (as seen in Section 6.1 we can calculate the value of $R_N$). Instead, the algorithm continues to rely on the fractional solution in order to randomly select the assignment.

Our second LP based algorithm, LP Max Assignment algorithm, starts by randomly selecting the allocation, as done in LP Fully Random, and then selects the optimal assignment. Since the optimal assignment of clients to infrastructure nodes is better than a random assignment, the approximation ratio achieved must be at least as good as the approximation ratio of Algorithm 10. In practice, selecting the optimal assignment greatly improves over the random assignment. This is demonstrated in Section 6.3. We summarize with:

**Algorithm 11: LP Max Assignment algorithm**

**Input:** An instance of CMAP.

1. Solve the linear relaxation of IP-CMAP;
2. $\forall i \in I, f \in F$: add $(i, f)$ to $B$ with probability $y_{if}$;
3. Find an optimal assignment $A : C \rightarrow I$ to allocation $B$;

**Theorem 6.2.4.** For every $\epsilon \in (0, 1)$, LP Max Assignment algorithm returns a $(1 - \epsilon)(1 - \frac{1}{\epsilon})$ approximation solution under the assumption that $\forall i \in I, f \in F : s_{if} \geq \epsilon^{-2}(1 - \epsilon)^{-1}\ln(\epsilon^{-1})$.

### 6.3 Experimental Evaluation

In this section, we evaluate the performance of our algorithms in realistic scenarios. We use the locations of cellular antennas around Central Park in New York City\(^1\) as

\(^{1}\)Data is available in https://opencellid.org
infrastructure node locations (see example in Figure 6.1). Client locations are randomly chosen in a bounded area between infrastructure node locations. In order to simulate reception range of the antennas realistically, infrastructure nodes are only connected to clients located within a fixed radius around them.

We measure the performance as the percentage of satisfied client algorithm $A$ achieves compared to the fractional optimal solution,

$$Performance(A) = 100 \cdot \frac{ALG}{FRAC-OPT}.$$  

We also use, throughout the evaluation, the term network load. The network load of instance $I$ is the overall number of clients divided by the total function capacity. We study the effect of multiple parameters, e.g., network load, the number of infrastructure nodes, and the number of network functions, on the performance of our algorithms,

![Figure 6.1: Central Park cellular antennas](image)

We first compare the Discrete greedy algorithm, Continuous greedy algorithm and Fractional LP rounding algorithms to a simple random heuristic that selects, uniformly at random, a network function allocation for each infrastructure node. A maximum profit assignment is then found for this random allocation. The quality of the results in terms of practical deployment is measured by two parameters: (i) performance, as
defined above (see e.g., Figure 6.2), and (ii) the average running time (see Table 6.1).

Figure 6.2: Algorithm performance with respect to the number of clients. Network settings: 20 infrastructure nodes, 5 network functions, uniform function capacity of 5

Figure 6.2 depicts the performance of the three algorithms and the Random heuristic. Despite the fact that the worst case theoretical guarantees of the Continuous Greedy algorithm are much higher, the Discrete Greedy and LP Max Assignment algorithms perform better in this setting. This is due to the fact that the theoretical bound are proven for the worst possible case while the behaviour over average cases might be different.

A close examination of the LP Max Assignment and Discrete Greedy performance graph reveals that the worst performance of both algorithms occurs when the load on the network is about 100%. The high performance at extreme loads can be explained by the ease of utilizing the full coverage potential in these scenarios. in low load, all allocations offer small benefit, and the intersection of infrastructure nodes is small. On the other hand, if the load is high, it is easy to exhaust all the capacity of the network functions regardless of the allocation.

In Figure 6.2 the performance graph of the Continuous Greedy algorithm behaves similarly to the graph of the random heuristic. However, this is not the case in Figure 6.3 where one can observe that the performance of Continuous Greedy does not decline as the number of VNFs grows, in contrast to the random heuristic which deteriorate.
As the number of VNFs grows, the probability in which a random function allocation produces the maximum profit decreases. This can explain the behaviour of the random heuristic when the number of VNFs grows.

Table 6.1: Algorithms running time. Network settings: 20 infrastructure nodes, 100 clients, 5 network function, uniform function capacity of 5

Table 6.1 presents the running time of the Discrete Greedy algorithm, Continuous Greedy algorithm, and LP Max Assignment algorithm on the same network settings. One can immediately notice that the Discrete Greedy and LP Max Assignment algorithms not only outperform the Continuous Greedy algorithm, but also their running time is shorter by multiple magnitudes. We conclude that in realistic scenarios the Continuous Greedy is irrelevant, and in the rest of this section we only present the Discrete Greedy and LP Max Assignment algorithms.

In order to compare the performance of our algorithms against the best possible heuristics, we consider an additional heuristic that we call Smart Random. Smart
Random allocates a network function to an infrastructure node with probability proportional to the number of clients it can serve. That is, if an infrastructure node is connected to 5 clients that require the green function and 10 clients that require the red function then the red function is allocated there with probability $2/3$ and the green one with probability $1/3$. We compare our algorithms to the Random and Smart Random heuristics since we are unfamiliar with any other algorithms for CMAP.

![Algorithm performance with respect to number of network functions](image)

Figure 6.4: Algorithm performance with respect to number of network functions. Network settings: 200 infrastructure nodes, 1000 clients, uniform function capacity of 5

Figure 6.4 depicts the performance of the algorithms with respect to the number of VNFs. The network layout includes 200 infrastructure nodes across the network, 1000 clients and the load is 100%. Even though the random heuristics perform well when the number of VNFs is small, their performance deteriorate as the number of VNFs grow. On the other hand, both the Discrete Greedy and LP Max Assignment algorithms show consistent performance, an important characteristic for future 5G networks where the number of VNFs is expected to be high.

Figure 6.5 shows the algorithms performance as a function of the number of infrastructure nodes for a fixed load. Note that when the number of infrastructure nodes increases the number of clients increases as well since the clients to infrastructure nodes ratio is 5. One can see that the performance of the Discrete Greedy and LP Max Assignment algorithms do not decline as the number of clients and infrastructure nodes grow.
Figure 6.5: Algorithm performance with respect to number of infrastructure nodes. Network settings: clients to infrastructure nodes ratio is 5, 3 network functions, uniform function capacity of 5

This shows that the algorithms are scalable and they perform well on large networks.

Figure 6.6: Algorithm performance with respect to total functions capacity. Network settings: 100 infrastructure nodes, 500 clients, 3 network functions, uniform functions capacity.

Figure 6.6 presents the effect of network load on the performance of the algorithms. Here, unlike Figure 6.2, the change in network load is due to a change in the total
capacity of the functions with a fix number of clients. Once again, one can observe that the graphs of our algorithms are concave, and that our algorithms worst performance occurs when the network load is about 100%.

Throughout the evaluation we have only considered homogeneous network functions and uniform infrastructure nodes capacities. In reality, the available resources at each infrastructure node may be different, leading to different network function capacities for functions allocated on different nodes. The setting presented in Figure 6.7 is similar to the one of Figure 6.6 since Both simulations consider the same network settings and the same expected network loads. However, in Figure 6.7 the capacity of each infrastructure node is chosen uniformly at random. As can be seen, the performance of the algorithms are similar thus, this change does not affect the performance.

Figure 6.7: Algorithm performance with respect to the network load. Network settings: 100 infrastructure nodes, 500 clients, 3 network functions, random network function capacities

<table>
<thead>
<tr>
<th>Algorithm \ Load Percentage</th>
<th>62%</th>
<th>100%</th>
<th>250%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discrete Greedy</td>
<td>21.61s</td>
<td>12.53s</td>
<td>13.64s</td>
</tr>
<tr>
<td>LP Max Assignment</td>
<td>4.33s</td>
<td>3.03s</td>
<td>3.76s</td>
</tr>
</tbody>
</table>

Table 6.2: Algorithms running times (random capacities). Network settings: 100 infrastructure nodes, 500 clients, 3 network functions, random network function capacities
Table 6.2 compares the running times of the Discrete Greedy and the LP Max Assignment algorithms in this setting. Note that compared to Table 6.1, the running time here is much higher since the instances are much bigger. When comparing the Discrete Greedy running time to the running time of the LP Max Assignment algorithm, one can clearly see that the later is much faster (by a factor of \( \sim 4.5 \)). When comparing the performance (see Figure 6.7), it is similar with a small advantage to the LP Max Assignment algorithm.

When a network operator considers which algorithms suits best for its network, he should take into consideration the fact that the LP Max Assignment algorithm shows a slight performance advantage while it also runs faster. Given response times and network size, a network operator can consider running both algorithms and select the best result, or just use the faster algorithm.

Lastly, we compared the LP Max Assignment and LP Fully Random algorithms to the LP rounding algorithm presented in [53]. Their LP rounding algorithm does not include capacity constraints, so it may return an unfeasible solution that is not comparable with our algorithms. Yet, it is the only existing algorithm with known performance guarantees that can be used for this problem. We modified that algorithm,
creating the LP NoCap algorithm, by using their algorithm to find an allocation, for which we find a maximal feasible assignment. Figure 6.8 compares the three LP based algorithms and, as one can expect, LP Max Assignment outperform the LP Fully Random algorithm by a margin of $6 - 8\%$, and the LP NoCap algorithm by a margin of $12 - 14\%$. This shows that our new algorithms improve over current results and increase the network availability for clients.

To summarize, our thorough simulation based evaluation indicates that the Discrete Greedy and LP Max Assignment algorithms perform well over their respective theoretical guaranties, and vastly improve the random heuristics and other existing algorithms. In addition, these two algorithms are scalable and can be deployed on large networks with multiple network functions.
Chapter 7

The VM Scheduling Problem

In this section we study the VM Scheduling problem. We first present the techniques and main ideas used in this section. This includes a presentation of definitions and algorithms for the Static Bin Packing problem we utilize in our solutions. Next, we consider both uniform and non-uniform variants of the offline problem, and present two algorithms for these variants. Also, the online problem is considered. We present algorithm for three levels of additional information given by predictions; no additional information, life time predictions and work load predictions. Finally, lower bounds are given for the problem.

7.1 Tools and Techniques

The main issue we cope with in the online setting is how to improve the competitive ratio by utilizing the additional information provided to an online scheduler. At a high level, this is achieved by drawing on ideas from our new offline algorithms; we show that these algorithms can be to some extent “simulated” in the online case, even when less information is available. Yet, the loss to performance is bounded.

How to utilize future load predictions? When loads for each future time step are available, we draw on ideas from Algorithm 13. In each iteration, this offline algorithm
considers the unscheduled requests, and finds greedily (and carefully) a set of requests (among the unscheduled requests) that can be scheduled on one or two machines, and having high enough load in every time step. This set can be interpreted as a cover of the time horizon. The offline algorithm then repeats this process with the remaining unscheduled requests, till all requests are scheduled.

Achieving this goal online is tricky, as multiple covers of the time horizon must be created in parallel without knowing future requests. Each “error” in assigning requests to covers may either increase the number of machines required for scheduling a cover, or increase the number of covers (again, resulting in too many active machines). We show that when given information on future demand, the number of “extra” covers generated is bounded. The high level idea is to maintain several open machines (and not just two as in the offline case), and schedule a new request on the lowest index machine that “must” accept it in order to preserve a “high load” invariant. Finding the right machine is done online utilizing the predictions on the remaining future demand and the new request’s lifetime. Surprisingly, we are able to get a constant competitive factor in this case, even though the online scheduler is not familiar with the full interval structure of the instance, as in the offline setting, only with cumulative load. The results are presented in Section 7.4.

How to use average load prediction? Interestingly, we show that this single value parameter can be extremely useful in improving the competitive factor. This is done by mimicking Algorithm 14. This offline algorithm finds a dense subset of requests of roughly the same duration that can be scheduled together (similarly in spirit to the greedy set-cover algorithm). In the offline setting we show that this is possible whenever there exists a point in time in which demand is high enough.

The online scheduler is not familiar with the demand ahead of time. Hence, it uses a careful classification of the requests by their duration, the current demand, and the average demand. While the idea of classification has been used before in the context of dynamic bin packing, we classify intervals in a more sophisticated way. Finally, to get our refined bounds we schedule each class of intervals using a new family of non-clairvoyant algorithms (discussed in the sequel) that trade off carefully multiplicative
and additive terms. The results are presented in Section 7.3.

**A new family of non-clairvoyant algorithms.** To get our refined bounds we show a general reduction that transforms any $k$-bounded space (static) bin packing algorithm (see exact definitions in Section 7.1.1) into a non-clairvoyant algorithm for the dynamic bin packing problem. Note that the optimal cost in the static bin packing is simply the number of bins (and not the total duration). Hence, we use $OPT^S$ to emphasize that this is the optimal solution for static instances. We prove the following.

**Lemma 7.1.1.** Given a $k$-bounded space bin packing algorithm whose cost is at most $c \cdot OPT^S + \ell$, there exists an online non-clairvoyant algorithm for the dynamic bin packing setting whose average cost is at most $c \cdot \mu \cdot OPT_{\text{avg}} + \max\{k, \ell\}$.

For example, substituting in this theorem the performance of the Harmonic Algorithm, we obtain a non-clairvoyant algorithm whose average cost is at most $\Pi_k \cdot \mu \cdot OPT_{\text{avg}} + k$.

### 7.1.1 Static Bin Packing Algorithms

We now discuss several well known static bin packing algorithms and related definitions. We prove useful properties that are later used by our dynamic bin packing algorithms.

First, the well known First-Fit algorithm appears as Algorithm 12.

**Algorithm 12: First-Fit**

1. When an interval $I$ arrives at time $t$, assign it to a machine with the earliest opening time among the available machines. If no machine is available, open a new machine.

The following properties hold for the First-Fit algorithm.

**Lemma 7.1.2.** The total cost of Algorithm 12 is at most:

1. $\|v\|_\infty \cdot \|v\|_0$ for the uniform size case.
2. $\left(\frac{1}{1-\beta} \cdot \|v\|_\infty + 1\right) \cdot \|v\|_0$ for the non-uniform size case when $\beta \leq \frac{1}{2}$.
3. $4 \cdot \|v\|_{\infty} \cdot \|v\|_0$ for the non-uniform size case when $\beta > \frac{1}{2}$.

4. $\|v\|_0$ if $\|v\|_{\infty} \leq 1$.

**Proof.** The proof for each case follows.

**Uniform case.** Assume in contradiction that there exists a time $t$ in which an interval $I$ arrives, resulting in $\|v\|_{\infty} + 1$ open machines. This can only happen if the $\|v\|_{\infty}$ machines that were open prior to the arrival of $I$ (at time $t$) are all fully occupied. Along with the interval $I$, by the properties of first fit, it implies that the number of intervals at time $t$ is greater than $N_t$, a contradiction.

**Non-uniform case when $\beta \leq \frac{1}{2}$.** Let $M$ denote the maximum number of machines used at any time over the horizon, and let $t$ be a time where the algorithm decided to open an $M$th machine. Since the size of each interval is at most $\beta$ and at time $t$ the algorithm could not accommodate an arriving interval in the existing $M - 1$ machines, that implies that the $M - 1$ first machines have load at least $1 - \beta$ at time $t$. Furthermore, the total load of machines $M - 1$ and $M$ at time $t$ is at least 1, otherwise the algorithm would not need to open a new machine. As a result, at time $t$:

$$\|v\|_{\infty} > (M - 2) \cdot (1 - \beta) + 1 \implies \frac{1}{1 - \beta} \cdot \|v\|_{\infty} \geq M - 2 + \frac{1}{1 - \beta} \geq M - 1.$$  

Therefore, $M \leq \frac{1}{1 - \beta} \cdot \|v\|_{\infty} + 1$, as claimed.

**Non-uniform case when $\beta > \frac{1}{2}$.** Fix a time $t$. A machine is called *wide* if it has load at least $1/2$ and *narrow* otherwise. Denote by $M_w$ and $M_n$ the number of wide and narrow machines, respectively. We show that $M_w \leq 2 \cdot \|v\|_{\infty}$ and $M_n \leq 2 \cdot \|v\|_{\infty}$.

- Wide machines: Assume $M_w \geq 2 \cdot \|v\|_{\infty} + 1$. The wide machines have load at least $1/2$, thus their total load is at least $\frac{1}{2}M_w \geq \|v\|_{\infty} + \frac{1}{2} > \|v\|_{\infty}$, which is a contradiction. Thus, $M_w \leq 2 \cdot \|v\|_{\infty}$.

- Narrow machines: Assume $M_n \geq 2 \cdot \|v\|_{\infty} + 1$ and let $m$ denote the last machine activated at $t$, among the narrow machines. At time $t$, by definition, $m$ has at least one interval $I$ with size less than $1/2$. At the start time $s_I$ of $I$, the algorithm chose to assign it to machine $m$, implying that $I$ could not be assigned to all other
narrow machines, meaning they each had load at least $1/2$ at time $s_I$. Hence, the total load on the narrow machines at time $s_I$ is at least $(M_n - 1)\frac{1}{2} + w_I \geq \|v\|_\infty + w_I > \|v\|_\infty$, which is a contradiction.

As a result, at time $t$ the total number of open machines is at most $M_w + M_n \leq 4\|v\|_\infty$, and, since this is true for all $t$, we get the desired result.

**When $\|v\|_\infty \leq 1$:** If the algorithm opens more than a single machine at any time $t$, the total load at that time is more than 1, contradicting our assumption. □

**Definition 7.1.1.** An online static bin packing algorithm is said to be $k$-bounded-space if at any time the number of bins that are accepting new items (active bins) is at most $k$.

The Next-Fit algorithm is a prime example of a bounded space algorithm. It holds exactly one active bin at any time. Upon arrival of an item that does not fit in the active bin, it closes it and opens a new one (in which the new item is placed). Thus, the Next-Fit algorithm is 1-bounded.

Another important example of a bounded space algorithm is the Harmonic algorithm [64]. The $k$-bounded space Harmonic algorithm partitions the instance $\mathcal{I} = \bigcup_{j=1}^{k} \mathcal{I}_j$ such that $\mathcal{I}_j = \left\{ I \in \mathcal{I} \mid w_I \in \left(\frac{1}{j+1}, \frac{1}{j}\right]\right\}$ for $j = 1, \ldots, k - 1$ and $\mathcal{I}_k = \left\{ I \in \mathcal{I} \mid w_I \in (0, \frac{1}{k}]\right\}$. Each sub-instance $\mathcal{I}_j$ is packed separately using Next-Fit. Given an instance $\mathcal{I}$ of static bin packing, the cost of the $k$-bounded space Harmonic algorithm is $\Pi_k \cdot OPT^S(\mathcal{I}) + k$. $\Pi_k$ is a monotonically decreasing number that approaches $\Pi_\infty \approx 1.691$. $\Pi_k$ quickly becomes very close to this number, for example, $\Pi_{12} \approx 1.692$. As shown by [64], no constant bounded space algorithm can achieve an approximation ratio better than $\Pi_\infty$.

For our analysis we need the following stronger guarantee for the performance of an online static bin packing algorithm, where $A(\mathcal{I})$ denotes the cost of algorithm $A$ on instance $\mathcal{I}$.

**Definition 7.1.2.** An online (static) bin packing algorithm $A$ is $(c, \ell)$-decomposable if for every instance $\mathcal{I} = \bigcup_{j=1}^{n} \mathcal{I}_j$, $\sum_{j=1}^{n} A(\mathcal{I}_j) \leq c \cdot OPT^S(\mathcal{I}) + n \cdot \ell$.

In particular, plugging $n = 1$, an algorithm $A$ being $(c, \ell)$-decomposable implies that for any instance $\mathcal{I}$ it holds that $A(\mathcal{I}) \leq c \cdot OPT^S(\mathcal{I}) + \ell$. 

109
Lemma 7.1.3. The following algorithms are decomposable:

1. Next-Fit is $(c, 1)$-decomposable where $c = 1$ in the uniform size case and $c = \min \left\{ 2, \frac{1}{1-\beta} \right\}$ in the non-uniform size case.

2. $k$-Harmonic is $(\Pi_k, k)$-decomposable.

Proof. Following are the proofs for each of the algorithms.

Proof of (1): Next-Fit holds a single active bin that is still accepting intervals. The rest of the bins are full in the uniform size case and at least $\max\{\frac{1}{2}, 1 - \beta\}$-full in the non-uniform size case. Similarly, in an instance decomposed into $n$ sub-instances there are $n$ active bins, while the rest of the bins are full in the uniform size case and at least $\max\{\frac{1}{2}, 1 - \beta\}$-full in the non-uniform size case. This translates to a total cost of at most $OPT^S(I) + n$ in the uniform size case and $\min \left\{ 2, \frac{1}{1-\beta} \right\} \cdot OPT^S(I) + n$ in the non-uniform size case.

Proof of (2): The $k$-bounded harmonic algorithm is composed of $k$ copies of the Next-Fit algorithm. The $k - 1$ first copies (of the biggest items) can be seen as uniform size bin packing since exactly $j$ items are packed in each bin in the $j$-th copy. In the $k$-th copy sizes are not uniform, though for the sake of the analysis of the harmonic algorithm a bin is considered as full if it is $1 - \frac{1}{k}$ full. Thus, this copy of Next-Fit is also $(1, 1)$-decomposable. Decomposing each copy of Next-Fit leads to an additional cost of $n - 1$, overall $k \cdot (n - 1)$.

7.2 Offline Scheduling Algorithms

In this section we design two offline algorithms proving Theorem 1.4.3. We start with the Covering Algorithm, which iteratively finds a set of requests that cover the time horizon and can be scheduled using one or two machines. We then present our Density Algorithm that finds dense subsets of requests of roughly the same duration.
7.2.1 The Covering Algorithm

This section presents the Covering Algorithm as Algorithm 13. The main tool of the algorithm is the following idea of covers that are subsets of intervals that can be easily scheduled together.

Definition 7.2.1. Given a set of intervals $\mathcal{I}$ with load vector $v$, a subset of intervals $C \subseteq \mathcal{I}$ is an $[\ell, u]$-cover if its load vector $v'$ satisfies $v'_t \in [\min\{v_t, \ell\}, u]$ for any time $t$.

We show that we can efficiently find a cover with load at any time at most 2 in the uniform size case, and at most 1 in the non-uniform size case. This property allows us to schedule each of these covers using just two or one machines respectively. At the same time, these covers preserve a “high load” invariant (important property for our algorithm’s performance), by ensuring a lower bound on their load, as presented more formally in the following lemma.

Lemma 7.2.1. Let $\mathcal{I}$ be a set of intervals. Then, it is possible to efficiently find

- A $[1, 2]$-cover for the uniform size case.
- A $[\frac{1}{2} - \beta, 1]$-cover for the non-uniform size case when $\beta < \frac{1}{2}$.

Proof. We start with the non-uniform size case. Given a set of intervals $\mathcal{I}$ each with size less than $1/2$ (i.e., $\beta < \frac{1}{2}$), we show how to efficiently construct a $[\frac{1}{2} - \beta, 1]$-cover $C \subseteq \mathcal{I}$. Initially, we start with $C = \mathcal{I}$ and $v'$ is the load vector of the subset $C$. Clearly, initially, for every $t$, $v'_t \geq \min\{v_t, \frac{1}{2} - \beta\}$. If for every $t$, $v'_t \leq 1$ then we are done. Otherwise, there exists a time $t$ such that $v'_t > 1$. Consider the set of intervals $A \subseteq C$ that are active at time $t$. Using Lemma 1.4.2 with $\alpha = 1$, we see that there exists an interval $I$ that observes load more than $1/2$ for its whole duration. Removing $I$, the load vector of the subset $A$ remains at least $\frac{1}{2} - \beta$. Since $A \subseteq C$ removing such an interval maintains the load at any time it intersects at least $\frac{1}{2} - \beta$ also in the subset $C$. Thus, after iteratively removing these intervals we have that $v'_t \in [\min\{v_t, \frac{1}{2} - \beta\}, 1]$.

The proof for the uniform case follows the same lines. Given a set of intervals $\mathcal{I}$, we initially set $C = \mathcal{I}$. Let $v'$ be the load vector of the subset $C$. Clearly, initially, for every $t$, $v'_t \geq \min\{v_t, 1\}$. If for every $t$, $v'_t \leq 2$ then we are done. Otherwise, there exists
time $t$ such that $v'_t > 2$. Using Lemma 1.4.2 with $\alpha = 2$, we see that there exists an interval $I$ that observes load strictly more than 1 at any point. Hence, removing any such intervals (and using the fact that we are in the uniform case), the load vector of the subset $A$ remains at least 1. Since $A \subseteq C$ this is true also for the subset $C$.

Given Lemma 7.2.1 the algorithm is simple. We iteratively construct covers according to Lemma 7.2.1 and schedule each cover separately using the First-Fit algorithm.

**Algorithm 13: Covering Algorithm**

1. **In the non-uniform size case:** Schedule each interval with size greater than $\frac{1}{4}$ on a separate machine and remove it from $I$.
2. **while $I \neq \emptyset$** do
   3. Find a cover $C \subset I$ as guaranteed by Lemma 7.2.1.
   4. Schedule the intervals in $C$ using Algorithm 12 (First-Fit), and remove the intervals from $I$.

**Theorem 7.2.2.** The total cost of Algorithm 13 is at most $2 \cdot \|v\|_1$ for the uniform size case, $4 \cdot \|v\|_1$ in the non-uniform case. If $\beta \leq \frac{1}{4}$ the total cost is at most $\sum_t \lceil \frac{vt}{2 - \beta} \rceil$.

**Proof.** Consider an iteration $r$ of the while loop of Algorithm 13. Let $I^r$ denote the set of intervals in the beginning of iteration $r$, i.e., the intervals that have not been assigned in previous iterations. Denote by $v^r$ the load vector of $I^r$, and by $C^r$ the cover that was obtained by Lemma 7.2.1 during iteration $r$. Let $v(C^r)$ be the load vector of $C^r$.

For the uniform case, since $v_t(C^r) \leq 2$ for all $t$ by construction, we have $\|v(C^r)\|_\infty \leq 2$, and the total cost of the algorithm in this iteration is $2 \cdot \|v^r\|_0$ by Lemma 7.1.2. By the properties of the cover, $|C^r(t)| \geq \min\{v^r_t, 1\}$ for every $t$. Hence, $\|v^r\|_1$ decreases by at least $\|v^r\|_0$. Summing up over all iterations we get that the cost is at most $2 \cdot \|v\|_1$.

For the non-uniform case, in each iteration $r$, $\|v(C^r)\|_\infty \leq 1$, so First Fit schedules the corresponding intervals using one machine by Lemma 7.1.2. Hence, if the maximum size interval is at most $\frac{1}{4}$, by the properties of the cover, $|C^r(t)| \geq \min\{v^r_t, \frac{1}{2} - \beta\}$ for every $t$, so summing up over all iterations, the algorithm pays at most $\sum_t \lceil \frac{vt}{2 - \beta} \rceil$. If $\beta > \frac{1}{4}$, let $W_t$ be the number of intervals with width larger than $\frac{1}{4}$ that are active at...
time \( t \). Since the algorithm opens a separate machine of unit size for each of them, it pays cost \( W_t \) at each time \( t \). Let \( v^w \) and \( v^n \) be the load vector of the intervals that have size more than \( \frac{1}{4} \) and at most \( \frac{1}{4} \) respectively, and let \( \beta_n \) be the largest size of intervals in \( v^n \). At each time \( t \) the algorithm pays:

\[
W_t + \left\lceil \frac{v^n_t}{2 - \beta_n} \right\rceil \leq W_t + \left\lceil 4v^n_t \right\rceil \leq 4 \cdot \left\lceil \frac{1}{4} \cdot W_t + v^n_t \right\rceil \leq 4 \cdot \left[ v^w_t + v^n_t \right] = 4 \left\lceil v_t \right\rceil
\]

In the above, the first inequality follows from \( \beta_n \leq \frac{1}{4} \) and the subsequent equality from \( W_t \) being an integer. The second inequality is due to \( \lceil \alpha x \rceil \leq \alpha \lceil x \rceil \) for \( \alpha \) integer, and the final inequality is based on the fact that \( v^w_t \geq \frac{1}{4} \cdot W_t \) since wide intervals have by definition size larger than \( \frac{1}{4} \). Summing over all \( t \), the total cost of the algorithm is at most:

\[
\sum_t 4\left\lceil v_t \right\rceil = 4\|v\|_1
\]

\[\square\]

### 7.2.2 The Density Algorithm

In this section we design our second algorithm whose cost is at most \( c \cdot \|v\|_1 + O \left( \sum_{t=1}^{T} \sqrt{v_t \log \mu} \right) \) where \( c = 1 \) in the uniform size case and \( c = \min \left\{ 2, \frac{1}{1-\beta} \right\} \) in the non-uniform size case. As can be deduced from the performance bound, the algorithm performs best when the load is much larger than \( \log \mu \). The cost of the algorithm is up to twice less in both uniform and non-uniform cases compared to previous algorithms. We abuse here the notation of \( v_t \) and define it as \( \sum_{I \in \mathcal{I}(t)} w_I \) and not \( \left\lceil \sum_{I \in \mathcal{I}(t)} w_I \right\rceil \). We prove the theorem with respect to these smaller values of \( v_t \) (making the result only stronger).

The algorithm is based on the following lemma that shows it is possible to find very dense packing whenever the load is large.

**Lemma 7.2.3.** Let \( c = 1 \) in the uniform size case and \( c = \min \left\{ 2, \frac{1}{1-\beta} \right\} \) in the non-uniform size case. Let \( \mathcal{I} \) be a set of intervals, and let \( t \) be a time at which \( v_t \geq 2 + 4 \ln \mu \). Then, it is possible to find efficiently a set \( C \subseteq \mathcal{I}(t) \) such that \( \frac{1}{c} \leq \sum_{I \in C} w_I \leq 1 \), and
a length $\ell$ such that:

1. The length of each interval $I \in C$ is at least $\ell$.

2. All intervals in $C$ can be scheduled on a single machine of length at most

$$\ell \left(1 + 2 \sqrt{\frac{2 + 4 \ln \mu}{v_t}}\right).$$

Proof. Let $t$ be a time with $v_t \geq 2 + 4 \ln \mu$. We partition the intervals in $I(t)$ into $1 + \log_{1+\epsilon} \mu$ length classes $C_i$, with $\epsilon = \sqrt{D/v_t}$ where $D = 2 + 4 \ln \mu$. The $i$th class contains all intervals in $I(t)$ whose length is in the range $[(1 + \epsilon)^{i-1}, (1 + \epsilon)^i]$. Note that since $v_t \geq 2 + 4 \ln \mu$, then $\epsilon \leq 1$.

Consider the intervals in the $i$th class $C_i$, and let $\ell = (1 + \epsilon)^{i-1}$. By our partition, all lengths of intervals in $C_i$ are in the range $[\ell, \ell(1 + \epsilon))$. Furthermore, as they all belong to $I(t)$ (and are hence active at time $t$), the starting time of each interval $I \in C_i$ is in the range $(t - \ell(1 + \epsilon), t]$. We next, further partition the intervals in $C_i$ to $(1 + \frac{1}{\epsilon})$ sub-classes $C_{i,j}$ by their starting times. $C_{i,j}$ contains all intervals in $C_i$ whose starting time is in the time interval $(t - \ell(1 + \epsilon) + (j - 1)\ell, t - \ell(1 + \epsilon) + j \cdot \ell]$.

Overall, the total number of sets in our partition is at most,

$$\left(\frac{1}{\epsilon} + 1\right) \left(1 + \frac{\ln \mu}{\ln(1 + \epsilon)}\right) \leq \left(\frac{1}{\epsilon} + 1\right) \left(1 + \frac{2 \ln \mu}{\epsilon}\right) \leq \frac{2 + 4 \ln \mu}{\epsilon^2} = \frac{D}{\epsilon^2},$$

where the first inequality follows since for $\epsilon \leq 1$, $\ln(1 + \epsilon) \geq \frac{\epsilon}{2}$.

Hence, one of the sets must contain a load of at least $v_t \cdot \frac{\epsilon^2}{D} \geq 1$ at time $t$. This means that in the uniform case, where each interval has size $1/g$ for some integer $g$, at least one set contains at least $g$ intervals. In the non-uniform case it means there exists a set with size at least 1. Given a max size of $\beta$, this set contains a subset of size at least $\max\{\frac{1}{2}, 1 - \beta\}$. As all the intervals are of length $[\ell, \ell(1 + \epsilon))$, and their starting time is at most $\ell \epsilon$ apart, it is possible to open a machine of length at most $\ell(1 + 2\epsilon) = \ell \left(1 + 2 \sqrt{\frac{D}{v_t}}\right)$ for the selected subset of size at least $1/c$.

Using Lemma 7.2.3 we design Algorithm 14. While the load is large, Algorithm 14 iteratively finds a set of intervals that create a dense packing if scheduled together,
and removes them from the instance. When the remaining load is low, the remaining intervals are scheduled using the Covering Algorithm of Section 7.2.1.

**Algorithm 14: Density Algorithm**

1. Let $I$ be our current set of intervals.
2. while $\|v\|_\infty$ of the current set $I$ is at least $2 + 4 \ln \mu$ do
   3. Apply Lemma 7.2.3 on $t_{\text{max}} = \arg \max_i v_t$ to find subset of intervals $C \subseteq I(t_{\text{max}})$.
   4. Schedule the intervals in $C$ on a single machine, and remove $C$ from $I$.
   5. Schedule the remaining intervals using Algorithm 13.

**Theorem 7.2.4.** The total cost of Algorithm 14 is at most

$$c \cdot \|v\|_1 + O \left( \sum_{t=1}^{T} v_t \log \mu \right) \leq c \cdot OPT + T \cdot O \left( \sqrt{OPT_{\text{avg}} \log \mu} \right)$$

where $c = 1$ in the uniform size case and $c = \min \left\{ 2, \frac{1}{1-\beta} \right\}$ in the non-uniform size case.

**Proof.** Consider an iteration $r$ of the loop of Algorithm 14. Let $I'$ be the current set of intervals with corresponding load values $v'_t$, and let $C'$ and $\ell'$ be the subset of intervals and the length promised by Lemma 7.2.3. By Lemma 7.2.3, the total cost paid by the algorithm in this iteration is at most $\ell' \left( 1 + 2 \sqrt{\frac{D}{\|v'_\|_\infty}} \right)$, where $D = 2 + 4 \ln \mu$. Let $\Delta v'_t$ be the decrease in $v'_t$ after removing the intervals in the subset $C'$ from $I'$. Since the sum of sizes of intervals in $C'$ is at least $1/c$, and the length of each interval is at least $\ell'$, we get that $\sum_{t=1}^{T} \Delta v'_t \geq \frac{1}{c} \cdot \ell'$. Let $R$ be the total number of iterations in the loop. The total cost over all iterations is at most,

$$\sum_{r=1}^{R} \ell' \left( 1 + 2 \sqrt{\frac{D}{\|v'_\|_\infty}} \right) \leq \sum_{r=1}^{R} c \sum_{t=1}^{T} \Delta v'_t \cdot \left( 1 + 2 \sqrt{\frac{D}{\|v'_\|_\infty}} \right)$$

(7.1)

$$\leq c \sum_{r=1}^{R} \sum_{t=1}^{T} \Delta v'_t \cdot \min \left\{ \left( 1 + 2 \sqrt{\frac{D}{v'_t}} \right), 3 \right\}$$

(7.2)
\[ = c \sum_{r=1}^{R} \sum_{t=1}^{T} \Delta v_{t}^{r} \cdot \left( 1 + \min \left\{ 2 \sqrt{\frac{D}{v_{t}^{r}}}, 2 \right\} \right). \]

Inequality (7.1) follows since \( \sum_{t=1}^{T} \Delta v_{t}^{r} \geq \frac{1}{c} \cdot \ell^{r} \). Inequality (7.2) follows since \( \|v^{r}\|_{\infty} \geq v_{t}^{r} \), and since \( \|v^{r}\|_{\infty} \geq D \) inside the loop.

Next, for each time \( t \), we may analyze the summation \( \sum_{r=1}^{R} \Delta v_{t}^{r} \cdot \left( 1 + \min \left\{ 2 \sqrt{\frac{D}{v_{t}^{r}}}, 2 \right\} \right) \). Let \( v_{t} = v_{t}^{1} \) be the starting value in the original instance \( I \).

We get that,

\[ \sum_{r=1}^{R} \Delta v_{t}^{r} \cdot \left( 1 + \min \left\{ 2 \sqrt{\frac{D}{v_{t}^{r}}}, 2 \right\} \right) \leq v_{t} + \sum_{r} \Delta v_{t}^{r} \cdot \min \left\{ 2 \sqrt{\frac{D}{v_{t}^{r}}}, 2 \right\} \]

\[ \leq v_{t} + \sum_{r|v_{t}^{r} < D} 2 \Delta v_{t}^{r} + 2 \sqrt{D} \sum_{r|v_{t}^{r} \geq D} \frac{\Delta v_{t}^{r}}{\sqrt{v_{t}^{r}}} \]

\[ \leq v_{t} + 2D + 2 \sqrt{D} \sum_{r|v_{t}^{r} \geq D} \frac{\Delta v_{t}^{r}}{\sqrt{v_{t}^{r}}} \]

As \( \frac{1}{\sqrt{v}} \) is a decreasing function of \( v \) for \( v > 0 \), \( \sum_{r|v_{t}^{r} \geq D} \frac{\Delta v_{t}^{r}}{\sqrt{v_{t}^{r}}} \leq \frac{1}{\sqrt{D}} + \int_{D}^{\infty} \frac{dv}{\sqrt{v}} = \frac{1}{\sqrt{D}} + 2 \left( \sqrt{v_{t}} - \sqrt{D} \right) \). Plugging this, we get that the total cost of all iterations is at most \( c \cdot \|v\|_{1} + O \left( \sum_{t=1}^{T} \sqrt{v_{t} \log \mu} \right) \).

Finally, by Theorem 7.2.2, the total cost of Algorithm 13 is at most \( 4\|v'\|_{1} \), where \( v' \) is the final load vector (after applying all iterations). However, by the stopping rule of our algorithm we have \( \|v'\|_{\infty} \leq 2 + 4 \log \mu \). Hence, the total additional cost is at most:

\[ 4\|v'\|_{1} = 4 \sum_{t=1}^{T} \sqrt{v_{t}'} \cdot \sqrt{v_{t}'} \leq 4 \sum_{t=1}^{T} \sqrt{v_{t}'} (2 + 4 \log \mu) \leq O(\sum_{t=1}^{T} \sqrt{v_{t} \log \mu}) \]

Using Jensen’s inequality and substituting \( v_{avg} \leq OPT_{avg} \) we get that \( \sum_{t=1}^{T} \sqrt{v_{t} \log \mu} \leq T \cdot \sqrt{OPT_{avg} \log \mu} \), which concludes the proof. \( \square \)
7.2.3 Improving the Approximation For Non-Uniform Sizes

In this section we show how to use ideas from the analysis of the Harmonic algorithm for the static bin packing to improve the performance of algorithms for the dynamic bin packing. This can be done for any algorithm for the dynamic case (with certain good properties). The reduction is given as Algorithm 15.

Algorithm 15: Partition Algorithm (parameter $k$)

1. Let $A$ be an offline algorithm for the dynamic bin packing problem.
2. Partition $\mathcal{I}$ so that $\mathcal{I} = \bigcup_{j=1}^{k} I_j$, where $I_j = \{ I \in \mathcal{I} \mid w_I \in (\frac{1}{j+1}, \frac{1}{j}] \}$ for $j = 1, \ldots, k - 1$, and $I_k = \{ I \in \mathcal{I} \mid w_I \leq \frac{1}{k} \}$.
3. Schedule each subset $I_j$ separately using $A$.

Lemma 7.2.5. Let $A$ be an offline dynamic bin packing algorithm that for instance $\mathcal{I}$ with load vector $v$ when measured without the ceiling on each coordinate has a total cost of:

- $c \cdot \|v\|_1 + f(v)$ for the uniform size case.
- $c_\beta \cdot \|v\|_1 + g(v)$ for the non-uniform size when parametrized by $\beta$,

where $f$ and $g$ are non-decreasing functions of the load vector. Then, for integer $k \geq 3$, the total cost of Algorithm 15 is at most

$$\Pi_k \cdot \max\{c, c_1 \frac{k - 1}{k} \} \cdot OPT + (k - 1) f(2v) + g(2v)$$

If $f$ is also concave then the total cost is at most:

$$\Pi_k \cdot \max\{c, c_1 \frac{k - 1}{k} \} \cdot OPT + (k - 1) \cdot f \left( \frac{2v}{k - 1} \right) + g(2v)$$

Proof. We define a new size for each interval. For $I \in I_j, 1 \leq j \leq k - 1$ we set $w'_I = \frac{1}{j}$ and for $I \in I_k$ we set $w'_I = w_I \cdot \frac{k}{k - 1}$.

Let $v_j$ and $v'_j$ be the load vectors with respect to $w_I$ and $w'_I$ (respectively) of $I_j, 1 \leq j \leq k - 1$. In any feasible schedule at most $j$ intervals can be scheduled on the same
machine given both size functions. The interval sizes \( w' \) in instance \( I_j \) are uniform thus, the total cost of scheduling \( I_j \) is at most \( c \cdot \|v'_j\|_1 + f(v'_j) \).

The total cost of the schedule of \( I_k \) created by algorithm \( A \) with respect to load vector \( v_k \) is \( c_\beta \cdot \|v_k\|_1 + g(v_k), \beta \leq \frac{1}{k} \). The load of each machine with respect to \( w' \) is larger as the size of each interval is multiplied by \( \frac{k}{k-1} \). Thus, the total cost with respect to vector \( v'_k \) is \( c_1 \cdot \frac{k-1}{k} \cdot \|v'_k\|_1 + g(v'_k) \).

Summing over \( I_1, ..., I_k \), the total cost of the algorithm is at most

\[
\max\{c, c_1 \cdot \frac{k-1}{k}\} \cdot \sum_{j=1}^k \|v'_j\|_1 + \sum_{j=1}^{k-1} f(v'_j) + g(v'_k)
\]

\[
\leq \max\{c, c_1 \cdot \frac{k-1}{k}\} \cdot \|v'\|_1 + (k - 1) \cdot f(v') + g(v')
\]

If \( f \) is concave we can use Jensen’s inequality to bound \( \sum_{j=1}^{k-1} f(v'_j) \) from above by \( (k - 1)f(\frac{1}{k} \cdot \sum_{j=1}^{k-1} v'_j) \leq (k - 1)f(v'_j) \).

Any optimal solution must pay \( 1 \) to pack \( \Pi_k \) of the load defined by \( w' \). Thus, we can bound the optimal solution, \( OPT \geq \frac{\|v'\|_1}{\Pi_k} \). In addition, \( v' \leq 2v \) and \( \|v'\|_0 = \|v\|_0 \) which proves the lemma.

**Corollary 7.2.6.** The total cost of Algorithm 15 is at most:

- \( 2 \cdot \left(1 + \frac{1}{k-2}\right) \cdot \Pi_k \cdot OPT + k \cdot \|v\|_0 \) for \( k \geq 4 \) with Algorithm 13 as \( A \).

- \( \Pi_k \cdot OPT + \left(\sqrt{k + 1}\right) \cdot O \left(\sum_{t=1}^T \sqrt{v_t \log \mu}\right) \) with Algorithm 14 as \( A \).

**Proof.** As proven in Theorem 7.2.2 the cost of Algorithm 13 is at most \( 2\|v\|_1 \leq 2\|v\|_1 + \|v\|_0 \) in the uniform size case and \( \sum_{t=1}^T \left[\frac{2v_t}{1-2\beta}\right] \leq \frac{2}{1-2\beta} \|v\|_1 + \|v\|_0 \) in the non-uniform size case. Thus, \( c = 2, c_1 = \frac{2k}{k-2} \) and \( f(v) = g(v) = \|v\|_0 \). Thus, the total cost of Algorithm 15 with Algorithm 13 as \( A \) is at most

\[
c_1 \cdot \frac{k-1}{k} \cdot \Pi_k \cdot OPT + k \cdot \|2v\|_0 \leq 2 \cdot \left(1 + \frac{1}{k-2}\right) \cdot \Pi_k \cdot OPT + k \cdot \|v\|_0
\]

By Theorem 7.2.4 Algorithm 14 has performance guarantee \( c = 1, c_1 = \frac{k}{k-1} \) and \( f(v) = g(v) = O \left(\sum_{t=1}^T \sqrt{v_t \log \mu}\right) \) which are concave functions. Thus, the total cost of
Algorithm 15 with Algorithm 14 as $A$ is at most
\[
\Pi_k \cdot OPT + k \cdot O \left( \sum_{t=1}^{T} \frac{v_t}{k} \log \mu \right) \leq \Pi_k \cdot OPT + O \left( \sqrt{k \cdot OPT \cdot T \cdot \log \mu} \right)
\]
where the inequality follows by Jensen’s inequality.

\section{7.3 Online Algorithm Using Lifetime and Average Load Predictions}

In this section we design an algorithm having extra knowledge of the average load, which is a single value (the total load divided by the length of the time horizon). We start by presenting a transformation of certain static bin packing algorithms to the dynamic case, and then use it as a building block in the design of our Combined Algorithm. We complement this result with a lower bound when the lifetimes and average load are available to the algorithm. Finally, we discuss the effect of prediction errors, or noise, on the algorithms’ guarantees.

\subsection{7.3.1 Transforming Static Bounded Space Algorithms to Non-Clairvoyant Dynamic Algorithms}

In this section we show a general transformation of an online $k$-bounded space (static) bin packing algorithm (see Definition 7.1.1) to a non-clairvoyant online algorithm for the dynamic bin packing setting. We first define a static bin packing instance, given a dynamic bin packing instance, and prove an easy observation.

\begin{definition}
Let $\mathcal{I} = \{I_1, \ldots, I_n\}$ be an instance of the dynamic bin packing problem where the size of interval $I_j$ is $w_{I_j}$. We define a corresponding static bin packing instance, $\mathcal{I}_S = \{i_1, \ldots, i_n\}$, such that for $j = 1, \ldots, n : w_{I_j} = w_{i_j}$. Let $OPT(\mathcal{I})$ be the cost of the optimal solution for $\mathcal{I}$ and $OPT^S(\mathcal{I}_S)$ be the number of bins in an optimal solution for $\mathcal{I}_S$.
\end{definition}
**Observation 7.3.1.** For any instance $\mathcal{I}$, $OPT^S(\mathcal{I}_S) \leq OPT(\mathcal{I})$.

**Proof.** Consider the instance $\mathcal{I}$. Obviously, shrinking all intervals to unit length can only decrease $OPT$ without affecting $OPT^S(\mathcal{I}_S)$. Hence, we can assume that all intervals are of unit length. Next, consider machine $m$ that is active in the range $[s, t)$ in the optimal solution of $\mathcal{I}$. Let $I^j(m)$ be the set of intervals that have arrived in the range $[s + j - 1, s + j)$ and are assigned to $m$. Notice that all items in $\mathcal{I}_S$ that correspond to intervals in $I^j(m)$ can be placed in a single bin in a solution of $\mathcal{I}_S$. Thus, all items in $\mathcal{I}_S$ corresponding to intervals that are assigned to machine $m$ can be assigned to $\lfloor t - s \rfloor$ bins. Hence, we can construct a feasible solution for instance $\mathcal{I}_S$ of cost (number of bins) no more than $OPT(\mathcal{I})$. \hfill $\Box$

Algorithm 16 is given as input an online $k$-bounded static bin packing algorithm and applies it to the dynamic setting. The static bin packing instance is generated according to Definition 7.3.1.

---

**Algorithm 16: Dynamic non-clairvoyant algorithm**

1. Let $A$ be $k$-bounded static bin packing algorithm (which maintains at any time at most $k$ active bins $b_1, b_2, \ldots, b_k$)
2. Upon arrival of a new interval $I$:
   3. **begin**
   4. If $A$ opens a new bin, then open a new machine.
   5. If $A$ accepts the item to bin $b_i$, accept the interval to machine $m_i$.
   6. Upon departure of an interval $I$:
      7. **begin**
      8. If $I$ is not the last interval departing from its machine, do nothing.
      9. If $I$ is the last interval departing from a non-active bin, close the machine.
     10. If $I$ is the last interval departing from an active bin, close the machine and associate a new machine with the bin. Open the new machine if a new assignment to the respective active bin is made.

In the following lemma we analyze the cost of Algorithm 16.

**Lemma 7.3.2.** Let $A$ be a $k$-bounded space bin packing algorithm that for instance $\mathcal{I}_S$ has cost at most $c \cdot OPT^S(\mathcal{I}_S) + \ell$. Then, Algorithm 16 is an online non-clairvoyant
algorithm for the dynamic bin packing setting whose cost on any instance \( I \) is at most:

\[
c \cdot \mu \cdot \text{OPT}(I) + \max\{k, \ell\} \cdot \|v\|_0.
\]

If \( A \) is also \((c, \ell)\)-decomposable (see Definition 7.1.2), then its total cost when run separately on \( n \) instances \( I_1, \ldots, I_n \), where instance \( I_j \) has load vector \( v^j \) and value \( \mu_j \), is at most:

\[
c \cdot \mu_{\text{max}} \cdot \text{OPT}(I) + \max\{k, \ell\} \cdot \sum_{j=1}^{n} \|v^j\|_0,
\]

where \( I = \bigcup_{j=1}^{n} I_j \) and \( \mu_{\text{max}} = \max_{j=1}^{n} \mu_j \).

The following is obtained by plugging Next-Fit and the Harmonic algorithm into Lemma 7.3.2.

**Corollary 7.3.3.** Algorithm 16 is a non-clairvoyant algorithm with total cost of at most:

- \( c \cdot \mu \cdot \text{OPT}(I) + \|v\|_0 \) when the underlying static bin packing algorithm is Next-Fit, where \( c = 1 \) if sizes are uniform; otherwise, \( c = \min\left\{2, \frac{1}{1-\beta}\right\} \).

- \( \Pi_k \cdot \mu \cdot \text{OPT}(I) + k \cdot \|v\|_0 \) when the underlying static bin packing algorithm is Harmonic with parameter \( k \).

**Proof of Lemma 7.3.2.** Consider an instance \( I \), and let \( M \) be the set of machines that the algorithm opens due to \( A \) opening a new bin. For the purposes of this analysis, when the last interval departs from an active bin, we consider the corresponding machine closed only if no other assignment is done on this bin. Otherwise, we consider the machine to be in a “frozen” state, where it is inactive (therefore not paying any cost), but it can become active again if a new assignment is made to the respective active bin. Each of these machines appears once in the set \( M \). For each machine \( m \in M \), let \( s_m \) and \( e_m \) be the times that the first and last interval is assigned to \( m \) respectively, and let \( f_m \) denote the duration for which \( m \) remains frozen during the interval \([s_m, e_m]\) (can be zero if \( m \) never becomes frozen).
Algorithm $A$ has at most $k$ active bins at any time $t$, and an interval is accepted to a machine only if $A$ accepts the item to the corresponding active bin. As a result, at any time there are at most $k$ accepting machines. We can therefore partition all machines $M$ into $k$ sets $P'_1, \ldots, P'_k$, such that for $j = 1, \ldots, k$, any two machines $m_1, m_2 \in P'_j$ are not accepting at the same time. It is obvious that such partition can also be found for any $k' \geq k$. For the following analysis, if $\ell \geq k$, we want to consider the partition into $\ell$ sets instead of $k$. So, for simplicity of exposition, we define $\alpha = \max\{k, \ell\}$, and consider the partition $P_1, \ldots, P_\alpha$ such that any two machines $m_1, m_2 \in P_j$ are not accepting at the same time. Let $M_j$ be the number of machines in $P_j$.

Let $m^j_i$ be the machine with the $i$-th earliest start time among the machines in $P_j$. Let $s^j_i$ denote the start time of $m^j_i$, $e^j_i$ the time it accepts its last interval, and $f^j_i$ the duration for which it is frozen (can be zero). As a result, $m^j_i$ remains open at most during the interval $[s^j_i, \min\{e^j_i + \mu, T\}]$ minus its freezing periods. Furthermore, it is obvious that $s^j_i \leq e^j_i$, $\forall i \in 1, \ldots, M_j - 1$, and by the properties of the algorithm, $e^j_i \leq s^j_{i+1}$, $\forall i \in 1, \ldots, M_j - 1$, since $m^j_{i+1}$ can become accepting only after $m^j_i$ has stopped accepting intervals, i.e. after time $e^j_i$. Finally, let $F_j = \sum_{i=1}^{M_j} f^j_i$ denote the total freezing time over all machines in $M_j$. The active machine-time required by the machines of each set $P_j$ is at most:

$$\sum_{i=1}^{M_j} (\min\{T, e^j_i + \mu\} - s^j_i - f^j_i) \leq T - s^j_{M_j} - F_j + \sum_{i=1}^{M_j-1} (e^j_i + \mu - s^j_i)$$

$$\leq \mu \cdot (M_j - 1) + T - s^j_{M_j} - F_j + \sum_{i=1}^{M_j-1} (s^j_{i+1} - s^j_i)$$

$$\leq \mu \cdot (M_j - 1) + T - s^j_1 - F_j \leq \mu \cdot (M_j - 1) + \|v\|_0.$$

Summing up the costs for all $j = 1, \ldots, \alpha$, the total cost of the algorithm is at most:

$$\mu \cdot \sum_{j=1}^{\alpha} (M_j - 1) + \alpha \cdot \|v\|_0.$$

Notice that the algorithm can open a new machine only when $A$ opens a new bin.
Since by the guarantee of $A$ the optimal number of bins is at most $c \cdot OPT^S(I_S) + \ell$, we conclude that:

$$\sum_{j=1}^{\alpha} M_j \leq c \cdot OPT^S(I_S) + \ell.$$ 

Using this observation, we obtain that the total cost of the algorithm is at most:

$$\mu \cdot \sum_{j=1}^{\alpha} (M_j - 1) + \alpha \cdot \|v\|_0 \leq \mu \cdot (c \cdot OPT^S(I_S) + \ell - \alpha) + \alpha \cdot \|v\|_0$$

$$\leq \mu \cdot c \cdot OPT(I) + \alpha \cdot \|v\|_0,$$

where the last inequality follows from Observation 7.3.1. This concludes the first part of the proof.

For the second part of the proof, let $A$ be a $(c, \ell)$-decomposable algorithm and assume that we run it separately on $n$ instances $I_1, \ldots, I_n$, where $I = \cup_{r=1}^{n} I_r$. Let $v_r$ be the load vector of $I_r$. We denote by $P_r^1, \ldots, P_r^\alpha$ the partition of the machines opened in the solution of $I_r$, and by $M_r^j$ the number of machines in $P_r^j$. Using previous arguments, the total cost of the solution for instance $I_r$ is at most:

$$\mu_r \cdot \sum_{j=1}^{\alpha} (M_r^j - 1) + \alpha \cdot \|v_r\|_0$$

Since algorithm $A$ is $(c, \ell)$-decomposable,

$$\sum_{r=1}^{n} \sum_{j=1}^{\alpha} M_r^j \leq c \cdot OPT^S(I_S) + n \cdot \ell$$

As a result, the total cost of the algorithm is at most:

$$\sum_{r=1}^{n} \mu_r \cdot \sum_{j=1}^{\alpha} (M_r^j - 1) + \alpha \cdot \sum_{r=1}^{n} \|v_r\|_0 \leq \mu_{\max} \cdot (c \cdot OPT^S(I_S) + n \cdot \ell - n \cdot \alpha) + \alpha \cdot \sum_{r=1}^{n} \|v_r\|_0$$

$$\leq \mu_{\max} \cdot c \cdot OPT(I) + \alpha \cdot \sum_{r=1}^{n} \|v_r\|_0$$
where again the last inequality follows from Observation 7.3.1. This concludes the proof.

7.3.2 Combined Algorithm

In this section we design an online algorithm (Algorithm 17) that uses the lifetimes and average load information and whose total cost is at most $\Pi_k \cdot OPT + T \cdot k \cdot O(\sqrt{\frac{v_{avg}}{\ell}} \log \mu)$. The algorithm uses two parameters. Let $\epsilon_j = \min\{1, \frac{j}{v_{avg}}\}$. We say that an interval $I$ is in class $c^j$ if its predicted length $\ell_I \in [e^{j \cdot \epsilon} \cdot e^{j \cdot \epsilon}, e^{j \cdot \epsilon} \cdot c^{j \cdot \epsilon}]$. Let $v^j$ be the load vector of intervals in class $c^j$. The second parameter used by the algorithm is $q_j = \min\{1, \sqrt{\frac{v_{avg}}{j}}\}$. Note that the values $\epsilon_j$ and $q_j$ depend only on $v_{avg}$ and not on $\mu$.

The way the algorithm schedules the intervals differs depending on the class of each interval and its current load. In particular, when an interval $I$ arrives, if its current class load is low, the Combined Algorithm schedules it using First-Fit. If its current class load is high, then the Combined Algorithm schedules it with intervals from the same class using Algorithm 16. The intuition is that if many intervals from the same class (having similar length) arrive close in time, we can achieve better packing by scheduling them together. To achieve that, the Combined Algorithm runs many copies of Algorithm 16 simultaneously (one for each class), and schedules intervals of each class using the respective copy.

Algorithm 17: Combined Algorithm (with $v_{avg}$ prediction)

1. Hold a single copy of Algorithm 12 (First-Fit), and several copies of Algorithm 16 with an underlying (static) $k$-bounded space $(c, \ell)$-decomposable algorithm (see Definitions 7.1.1, 7.1.2).
2. Upon arrival of a new interval $I \in c^j$ at time $t$:
   3. if $v^j_t \leq q_j$ then
      4. Schedule the interval $I$ using the single copy of Algorithm 12 (First-Fit).
   5. else
      6. Schedule the interval $I$ using the $j$th copy of Algorithm 16.

Let $v^{j1}$ be the load vector of intervals $I \in c^j$ that were scheduled by Algorithm 12. Let $v^{j2}$ be the load vector of intervals $I \in c^j$ that were scheduled be Algorithm 16. By
this definition $v^j = v^{j1} + v^{j2}$.

Lemma 7.3.4. For any $j$,

1. $\|v^{j1}\|_\infty \leq q_j$.
2. $\|v^{j2}\|_0 \leq (1 + e^{\epsilon_j})\{t \mid v^{j1}_t \geq \frac{q_j}{2}\} \leq (1 + e^{\epsilon_j})\sum_{i=1}^T \min\{1, \frac{2v^{j1}_t}{q_j}\}$.

Proof. We prove the two claims.

Proof of (1): Consider any $t$, and take the intervals in $v^{j1}_t$. Let $I$ be the interval in $v^{j1}_t$ that arrived last, and let $t' \leq t$ be its start time. Since $I$ was scheduled using Algorithm 12, $v^{j1}_{t'} \leq q_j$. At time $t'$, all other intervals in $v^{j1}_t$ are alive, and no other interval arrives in $v^{j1}_t$ between $t'$ and $t$, since $I$ was the last one. Therefore, $v^{j1}_t \leq q_j$, and since this holds for all $t$, $\|v^{j1}\|_\infty \leq q_j$.

Proof of (2): Since we are working with a single class $c^j$, we will remove the index $j$ and simply refer to it as $c$ and to its load vector as $v$. Let $[\ell, \ell \cdot e^\epsilon)$ denote the lengths of intervals in class $c$. Let $c^\prime$ be the set of intervals in $c$ that were scheduled by Algorithm 16, and let $v^\prime$ be the load of $c^\prime$. Finally, we use $R_1, R_2, \ldots, R_r$ for the disjoint ranges (time intervals) in which the load is at least $\frac{q}{2}$ and there is an interval $I \in c^\prime$ with $s_I \in R_i$. We use $|R_i|$ for the total duration of $R_i$.

By the behavior of the algorithm, for each interval $I \in c^\prime$, we have $v_{s_I} > q$, i.e. $s_I \in R_i$ for some index $i$. Therefore, all $I \in c^\prime$ can be associated with a range $R_i$ with $s_I \in R_i$. Let $c^\prime_i$ be the set of intervals that are associated with range $R_i$. We prove that:

$$\|c^\prime\|_0 \leq \sum_{i=1}^r \|c^\prime_i\|_0 \leq \sum_{i=1}^r (|R_i| + \ell \cdot e^\epsilon) \leq (1 + e^\epsilon) \sum_{i=1}^r |R_i| \leq (1 + e^\epsilon)|\{t \mid v^\prime_t \geq \frac{q_j}{2}\}|$$

The first inequality follows since $\bigcup c^\prime_i = c^\prime$. The second inequality follows since each interval $I \in c^\prime_i$ starts within $R_i$ and ends at most $\ell \cdot e^\epsilon$ after $R_i$’s end. The last inequality is true by definition of $R_i$, and we will now show that $|R_i| \geq \ell$ for all $i$ which proves the third inequality and completes this part of the proof.

Consider a range $R_i$ and take an interval $I \in c^\prime_i$ with start time $s_I \in R_i$. Let $A \subseteq c$ be the intervals that are active at time $s_I$. Using Lemma 1.4.2 with $\alpha = q$, we see that there exists an interval $I \in A$ (with length at least $\ell$) that is active at time $t$ and sees
more than $\frac{q}{2}$ load for its whole duration. This means that it is associated with range \( R_i \) (otherwise the ranges would not be disjoint), and proves that \( R_i \) has length at least \( \ell \).

Finally, it is easy to see that \( |\{ t \mid v_j^t \geq \frac{q_j}{2} \}| = \sum_{t=1}^T \min\{1, \lfloor \frac{2v_j^t}{q_j} \rfloor \} \) which concludes the proof. \( \square \)

**Lemma 7.3.5.** Let \( n \) be the maximal class such that \( c^n \neq \emptyset \). Then,

\[
n = \begin{cases} 
O(\sqrt{v_{\text{avg}} \log \mu}) & v_{\text{avg}} \geq 2 \log \mu \\
O(\log \mu) & v_{\text{avg}} \leq 2 \log \mu 
\end{cases}
\]

**Proof.** The value \( n \) satisfies that \( e^{\sum_{j=1}^{n-1} \epsilon_j} \leq \mu \leq e^{\sum_{j=1}^{n} \epsilon_j} \), which means \( \sum_{j=1}^{n-1} \epsilon_j \leq \log \mu \).

By the choice of \( \epsilon_j = \min\{1, \frac{j}{v_{\text{avg}}} \} \), we get:

\[
\min\{n, v_{\text{avg}}\} - 1 \sum_{j=1}^{n-1} \frac{j}{v_{\text{avg}}} + \sum_{j=1}^{n-1} 1 = \min\left\{ \frac{(n-1)n}{2v_{\text{avg}}}, \frac{v_{\text{avg}} - 1}{2} \right\} + \max\{0, n - v_{\text{avg}}\} \leq \log \mu \tag{7.3}
\]

We now consider separately the two cases:

- **When** \( v_{\text{avg}} \geq 2 \log \mu \): In this case, \( n \leq v_{\text{avg}} \), as if we assume the contrary, (7.3) leads to a contradiction. Therefore, (7.3) becomes \( \log \mu \geq \frac{(n-1)n}{2v_{\text{avg}}} \). Rearranging this gives \( n = O(\sqrt{v_{\text{avg}} \log \mu}) \).

- **When** \( v_{\text{avg}} \leq 2 \log \mu \): If \( n \leq v_{\text{avg}} \), then \( n \leq 2 \log \mu \) and we are done. In the opposite case, (7.3) becomes:

\[
\frac{v_{\text{avg}} - 1}{2} + n - v_{\text{avg}} \leq \log \mu \Rightarrow n \leq \log \mu + \frac{v_{\text{avg}}}{2} + \frac{1}{2} \leq 2 \log \mu + \frac{1}{2}
\]

and this concludes the proof. \( \square \)

We are now ready to prove Theorem 7.3.6.
Theorem 7.3.6. Given an instance of Clairvoyant Bin Packing with average load prediction, the total cost of Algorithm 17 when it is executed with an underlying $k$-bounded space $(c, \ell)$-decomposable algorithm is at most $c \cdot \OPT + T \cdot \max\{k, \ell\} \cdot O(\sqrt{\text{avg} \log \mu})$.

Proof. The total cost of the algorithm is composed of two parts; the cost of Algorithm 12 (First-Fit) and the cost of all copies of Algorithm 16. Let $I_j$ be the set of intervals scheduled by the $j$-th copy of Algorithm 16 and $I' = \bigcup_{j=1}^{n} I_j$. The underlying static bin packing algorithm of Algorithm 16 is $k$-bounded and $(c, \ell)$-decomposable. Thus, by Lemma 7.3.2 the total cost of scheduling $I'$ is at most

$$c \cdot e^\epsilon n \cdot \OPT(I') + \max\{k, \ell\} \cdot \sum_{j=1}^{n} \|v^{j2}\|_0$$

The cost of Algorithm 12

$\text{1}$ is at most $4T \cdot \|\sum_{j=1}^{n} v^{j1}\|_\infty$. Combine the two bounds

$\text{1}We remark that in the analysis of Algorithm 12 we took the worst case performance of 4. This only affects the performance by additional constants.
to get:

\[
\text{Cost of Alg} \leq 4T \cdot \| \sum_{j=1}^{n} v^j \|_{\infty} + c \cdot e^{\epsilon_n} \cdot OPT(I) + \max\{k, \ell\} \cdot \sum_{j=1}^{n} \| v^j \|_0
\]

\[
\leq 4T \cdot \sum_{j=1}^{n} q_j + c \cdot e^{\epsilon_n} OPT(I) + \max\{k, \ell\} \cdot \sum_{j=1}^{n} (1 + e^{\epsilon_j}) \sum_{t=1}^{T} \min\{1, \frac{2v^j_t}{q_j}\}
\]

(7.4)

\[
\leq 4T \cdot \sum_{j=1}^{n} q_j + c \cdot e^{\epsilon_n} OPT(I) + 2(1 + e) \max\{k, \ell\} \cdot \sum_{j=1}^{n} \min\{T, \frac{\| v^j \|_1}{q_j}\}
\]

(7.5)

\[
\leq 4T \cdot \sum_{j=1}^{n} q_j + c \cdot e^{\epsilon_n} OPT(I) + 2(1 + e) \max\{k, \ell\} \cdot T \cdot \min\{n, \frac{v_{\text{avg}}}{q_n}\}
\]

(7.6)

\[
\leq c \cdot OPT(I) + T \cdot \max\{k, \ell\} \cdot O\left( \epsilon_n \cdot v_{\text{avg}} + \sum_{j=1}^{n} q_j + \min\{n, \frac{v_{\text{avg}}}{q_n}\} \right)
\]

(7.7)

Inequality (7.4) follows by using the fact that \( \epsilon_j \) is increasing as \( j \) is larger, and applying Lemma 7.3.4. Inequality (7.5) follows since \( \epsilon_j \leq 1 \) and using that \( \sum_{t=1}^{T} \min\{1, \frac{v^j_t}{q_j}\} \leq \min\{T, \frac{\| v^j \|_1}{q_j}\} \). Inequality (7.6) follows since \( q_j \) is non-increasing in \( j \). Finally, Inequality (7.7) follows by rearranging and using that \( \epsilon_j \leq 1 \) and hence \( e^{\epsilon_n} \leq 1 + (e - 1)\epsilon_n = 1 + O(\epsilon_n) \) and the fact that \( OPT \leq 4\| v \|_1 \). We next analyze two cases:

Case 1, \( v_{\text{avg}} \geq 2 \log \mu \): In this case, by Lemma 7.3.5, \( n = O(\sqrt{v_{\text{avg}} \log \mu}) \). Substituting \( n \) for this value, we get that \( \epsilon_n = \min\{1, \frac{n}{v_{\text{avg}}}\} \leq \frac{n}{v_{\text{avg}}} = O(\sqrt{\frac{\log \mu}{v_{\text{avg}}}}) \). Finally, as \( q_j \leq 1 \), \( \sum_{j=1}^{n} q_j \leq n = O(\sqrt{v_{\text{avg}} \log \mu}) \). Plugging these bounds into Inequality (7.7) we get the desired result.

Case 2, \( v_{\text{avg}} \leq 2 \log \mu \): In this case \( v_{\text{avg}} = O(\sqrt{v_{\text{avg}} \log \mu}) \). By the Lemma 7.3.5, \( n = O(\log \mu) \). Using this bound we get that: \( q_n = \min\{1, \sqrt{\frac{v_{\text{avg}}}{n}}\} \). Thus, \( \frac{v_{\text{avg}}}{q_n} \leq \frac{v_{\text{avg}}}{1} \leq 2 \log \mu \).
max\{v_{avg}, O(\sqrt{v_{avg} \log \mu})\} = O(\sqrt{v_{avg} \log \mu})$. Finally,

$$\sum_{j=1}^{n} q_j \leq \sum_{j=1}^{n} q_j + \sum_{j=v_{avg}}^{n} q_j \leq v_{avg} + \sum_{j=v_{avg}}^{n} \sqrt{\frac{v_{avg}}{j}} \leq v_{avg} + \sqrt{v_{avg}} \cdot n = O(\sqrt{v_{avg} \log \mu})$$

Using that $\epsilon_n \leq 1$, and plugging everything into Inequality (7.7) we get the desired result.

As seen in Lemma 7.1.3 the Next-Fit and Harmonic algorithms are decomposable (see Definition 7.1.2). Thus, using the Next-Fit algorithm (in the uniform size case) and the Harmonic algorithm (in the non-uniform size case) as the underlying algorithms of Algorithm 16 produces the following corollary.

**Corollary 7.3.7.** The total cost of Algorithm 17 is at most:

- $OPT + T \cdot O(\sqrt{v_{avg} \log \mu})$ (uniform size).
- $\Pi_k \cdot OPT + T \cdot k \cdot O(\sqrt{v_{avg} \log \mu})$ (non-uniform size).

### 7.3.3 Lower Bound: Dynamic Clairvoyant Bin Packing

In this section we complement the results of Section 7.3.2 by generalizing the lower bound of [4] to take into account also $OPT_{avg}$ showing that the additive term $O(\sqrt{OPT_{avg} \cdot \log \mu})$ is indeed unavoidable, if only the average future load and lifetime is available to an online algorithm.

**Lemma 7.3.8.** For any values $\mu, v_{avg}$ the total cost of any algorithm is at least:

$$\Omega\left(T \cdot \sqrt{v_{avg} \log \mu}\right) = \Omega\left(T \cdot \sqrt{OPT_{avg} \cdot \log \mu}\right).$$

**Proof.** First if $\frac{\log \mu}{v_{avg}} \leq 2$. Then the bound is meaningless since in this case the cost of $OPT$ is at least $\|v\|_1 \cdot \Omega(1) = \Omega\left(T \cdot \sqrt{v_{avg} \log \mu}\right)$. Otherwise, $v_{avg} < \frac{\log \mu}{2}$, and we show an adversary that given a parameter $a \leftarrow v_{avg}$ (the desired average load) and $\mu$, creates an instance such that the average load is always at least $a$ and the algorithm pays at
least \( \|v\|_1 \cdot \sqrt[4]{\frac{\log \mu}{2a}} \). If the actual average load of the instance is strictly more than \( a \) (the desired average load), we can extend the time horizon without adding new requests until the average load drops to the desired average, \( a \). Of course, this extension does not affect the total cost. The adversary initiates the following sequence:

- at each time \( t = 1, \ldots, \mu \) as long as the algorithm has strictly less than \( N = \sqrt{2a \log \mu} \) active machines:

- Initiate sequentially requests of size \( w = \sqrt{\frac{2a}{\log \mu}} \leq 1 \) (since \( a \leq \frac{\log \mu}{2} \)) and of increasing length of \( 2^i \), \( i = 0, 1, \ldots, \lceil \log \mu \rceil \).

Since each request is of length at most \( \mu \), the total length of the time horizon \( T \leq 2\mu \) (and this adversarial sequence can be repeated again afterwards).

First, the adversary indeed manages to make the algorithm open at least \( N = \sqrt{2a \log \mu} \) machines at each time \( t \) since otherwise the load of the requests initiated at time \( t \) is at least \( \lceil \log \mu \rceil \cdot w \geq \log \mu \cdot \sqrt{\frac{2a}{\log \mu}} = N \). Hence, the average load at each time \( t = 1, \ldots, \mu \) is at least \( w \cdot N = 2a \), and, since the length of the time horizon of the sequence is at most \( 2\mu \), the average load over the whole horizon is at least \( a \) as promised.

Let \( \ell_t \) be the longest interval that the adversary releases at time \( t \) (0 if there is no such interval). We have that:

\[
\|v\|_1 \leq 2 \sum_{t=1}^{\mu} w \cdot \ell_t \quad (7.8)
\]

\[
\leq 2w \cdot c_{alg} = c_{alg} \cdot 2\sqrt{\frac{2a}{\log \mu}} \quad (7.9)
\]

Inequality (7.8) follows since the total size of the last interval in the round dominates all previous ones at that round by the geometric power of 2 (the longest item dominates the rest). Inequality (7.9) follows by the observation that the algorithm opens a new machine for the last interval the adversary gives at a certain round. Hence, \( c_{alg} \geq \sum_{t=1}^{\mu} \ell_t \). Rearranging, we get that \( c_{alg} \geq \|v\|_1 \cdot \Omega \left( \sqrt{\frac{\log \mu}{v_{avg}}} \right) = \Omega \left( T \cdot \sqrt{v_{avg} \log \mu} \right) \).

Lastly, Theorem 7.2.2 states that \( \text{OPT}_{avg} \leq 4 \cdot v_{avg} \) which concludes the proof. \( \square \)
7.3.4 Handling Inaccurate Predictions

So far, we have assumed that predictions for either interval lengths or average load are accurate (or “noiseless”). Naturally, some predictions are in practice prone to errors. In this section we examine the performance of Algorithm 17 in the presence of prediction errors. We show that the algorithm is robust to prediction errors with respect to both \(v_{avg}\) and interval lengths. Formally,

**Theorem 7.3.9.** Suppose that Algorithm 17 is given predicted values \(v'_{avg}\) and interval lengths \(\ell'_I\) such that \(v'_{avg} \in \left[\frac{v_{avg}}{1+\delta}, v_{avg} \cdot (1+\delta)\right]\), and each \(\ell'_I \in \left[\frac{\ell_I}{1+\alpha}, \ell_I \cdot (1+\lambda)\right]\), with \(\delta, \alpha, \lambda \geq 0\). Then, its total cost is at most:

- \((1 + \alpha) \cdot (1 + \lambda) \cdot (OPT + T \cdot O(\sqrt{(1 + \delta)v_{avg} \log \mu})) \) (uniform size).
- \((1 + \alpha) \cdot (1 + \lambda) \cdot (\Pi_k \cdot OPT + Tk \cdot O(\sqrt{(1 + \delta)v_{avg} \log \mu})) \) (non-uniform size).

**Proof.** We first show how to handle the inaccuracy in predicting \(v_{avg}\). We claim that for the purpose of analysis (with loss in performance) we can assume that our prediction \(v'_{avg}\) is accurate. Indeed, if \(v'_{avg} < v_{avg}\), we can extend the time horizon with no additional requests from \(T\) to \(T'\) such that \(T' \cdot v'_{avg} = T \cdot v_{avg}\). This makes the average load \(v'_{avg}\) (and clearly does not change the costs of the algorithm and OPT). However, the additive cost increases to \(T' \cdot O(\sqrt{v'_{avg} \log \mu}) = T \cdot O(\sqrt{(1 + \delta)v_{avg} \log \mu})\). If \(v'_{avg} > v_{avg}\), we can add fictitious \(T(v'_{avg} - v_{avg})\) intervals of length 1 at the end of the time horizon. This increases (for the analysis) the cost of both the algorithm and OPT by this value. Again, this increases the actual load to \(v'_{avg}\), and the additive term becomes \(T \cdot O(\sqrt{v'_{avg} \log \mu})\).

To analyse the errors in the length predictions we observe that in this case the analysis of Algorithm 17 is almost unchanged. There are only two modifications to be made. First, we generalize Lemma 7.3.4 as follows. For any \(j\),

\[
\|v^2\|_0 \leq (1 + \alpha)(1 + \lambda)(1 + e^{\epsilon_j})|\{t \mid v^j_t \geq \frac{q_j}{2}\}| \leq (1 + \alpha)(1 + \lambda)(1 + e^{\epsilon_j}) \sum_{t=1}^{T} \min\{1, \frac{2v^j_t}{q_j}\}.
\]

Second, the maximum length ratio of all intervals scheduled by each copy of Algorithm
16 grows by the length prediction error to at most $e^{\epsilon n} \cdot (1 + \alpha)(1 + \lambda)$. Thus, the total scheduling cost is at most:

$$c(1 + \alpha)(1 + \lambda) \cdot e^{\epsilon n} \cdot OPT(\mathcal{I}') + \max\{k, \ell\} \cdot \sum_{j=1}^{n} \|v^j\|_0.$$ 

Note that as a special case of the above theorem, the algorithm pays a rather small multiplicative factor $(1 + \delta)$ when the only inaccurate prediction is in the average load. This is significant in other domains of interest in which the interval lengths are precisely known upon arrival (e.g., establishing virtual network connections, see [4]). Unfortunately, in the general case, Theorem 7.3.9 implies that the algorithm pays (at the worst case) an additional cost which is proportional to the noisiest length prediction. This result is tight as suggested by the lemma below.

**Lemma 7.3.10.** Given length prediction $\ell'_I$ such that $\ell'_I \in \left[\frac{\ell_I}{1+\alpha}, \ell_I \cdot (1 + \lambda)\right]$ with $\alpha, \lambda \geq 0$, the cost of any online algorithm is at least $(1 + \alpha)(1 + \lambda)OPT$.

**Proof.** At time 0 the adversary adds $k^2$ intervals with predicted length $(1 + \lambda)$ and width $\frac{1}{k}$. All these intervals are located on at least $k$ machines. The true length of each interval is as follows: on each machine there is exactly one interval of length $(1 + \alpha)(1 + \lambda)$ and the length of the rest of the intervals is 1.

The total cost of the algorithm is at least $(1 + \alpha)(1 + \lambda)k$ as there are at least $k$ active machines. The cost of the optimal solution is $(1 + \alpha)(1 + \lambda) + k - 1$. For $k$ large enough compared to the prediction error of the length, we get that the ratio between the cost of any algorithm and the optimal solution is,

$$\frac{(1 + \alpha)(1 + \lambda)k}{(1 + \alpha)(1 + \lambda) + k - 1} \rightarrow (1 + \alpha)(1 + \lambda).$$ 

$\square$
7.4 Online Algorithm Using Lifetime and Load Vector Predictions

In this section we design an online algorithm with an extra knowledge of the future load. In particular, at each time \( t \) the algorithm is given the value \( v_{t'} \) for all \( t' \in [t, t + \mu) \). Without loss of generality (by refining the discretization of the time steps), we assume that at most one interval arrives at any time \( t \). Note that, even though the load vector reveals the time in which the next interval arrives (the next time step with a positive residual load), we cannot infer the interval’s length.

Algorithm 18 shows how to construct a single cover, similar in nature to the covers offline Algorithm 13 is creating, but in an online fashion. To this end, it takes as an input a load vector \( v' \) which might be an overestimate load prediction\(^2\) of the real load \( v \), i.e. \( v' \geq v \), and let \( \Delta = v' - v \geq 0 \). When an interval arrives, Algorithm 18 decides whether to accept it or reject it, with accepted intervals becoming part of the cover. A cover accepts an interval only if its rejection would not leave enough future load for the cover to reach its lower bound. Both load and length predictions are necessary to make this decision as the algorithm queries the residual load at every time step along the interval. Algorithm 19 then uses Algorithm 18 to create a set of covers and schedule all intervals to machines.

Algorithm 18: Online covering with overestimate predictions

1. Let \( v' \geq v \) be an overestimate load prediction given to the algorithm.
2. When interval \( I \) arrives at time \( t' \). Let \( v^a(t'), v^r(t') \) be the load vector of the intervals that arrived prior to time \( t' \), and were accepted or rejected respectively.
3. Accept \( I \) if there is a time \( t \in I \) such that,

\[
v'_t - v^r_t(t') \leq \begin{cases} 
1 & \text{in the uniform size case} \\
1/2 & \text{in the non-uniform size case}
\end{cases}
\]

Otherwise, reject \( I \).

\(^2\)This can be an overestimate due to prediction errors, however, the algorithm itself later produces overestimates by design (and not due to errors), which are used recursively in our analysis.
Lemma 7.4.1. Let \( v^a \) be the intervals that Algorithm 18 accepted. Then, for each time \( t \):

\[
\min\{(1 - \Delta t)^+, v_t\} \leq v^a_t \leq 2 \quad \text{for the uniform size case}
\]
\[
\min\{\left(\frac{1}{2} - \beta - \Delta t\right)^+, v_t\} \leq v^a_t \leq 1 \quad \text{for the non-uniform size case}
\]

Proof. We first show that the upper bounds hold, and then provide a proof for the lower bounds.

Upper bounds: Suppose there is a time \( t \) where \( v^a_t \) exceeds the upper bound. We will show that the algorithm cannot have accepted all intervals in \( v^a \). Consider the intervals \( I \) that belong to \( v^a \) and which are active at time \( t \), i.e., \( t \in I \). Let that set of intervals be denoted by \( A \), and its load with \( v'' \). Since all intervals in \( A \) are active at time \( t \), we can apply Lemma 1.4.2.

For the uniform case: Using Lemma 1.4.2 with \( \alpha = 2 \), we can see that there is at least one interval that observes load strictly more than 1 for its whole duration. Let \( I \) be the first such interval, and let \( s_I \) be its arrival time. The intervals in \( v^a \) are never rejected by the algorithm and hence, \( v''(s_I) \leq v_t - v^a_t \). Hence, upon arrival of \( I \), for any time \( t \in I \) \( v_t - v''(s_I) \geq v_t - v^a_t(s_I) \geq v^a_t > 1 \), and \( I \) will not be accepted by the algorithm.

For the non-uniform case: Similarly, using Lemma 1.4.2 with \( \alpha = 1 \), we see that there is at least one interval that observes load greater than \( \frac{1}{2} \) for its whole duration. Let \( I \) be the first such interval, and let \( s_I \) be its arrival time. The intervals in \( v^a \) are never rejected by the algorithm and hence, \( v''(s_I) \leq v_t - v^a_t \). Hence, upon arrival of \( I \), for any time \( t \in I \) \( v_t - v''(s_I) \geq v_t - v^a_t(s_I) \geq v^a_t > \frac{1}{2} \), and \( I \) will not be accepted by the algorithm.

Lower bounds: This proof is also by contradiction. Let \( t \) be a time for which \( v^a_t \) is smaller than the lower bound. Let \( I \) be the last arrived interval that is active at time \( t \) and which was rejected by the algorithm. Upon \( I \)'s arrival at time \( s_I \), the current load vector of rejected intervals at time \( t \) is \( v''(s_I) \) (does not yet include \( I \)), and so,
$v^i_t(s_I) + v^a_t + w_I = v_t$. Upon $I$’s arrival, the algorithm considers the quantity $v^i_t - v^i_t(s_I)$.

For the uniform case: The assumption that the lower bound is violated is translated to $v^a_t < \min \{(1 - \Delta_t)^+, v_t\}$. If $\Delta_t \geq 1$, this leads to an obvious contradiction, since it would imply that $v^a_t < 0$. Therefore, we focus on the case $\Delta_t < 1$, so $(1 - \Delta_t)^+ = 1 - \Delta_t$.

This gives: $v^i_t - v^i_t(s_I) = v^i_t - v_t + v^a_t + w_I = \Delta_t + v^a_t + w_I < \Delta_t + \min \{1 - \Delta_t, v_t\} + w_I = \min \{1, v_t + \Delta_t\} + w_I \leq 1 + w_I$. Since all intervals have size $1/g$ for some integer $g$, both the overestimated load $v^i_t$ and the rejected load $v^i_t(s_I)$ are multiples of $1/g$ (if $v^i_t$ is not, it can be rounded down to the closest $1/g$ multiple, since we know the extra load does not correspond to some interval). Therefore, since $v^i_t - v^i_t(s_I) < 1 + w_I$ and both loads are multiples of $1/g$, we have $v^i_t - v^i_t(s_I) \leq \frac{g-1}{g} + \frac{1}{g} = 1$. This shows that $I$ is accepted by the algorithm and leads to a contradiction proving that $v^a_t \geq \min \{(1 - \Delta_t)^+, v_t\}$ for each time $t$.

For the non-uniform case: We assumed that $v^a_t < \min \{(\frac{1}{2} - \beta - \Delta_t)^+, v_t\}$. If $\Delta_t \geq \frac{1}{2} - \beta$, this leads to an obvious contradiction, since it would imply that $v^a_t < 0$. Therefore, we focus on the case $\Delta_t < \frac{1}{2} - \beta$, so $(\frac{1}{2} - \beta - \Delta_t)^+ = \frac{1}{2} - \beta - \Delta_t$. We then have $v^i_t - v^i_t(s_I) = v^i_t - v_t + v^a_t + w_I = \Delta_t + v^a_t + w_I < \Delta_t + \min \{\frac{1}{2} - \beta - \Delta_t, v_t\} + w_I = \min \{\frac{1}{2} - \beta, v_t + \Delta_t\} + w_I \leq \frac{1}{2} - \beta + \beta = \frac{1}{2}$. Therefore, $I$ is accepted by the algorithm leading to a contradiction that proves that $v^a_t \geq \min \{(\frac{1}{2} - \beta - \Delta_t)^+, v_t\}$ for each time $t$. \qed

We now present the Online Covering Algorithm. This algorithm uses copies of Algorithm 18 to create a set of covers online and schedule all intervals to machines. In particular, when an interval $I$ arrives, the algorithm passes it as input to the first copy of Algorithm 18. If it gets rejected, it is then passed on to the second copy, followed by the remaining copies in increasing order until it gets accepted by a cover. Each copy $i$ of Algorithm 18 also receives as input an estimate $v^i$ of the load it will receive; $v^i$ might be an overestimate of the real load the copy will end up receiving.

**Lemma 7.4.2.** Let $v^a_{t,i}$ be the total load of accepted intervals of copy $i$ at time $t$. Let $v^w$ and $v^a$ be the load vector of the intervals that have size more than $\frac{1}{4}$ and at most $\frac{1}{4}$ respectively, and let $\beta_n$ be the largest size of intervals in $v^a$. For every time $t$, and $j$: 135
Algorithm 19: Online Covering Algorithm (with load vector predictions)

1. **In the non-uniform size case:** Schedule each interval with size greater than $\frac{1}{4}$ on a separate machine.

2. Run copies $i = 1, 2, \ldots$ of the online covering algorithm with overestimate predictions (Algorithm 18). The $i$th copy receives an overestimate of

$$v_i^n = \begin{cases} (v_t - (i - 1))^+ & \text{in the uniform size case} \\ (v_t - (i - 1) \cdot (\frac{1}{2} - \beta))^+ & \text{in the non-uniform size case} \end{cases}$$

The $i$th copy of the algorithm receives as its input all intervals that are rejected from copies $1, 2, \ldots, i - 1$.

3. Schedule all intervals accepted by copy $i$ using Algorithm 12 (First Fit).

$$\sum_{i=1}^{j} v_{i,i}^u \geq \min\{j, v_t\} \quad \text{for the uniform size case}$$

$$\sum_{i=1}^{j} v_{i,i}^n \geq \min\{(j \cdot (\frac{1}{2} - \beta_n) - v_t^w)^+, v_t^n\} \quad \text{for the non-uniform size case}$$

**Proof.** The proof is by induction on $j$. For the first copy, $j = 1$, we have $\Delta_t = 0$ for the uniform size case and $\Delta_t = v_t^w$ for the non-uniform size case, where $v_t^w$ is the total width of intervals of sizes larger than $1/4$. For $j = 1$ the claim follows by Lemma 7.4.1.

For time $t$, let $v_1, v_2, \ldots, v_{j-1}$ be the load accepted by copies $1, 2, \ldots, j - 1$.

For the uniform case: By our guarantee, $\sum_{i=1}^{j-1} v_i \geq \min\{j - 1, v_t\}$. We assume that $v_t > \sum_{i=1}^{j-1} v_i$, and $\sum_{i=1}^{j-1} v_i < j$, otherwise we are done. For the last copy the actual load at time $t$ is $v'' = v_t - \sum_{i=1}^{j-1} v_i$. Hence, $\Delta_t = \sum_{i=1}^{j-1} v_i - (j - 1)$ (which is greater than 0 by the induction hypothesis), and it is guaranteed to accept at least $\min\{(1 - \Delta_t)^+, v_t''\}$ load by Lemma 7.4.1. Therefore, the total load of accepted intervals of the first $j$ copies is at least:

136
\[
\sum_{i=1}^{j-1} v_i + \min\{(1 - \Delta_t)^+, v_t''\} = \sum_{i=1}^{j-1} v_i + \min\{(j - \sum_{i=1}^{j-1} v_i)^+, v_t - \sum_{i=1}^{j-1} v_i\} = \min\{j, v_t\}
\]

For the non-uniform case: Similarly to the uniform case, by our guarantee, 
\[
\sum_{i=1}^{j-1} v_i \geq \min\{((j - 1) \cdot \left(\frac{1}{2} - \beta_n\right) - v_t^w)^+, v_t^n\}. \quad \text{We assume that } v_t^n > \sum_{i=1}^{j-1} v_i, \text{ and } \\
\sum_{i=1}^{j-1} v_i < j \cdot \left(\frac{1}{2} - \beta_n\right) - v_t^w, \text{ otherwise we are done. For the last copy the actual load} \\
at time \(t\) is \(v'' = v_t^n - \sum_{i=1}^{j-1} v_i\). \text{ Hence, } \Delta_t = \sum_{i=1}^{j-1} v_i - (j - 1) \cdot \left(\frac{1}{2} - \beta_n\right) + v_t^w, \text{ and it is guaranteed to accept at least } \min\{((\frac{1}{2} - \beta_n - \Delta_t)^+, v_t''\} \text{ load by Lemma 7.4.1. Therefore,} \\
the total load of accepted intervals of the first \(j\) copies is at least:
\[
\sum_{i=1}^{j-1} v_i + \min\{\left(\frac{1}{2} - \beta_n - \Delta_t\right)^+, v_t''\} = \sum_{i=1}^{j-1} v_i + \min\{\left(j \cdot \left(\frac{1}{2} - \beta_n\right) - v_t^w - \sum_{i=1}^{j-1} v_i\right)^+, v_t^n - \sum_{i=1}^{j-1} v_i\} \\
\geq \min\{\left(j \cdot \left(\frac{1}{2} - \beta_n\right) - v_t^w\right)^+, v_t^n\}
\]
\]

\[\square\]

**Theorem 7.4.3.** Given an instance of Clairvoyant Bin Packing with load vector predictions, the total cost of Algorithm 19 is at most \(2 \cdot \|v\|_1\), for the uniform size case, \(8 \cdot \|v\|_1\) in the non-uniform case. If \(\beta \leq \frac{1}{4}\) the total cost is at most \(\sum_{t} \lceil \frac{2v_t}{1-2\beta} \rceil\).

**Proof.** *For the uniform case:* From Lemma 7.4.2, for each time \(t\), the first \(v_t\) copies will accept all intervals that are active at time \(t\), and according to Lemma 7.1.2, each copy can schedule its intervals using 2 machines. As a result, the total cost of the algorithm is \(\sum_{t} 2v_t = 2\|v\|_1\), proving 2-competitiveness.

*For the non-uniform case:* If \(\beta \leq \frac{1}{4}\), then \(v_t^n = v_t\) and \(v_t^w = 0\). Then, by Lemma 7.4.2, the first \(\lceil \frac{v_t}{\frac{1}{2} - \beta} \rceil\) copies will accept all intervals that are active at each time \(t\). By Lemma 7.4.1, each of them is paying 1, so the total cost is at most \(\sum_{t} \lceil \frac{2v_t}{1-2\beta} \rceil\).

If \(\beta > \frac{1}{4}\), let \(W_t\) be the number of intervals with size larger than \(\frac{1}{4}\) that are active at time \(t\). Since the algorithm opens a separate machine of unit size for each of them, it pays cost \(W_t\) at each time \(t\). The cost at time \(t\) to schedule intervals of size smaller than \(\frac{1}{4}\) is at most \(\lceil \frac{v_t^w + v_t^n}{\frac{1}{2} - \frac{1}{4}} \rceil = 4v_t\). Thus, the total cost is at most: 
\[W_t + \lceil 4v_t \rceil =
\]
\[ W_t + 4v_t \leq [4v_t^w + 4v_t] \leq 8\lfloor v_t \rfloor. \]
Chapter 8

The Generalized Multistage $d$-Knapsack Problem

In this section we present approximation algorithms for $d$-GMK and Submodular $d$-GMK. We first present a reduction to $d$-MKCP, providing an algorithm for instances with a fixed time horizon. Next, the algorithm is extended via an additional reduction to solve instance with general time horizon as well. An equivalent reduction and extension is provided for Submodular $d$-GMK as well. Finally, we accompany these results with lower bounds for $d$-GMK.

8.1 Approximation Scheme for $d$-GMK

In this section we derive the approximation scheme for $d$-GMK with bounded profit-cost ratio. In Section 8.1.1 we show how a PTAS for instances with bounded time horizon can be obtain via a reduction to a variant of $d$-MKCP. Subsequently, in Section 8.1.2 we show how the algorithm for bounded time horizon can be used to approximate general instance.
8.1.1 Bounded Time Horizon

In this section we provide a reduction from an instance of \(q\)-GMK to a generalization of \(d\)-MKCP (for specific values of \(d\) and \(q\)). The generalization was presented in [37] and is called \(d\)-MKCP With A Matroid Constraint (\(d\)-MKCP+) and is defined by a tuple \((I, \mathcal{K}, p, \mathcal{I})\), where \((I, \mathcal{K}, p)\) forms an instance of \(d\)-MKCP. Also, the set \(\mathcal{I} \subseteq 2^I\) defines a matroid\(^1\) constraint. A feasible solution for \(d\)-MKCP+ is a set \(S \in \mathcal{I}\) and a tuple of feasible assignments \(A\) (w.r.t \(\mathcal{K}\)) of \(S\). The goal is to find a feasible solution which maximizes \(\sum_{i \in S} p(i)\). The following definition presents the construction of the reduction.

**Definition 8.1.1.** Let \(Q = ((P_t)_{t=1}^T, g^+, g^-, c^+, c^-)\) be an instance of \(d\)-GMK, where \(P_t = (I, \mathcal{K}_t, p_t)\) and \(\mathcal{K}_t = (K_{t,j})_{j=1}^{d_t}\). Define \(R(Q) = (E, \tilde{\mathcal{K}}, \tilde{p}, \mathcal{I})\) where

- \(E = I \times 2^{[T]}\)
- \(\mathcal{I} = \{S \subseteq E \mid \forall i \in I : |S \cap (\{i\} \times 2^{[T]})| \leq 1\}\)
- For \(t \in [T], j \leq d_t\) set MKC \(\tilde{K}_{t,j} = (\tilde{w}_{t,j}, B_{t,j}, W_{t,j})\) over \(E\), where \(K_{t,j} = (w_{t,j}, B_{t,j}, W_{t,j})\) and

\[
\tilde{w}_{t,j}((i, D)) = \begin{cases} w_{t,j}(i) & t \in D \\ 0 & \text{otherwise} \end{cases}
\]

- For \(t = 1, \ldots, T, d_t < j \leq d\) set MKC \(\tilde{K}_{t,j} = (w_0, \{b\}, W_0)\) over \(E\), where \(w_0 : 2^E \to \{0\}\), \(W_0(b) = 0\) and \(b\) is an arbitrary bin (object).
- \(\tilde{\mathcal{K}} = \left(\tilde{K}_{t,j}\right)_{t \in [T], j \in [d]}\).
- The objective function \(\tilde{p}\) is defined as follows.

\[
\tilde{p}(S) = \sum_{(i,D) \in S} \left(\sum_{t \in D} p_t(i) + \sum_{t+1 \notin D} g^+_{i,t} - \sum_{t \notin D} g^-_{i,t} - \sum_{t+1 \notin D} c^+_{i,t} + \sum_{t-1 \notin D} c^-_{i,t}\right)
\]

\(^1\)A formal definition for matroid can be found in [80]
Each element \((i, D) \in E\) states the subset of stages in which item \(i\) is assigned. I.e., \(i\) is only assigned in stages \(t \in D\). Thus any solution should include at most one element \((i, D)\) for each \(i \in I\). This constraint is fully captured by the partition matroid constraint defined by the set of independent sets \(\mathcal{I}\). Finally, if an element \((i, D)\) is selected, we must assign \(i\) in each MKC \(K_{t,j}\) for \(j \in [d_t], t \in D\). This is captured by the weight function \(\tilde{w}\), as an element \((i, D)\) weighs \(w_{t,j}(i)\) if and only if \(t \in D\) (otherwise its weight is zero and it can be assigned for “free”).

**Lemma 8.1.1.** For any \(d\)-GMK instance \(Q\) with time horizon \(T\), it holds that \(R(Q)\) is a \(dT\)-MKCP\(^+\) instance.

**Proof.** Let \(Q = (\mathcal{P}_t)_{t=1}^{T}, g^+, g^-, c^+, c^-\) be an instance of \(d\)-GMK, where \(\mathcal{P}_t = (I, K_t, p_t)\) and \(K_t = (K_{t,j})_{j=1}^{d_t}\). Also, let \(R(Q) = (E, \tilde{K}, \tilde{\mathcal{P}}, \tilde{I})\) be the reduced instance of \(Q\) as defined in Definition 8.1.1. It is easy to see that the set \(\tilde{I}\) is the independent sets of a partition matroid, as for each item \(i\) at most one element \((i, D)\) can be chosen. Thus, \(\tilde{I}\) is the family of independent sets of a matroid as required.

Next, \(\tilde{K}\) defines a tuple of MKCs, so all that is left to prove is that \(\tilde{\mathcal{P}}\) is non-negative and modular. For each element \((i, D) \in E\) we can define a fixed value

\[
v((i, D)) = \sum_{t \in D} p_t(i) + \sum_{t \in D: t-1 \in D} g_{i,t}^+ + \sum_{t \notin D: t-1 \notin D} g_{i,t}^- - \sum_{t \in D: t-1 \notin D} c_{i,t}^+ + \sum_{t \in D: t+1 \notin D} c_{i,t}^-
\]

It immediately follows that \(\tilde{\mathcal{P}}(S) = \sum_{e \in S} v(e)\) and that \(\tilde{\mathcal{P}}\) is modular. As stated in Section 1.5, elements with negative values are discarded in advance such that \(\tilde{\mathcal{P}}\) is also non-negative. \(\square\)

**Lemma 8.1.2.** Let \(Q\) be an instance of \(d\)-GMK with time horizon \(T\). For any feasible solution \((S_t, A_t)_{t=1}^{T}\) of \(Q\) there exists a feasible solution \((S, (\tilde{A}_{t,j})_{t \in [T], j \in [d]}\) of \(R(Q)\) such that \(f_Q((S_t)_{t=1}^{T}) = \tilde{\mathcal{P}}(S)\).

**Proof.** Let \(Q = (\mathcal{P}_t)_{t=1}^{T}, g^+, g^-, c^+, c^-\) be an instance of \(d\)-GMK, where \(\mathcal{P}_t = (I, K_t, p_t)\) and \(K_t = (K_{t,j})_{j=1}^{d_t}\). Also, let \(R(Q) = (E, \tilde{K}, \tilde{\mathcal{P}}, \tilde{I})\) be the reduced instance of \(Q\), where \(\tilde{\mathcal{K}} = (\tilde{K}_{t,j})_{t \in [T], j \in [d]}\) and \(\tilde{K}_{t,j} = (\tilde{w}, B_{t,j}, W_{t,j})\) (see Definition 8.1.1). Consider some
feasible solution \((S_t, A_t)_{t=1}^T\) for \(Q\), where \(A_t = (A_{t,j})_{j=1}^d\). In the following we define a solution \(\left(S, \left(\tilde{A}_{t,j}\right)_{t \in [T], j \in [d]}\right)\) for \(R(Q)\). Let

\[
S = \{(i, D) \mid i \in I, D = \{t \in [T] \mid i \in S_t\}\}
\]

It can be easily verified that \(S \in \mathcal{I}\). The value of the subset \(S\) is

\[
\tilde{p}(S) = \sum_{(i,D) \in S} \left( \sum_{t \in D} p_t(i) + \sum_{t \in D: t-1 \notin D} g_{i,t}^+ + \sum_{t \notin D: t-1 \notin D} g_{i,t}^- - \sum_{t \in D: t-1 \notin D} c_{i,t}^+ - \sum_{t \notin D: t+1 \notin D} c_{i,t}^- \right) = f_\mathcal{Q}\left(\left(S_t\right)_{t=1}^T\right).
\]

Next, for each \(t \in [T], j \in [d]\) we present an assignment \(\tilde{A}_{t,j}\) of \(S\). Consider the following two cases:

1. If \(j > d_t\), recall \(\tilde{K}_{t,j} = (w_0, \{b\}, W_0)\) where \(w_0(i, D) = 0\) for all \((i, D) \in E\) and \(W_0(b) = 0\). We define \(\tilde{A}_{t,j}\) by \(\tilde{A}_{t,j}(b) = S\). It thus holds that \(w_0(\tilde{A}_{t,j}(b)) = 0 = W_0\). That is, \(\tilde{A}_{t,j}\) is feasible.

2. If \(j \leq d_t\), let \(b^* \in B_{t,j}\) be some unique bin in \(B_{t,j}\) and define assignment \(\tilde{A}_{t,j}: B_{t,j} \rightarrow 2^E\) by

\[
\tilde{A}_{t,j}(b) = (A_{t,j}(b) \times 2^{[T]}) \cap S
\]

\[
\tilde{A}_{t,j}(b^*) = ((A_{t,j}(b^*) \times 2^{[T]}) \cap S) \cup \{(i, D) \in S \mid t \not\in D\}
\]

The assignment \(\tilde{A}_{t,j}\) is a feasible assignment w.r.t \(\tilde{K}_{t,j}\) since for each bin \(b \in B_{t,j}\) it holds that

\[
\sum_{(i,D) \in \tilde{A}_{t,j}(b)} \tilde{w}_{t,j}((i,D)) = \sum_{i \in A_{t,j}(b)} w_{t,j}(i) \leq W_{t,j}(b)
\]

143
Let \((i, D) \in S\). If \(i \in S_t\) there is \(b \in B_{t,j}\) such that \(i \in A_{t,j}(b)\), hence \((i, D) \in \tilde{A}_{t,j}(b)\) by (8.1). If \(i \notin S_t\) then \(t \notin D\) and thus \((i, D) \in \tilde{A}_{t,j}(b^*)\). Overall, we have \(S \subseteq \bigcup_{b \in B_{t,j}} \tilde{A}_{t,j}(b)\). By (8.1) it follows that \(S \supseteq \bigcup_{b \in B_{t,j}} \tilde{A}_{t,j}(b)\) as well, thus \(S = \bigcup_{b \in B_{t,j}} \tilde{A}_{t,j}(b)\). I.e., \(\tilde{A}_{t,j}\) is an assignment of \(S\).

Note that the assignments can be constructed in polynomial time. We can conclude that \((S, (\tilde{A}_{t,j})_{t \in [T], j \in [d]})\) is a feasible solution for \(R(Q)\), and its value is \(f_Q((S_t)_{t=1}^T)\).

**Lemma 8.1.3.** Let \(Q\) be an instance of \(d\)-GMK (with arbitrary time horizon \(T\)). For any feasible solution \((S, (\tilde{A}_{t,j})_{t \in [T], j \in [d]})\) for \(R(Q)\) a feasible solution \((S_t, A_t)_{t=1}^T\) for \(Q\) such that \(f_Q((S_t)_{t=1}^T) = \tilde{f}(S)\) can be constructed in polynomial time.

**Proof.** Let \(Q = (P_t)_{t=1}^T, g^+, g^-, c^+, c^-\) be an instance of \(d\)-GMK, where \(P_t = (I, K_t, p_t)\) and \(K_t = (K_{t,j})_{j=1}^{d_t}\). Also, let \(R(Q) = (E, \tilde{K}, \tilde{p}, \tilde{T})\) be the reduced instance of \(Q\). and \((S, (\tilde{A}_{t,j})_{t \in [T], j \in [d]})\) be a feasible solution for \(R(Q)\). We define the solution \((S_t, A_t)_{t=1}^T\) for \(Q\) as follows. For every stage \(t\) we set \(S_t = \{i \in I \mid \exists (i, D) \in S : t \in D\}\). For every \(t = 1, \ldots, T\), \(j = 1, \ldots, d_t\) and bin \(b \in B_{t,j}\) (the set of bins in MKC \(K_{t,j}\)) let \(A_{t,j}(b) = \{i \in I \mid \exists (i, D) \in \tilde{A}_{t,j}(b) : t \in D\}\). Observe that the sets \((S_t)_{t=1}^T\) and assignments \((A_{t,j})_{t \in [T], j \in [d]}\) can be constructed in polynomial time as at most \(|I|\) elements can be chosen due to the matroid constraint. The assignment \(A_{t,j}\) is an assignment of \(S_t\) since

\[
S_t = \{i \in I \mid \exists (i, D) \in S : t \in D\} = \bigcup_{b \in B_{t,j}} \{i \in I \mid (i, D) \in \tilde{A}_{t,j}(b) : t \in D\} = \bigcup_{b \in B_{t,j}} A_{t,j}(b),
\]

where the second equality follows the feasibility of the solution for \(R(Q)\). In addition, \(A_{t,j}\) is a feasible assignment for MKC \(K_{t,j}\) since for every bin \(b \in B_{t,j}\) it holds that

\[
\sum_{i \in A_{t,j}(b)} w_{t,j}(i) = \sum_{(i, D) \in \tilde{A}_{t,j}(b) : t \in D} \tilde{w}_{t,j}((i, D)) \leq \sum_{(i, D) \in \tilde{A}_{t,j}(b)} \tilde{w}_{t,j}((i, D)) \leq W_{t,j}(b)
\]

Thus \((S_t, A_t)_{t=1}^T\) is a feasible solution for \(Q\).
Lastly, consider the value of the solution for \( Q \). It holds that

\[
f_Q \left( \left( S_t \right)_{t=1}^T \right) = \sum_{t=1}^{T} \sum_{i \in S_t} p_t(i) + \sum_{t=2}^{T} \left( \sum_{i \in S_{t-1} \cap S_t} g_{i,t}^+ + \sum_{i \notin S_{t-1} \cup S_t} g_{i,t}^- \right) - \sum_{t=1}^{T} \left( \sum_{i \in S_t \setminus S_{t-1}} c_{i,t}^+ + \sum_{i \in S_t \setminus S_{t+1}} c_{i,t}^- \right) = \sum_{(i,D) \in S} \left( \sum_{t \in D} p_t(i) + \sum_{t \in D: t-1 \in D} g_{i,t}^+ + \sum_{t \notin D: t-1 \notin D} g_{i,t}^- - \sum_{t \in D: t-1 \notin D} c_{i,t}^+ - \sum_{t \in D: t+1 \notin D} c_{i,t}^- \right) = \tilde{p}(S)
\]

For any \( d \)-GMK instance with a fixed time horizon \( T \), the reduction \( R(Q) \) can be constructed in polynomial (as \( |E| = |I| \cdot 2^{O(T)} \)). The next corollary follows from this observation and lemmas 8.1.2 and 8.1.3.

**Corollary 8.1.4.** For any fixed \( T \in \mathbb{N} \), there exists an approximation factor preserving reduction from \( d \)-GMK with a time horizon bounded by \( T \) to \( dT \)-MKCP+.

In [37] a PTAS for \( d \)-MKCP+ is presented. Thus, the next lemma follows from the above corollary.

**Lemma 8.1.5.** For any fixed \( T \in \mathbb{N} \) there exists a randomized PTAS for \( d \)-GMK with a time horizon bounded by \( T \).

### 8.1.2 General Time Horizon

In this section we present an algorithm for \( d \)-GMK with a general time horizon \( T \). This is done by cutting the time horizon at several stages into sub-instances. Each sub-instance is optimized independently and then the solutions are combined to create a solution for the complete instance. A somewhat similar technique was used in [8]. However, they considered a model without change costs which is much simpler. Our analysis is more delicate as it requires local consideration of the assignment of each item to ensure that any additional costs charged are covered by profit and gains earned.
Given an instance \( Q = ((P_t)_{t=1}^T, g^+, g^-, c^+, c^-) \) we define a sub-instance for the sub-range \([t_1, t_2]\), denoted throughout this section by \(((P_t)_{t=t_1}^{t_2}, g^+, g^-, c^+, c^-)\) without shifting or truncating the gain and change costs vectors. For example, for \( t \in [t_1, t_2] \) gain \( g_{i,t}^+ \) is earned for assigning item \( i \) in stages \( t \) and \( t + 1 \). Also, observe that stages \( t_1 - 1 \) and \( t_2 + 1 \) are outside the scope of the instance. Thus, when evaluating a solution for the sub-instance it is assumed that \( S_{t_1-1} = S_{t_2+1} = \emptyset \).

Given an integer \( T \in \mathbb{N} \), a set of cut points \( U = \{u_0, ..., u_k\} \) of \( T \) is a set of integers such that for every \( j = 0, ..., k - 1 \) it holds that \( u_j < u_{j+1} \) and \( 1 = u_0 < u_k = T + 1 \).

**Definition 8.1.2.** Let \( Q = ((P_t)_{t=1}^T, g^+, g^-, c^+, c^-) \) be an instance of \( d \)-GMK, where \( P_t = (I, K_t, p_t) \). Also, let \( U = \{u_0, ..., u_k\} \) be a set of cut points. The tuple of \( d \)-GMK instances \( Q_U = ((P_t)_{t=u_j}^{u_{j+1}-1}, g^+, g^-, c^+, c^-)_{j=0}^{k-1} \) is defined as the cut instances of \( Q \) w.r.t \( U \).

**Definition 8.1.3.** Let \( Q \) be an instance of \( d \)-GMK, \( U = \{u_0, ..., u_k\} \) be a set of cut points and \( Q_U \) be the respective cut instances. Also, let \( ((S_t, A_t)_{t=u_j}^{u_{j+1}-1})_{j=0}^{k-1} \) be a tuple of feasible solutions for the tuple of cut instances \( Q_U \). Then, the solution \((S_t, A_t)_{t=1}^T\) for \( Q \) is called a cut solution.

The next corollary elaborates on the relationship between a cut solution and the cut instance solutions from which it is constructed.

**Corollary 8.1.6.** Given any \( d \)-GMK instance \( Q \), cut points \( U \), cut instances \( Q_U \) and feasible solutions for the cut instances, the respective cut solution is a feasible solution for \( Q \), and its value is at least the sum of values of the solutions for the cut instances.

**Proof.** Let \( Q \) be an instance of \( d \)-GMK, \( U \) be a set of cut points. Also, let \( Q_U = (q_j)_{j=0}^{k-1} = ((P_t)_{t=u_j}^{u_{j+1}-1}, g^+, g^-, c^+, c^-)_{j=0}^{k-1} \) be the corresponding tuple of cut instances, and \((S_t, A_t)_{t=u_j}^{u_{j+1}-1})_{j=0}^{k-1} \) be a tuple of feasible solutions for the cut instances.

We define the solution \((S_t, A_t)_{t=1}^T\) for \( Q \). It is easy to see that the assignments \( A_t \)
to $K_t$ are all feasible assignments of $S_t$. In addition, it holds that

$$f_Q((S_t)_{t=1}^T) = \sum_{t=1}^T p_t(S_t) + \sum_{t=2}^T \left( \sum_{i \in S_{t-1} \cap S_t} g^+_{i,t} + \sum_{i \notin S_{t-1} \cup S_t} g^-_{i,t} \right) - \sum_{t=1}^T \left( \sum_{i \in S_t \backslash S_{t+1}} c^-_{i,t} + \sum_{i \in S_t \backslash S_{t-1}} c^+_{i,t} \right) \geq$$

$$\sum_{j=0}^{k-1} \left( \sum_{t=1}^{u_{j+1}+1} - \sum_{t=1}^{u_j+1} \left( \sum_{i \in S_{t-1} \cap S_t} c^-_{i,t} + \sum_{i \notin S_{t-1} \cup S_t} c^+_{i,t} \right) - \sum_{i \in S_{u_{j+1}-1}} c^-_{i,u_{j+1}} - \sum_{i \in S_{u_j}} c^+_{i,u_j} \right)$$

where $f_{q_j}$ is the objective functions of cut instance $q_j$. This proves that a cut solution has a higher value than the sum of solutions for cut instance from which it was created.

We are now ready to present the algorithm for $d$-GMK with general time horizon length.

**Algorithm 20: General Time Horizon**

**Input:** $0 < \epsilon < \frac{1}{4}$, $\phi \geq 1$, a $d$-GMK instance $Q$ with time horizon $T$ such that $\phi_Q \leq \phi$, and $\alpha$-approximation algorithm $A$ for $d$-GMK with time horizon $T \leq \frac{2\phi}{\epsilon^2}$.

1. Set $\mu = \frac{\epsilon^2}{\phi}$.

2. for $j = 1, \ldots, \frac{1}{\frac{\epsilon}{\phi}}$ do

3. Set $U_j = \left\{ \frac{a}{\mu} + j - 1 \mid a \in \mathbb{N}, \ a \geq 1, \ \frac{a}{\mu} + j - 1 \leq T - \frac{1}{\mu} \right\} \cup \{1, T + 1\}$.

4. Find a solution for each cut instance in $Q_{U_j}$ using algorithm $A$ and set $S_j$ as the respective cut solution.

5. Return the solution $S_j$ which maximizes the objective function $f_Q$.

Before analysing the algorithm we present several definitions and lemmas that are essential for the proof. First, we start by reformulating the solution. Instead of de-
scribing the assignment of items by the tuple \((S_t)_{t=1}^T\), we define a new set of elements 
\(E = I \times [T] \times [T]\), where each element \((i, t_1, t_2) \in E\) states that item \(i\) is assigned in the interval \([t_1, t_2]\). Given a feasible solution \((S_t, A_t)_{t=1}^T\) for \(Q\) we denote the representation of \((S_t)_{t=1}^T\) as a subset of \(E\) by \(E((S_t)_{t=1}^T)\) and it is equal to

\[
E((S_t)_{t=1}^T) = \{(i, t_1, t_2) \in E \mid \forall t \in [t_1, t_2] : i \in S_t \text{ and } i \notin S_{t-1} \cup S_{t+1}\}.
\]

If \(\tilde{S} = E((S_t)_{t=1}^T)\), we define the reverse mapping as \(\tilde{S}(t) = \{i \in I \mid \exists (i, t_1, t_2) \in S : t \in [t_1, t_2]\} = S_t\). Now, we can define a solution for \(d\)-GMK using our new representation as \((\tilde{S}, A_t)_{t=1}^T\).

**Definition 8.1.4.** The value, \(v(e)\), of element \(e = (i, t_1, t_2)\) is defined as the total value earned from assigning \(i\) in the range \([t_1, t_2]\) minus the change costs charge for assigning and discarding it. Formally,

\[
v(e) = \sum_{t=t_1}^{t_2} p_t(i) + \sum_{t=t_1+1}^{t_2} g_t^+ - c_{i,t_1}^+ - c_{i,t_2}^{-}.
\]

The value of solution \(\tilde{S} \subseteq E\) for \(d\)-GMK instance \(Q\) is \(\sum_{e \in \tilde{S}} v(e) + \sum_{t=2}^T \sum_{i \notin \tilde{S}(t-1) \cup \tilde{S}(t)} g_t^+\) and it is equal to \(f_Q((\tilde{S}(t))_{t=1}^T)\).

In Algorithm 20 we consider a solution for a tuple of cut instances created by cutting an instance at a set of cut points. Here we consider the opposite action, the effect of cutting a solution at these cut points. We start by considering a single cut point.

**Definition 8.1.5.** Given an element \(e = (i, t_1, t_2)\) and a cut point \(u \in (t_1, t_2]\) we define the outcome of cutting \(e\) at \(u\) as the set of intervals \(u(e) = \{(i, t_1, u-1), (i, u, t_2]\}\). Also, the loss caused by cutting \(e\) at \(u\) is defined as the difference between the value of \(e\) and the sum of value of elements in \(u(e)\). It is denoted by \(\ell(e, u)\) and is equal to

\[
\ell(e, u) = v(e) - \sum_{e' \in u(e)} v(e') = g_{i,u}^+ + c_{i,u}^+ + c_{i,u-1}^{-}.
\]

We can similarly extend the definition to include more than one cut point as follows.
**Definition 8.1.6.** Given a set of cut points $U = \{u_0, u_1, \ldots, u_k\}$ and an element $e = (i, t_1, t_2)$ define $U(e)$ as the set of elements created by cutting the $e$ at all cut points in $U$. Formally,

$$U ((i, t_1, t_2)) = \{(i, \max\{t_1, u_{r-1}\}, \min\{u_r - 1, t_2\}) \mid r \in [k] \text{ and } [u_{r-1}, u_r - 1] \cap [t_1, t_2] \neq \emptyset\}$$

Consider the following example as a demonstration of the above definition. If $e = (i, t_1, t_2)$ and $(t_1, t_2) \cap U_j = \{u_2, u_3\}$, then $U_j(e) = \{(i, t_1, u_2 - 1), (i, u_2, u_3 - 1), (i, u_3, t_2)\}$.

The definition of loss caused by cutting an element can be extended to a set of cut points $U$. If element $e = (i, t_1, t_2)$ is cut by a of cut points $U$ the loss is

$$\ell(e, U) = v(e) - \sum_{e' \in U(e)} v(e') = \sum_{u \in U \cap (t_1, t_2)} (g^+_i + c^+_i + c^-_{i,u-1}) = \sum_{u \in U \cap (t_1, t_2)} \ell(e, u) \quad (8.3)$$

since only gains $g^+$ are lost due to cutting as well as change cost for splitting an assignment into two intervals. This means that even if an element is cut multiple times, the loss due to each cut point can be considered separately.

**Lemma 8.1.7.** Let $0 < \epsilon < \frac{1}{4}$, $\phi \geq 1$ and $A$ be an $\alpha$-approximation algorithm for $d$-GMK with time horizon $T \leq \frac{2\phi}{T^*}$. Also, let $Q$ be an instance of $d$-GMK such that $\phi_Q \leq \phi$. Algorithm 20 approximates $Q$ within a factor of $(1-\epsilon)\alpha$.

**Proof.** Let $Q = ((P_t)_{t=1}^T, g^+, g^-, c^+, c^-)$ be an instance of $d$-GMK with time horizon $T$ and profit-cost ratio $\phi_Q \leq \phi$. We assume for simplicity $\phi$ is integral. Let $0 < \epsilon < \frac{1}{4}$ and $\mu = \frac{\epsilon^2}{\phi}$. Also, let $A$ be an $\alpha$-approximation algorithm for $d$-GMK with time horizon $T' \leq \frac{2\phi}{T^*} = \frac{2}{\mu}$. Note, if $T \leq \frac{2}{\mu}$, the cut points set $U_0$ is an empty set, and in this case $A$ returns an $\alpha$-approximation solution for $Q$ as required.

Let $U_j = \{u_1^j, \ldots, u_{k_j}^j\}$ for $j = 1, \ldots, \frac{1}{\mu}$. We show that there exists a set of cut points $U_j$ and a tuple of solutions $(S_t, A_t)_{t=u_{r+1}}^{u_{r+1}-1}$ for each cut instance in $Q_{U_j} = (q_{r}^j)_{r=0}^{k_j-1}$, such that the sum of values of the solutions, $\sum_{r=0}^{k_j-1} f_{q_r^j}((S_t)_{t=u_r}^{u_{r+1}-1})$, is sufficiently large. From Corollary 8.1.6 it follows that the value of a cut solution is larger than the sum of its parts (due to lost gains and change costs saved if an item is assigned in adjacent
instances). Thus, this also proves that the maximum cut solution found is sufficiently large as well.

Let \((S_t^*, A_t^*)_{t=1}^T\) be an optimal solution for \(Q\), and let \(\tilde{S}^* = E((S_t^*)_{t=1}^T)\). We partition \(\tilde{S}^*\) into two subsets by the length of the interval they describe. Formally, 
\[
X = \{(i, t_1, t_2) \in \tilde{S}^* \mid t_2 - t_1 < \frac{s}{2}\}
\]
and 
\[
Y = \tilde{S}^* \setminus X.
\]
So \(X\) contains short intervals, and \(Y\) contains long intervals.

Define \(\tilde{S}_j\) as the subset of elements longer than \(\phi\) in \(\cup_{e \in Y} U_j(e)\) as well as short elements \(e \in X\) that are not cut by \(U_j\). I.e.,
\[
\tilde{S}_j = \{e \in X \mid U_j(e) = \{e\}\} \cup \bigcup_{e \in Y} \{(i, t_1, t_2) \in U_j(e) \mid t_2 - t_1 \geq \phi\}
\]
At each stage \(t \in [T]\) it holds that \(\tilde{S}_j(t) \subseteq S_t^*\). Thus there exists a tuple of assignments, denoted by \(A_t^j\), such that \((\tilde{S}_j, A_t^j)_{t=1}^T\) is a feasible solution for \(Q\). We partition set \(\tilde{S}_j\) as follows. Let 
\[
\tilde{S}_{j,r} = \{(i, t_1, t_2) \in \tilde{S}_j \mid [t_1, t_2] \subseteq [u_r^j, u_{r+1}^j - 1]\}
\]
It holds that \(\tilde{S}_j = \bigcup_{r=0}^{k_j-1} \tilde{S}_{j,r}\) as each element \((i, t_1, t_2)\) is contained in exactly one interval \([u_r^j, u_{r+1}^j - 1]\).

Thus 
\[
((\tilde{S}_{j,r}, A_t^j)_{t=u_r^j}^{u_{r+1}^j-1})_{r=0}^{k_j-1}
\]
is a tuple of feasible solutions for the cut instances in \(Q_{U_j}\) such that \((\tilde{S}_{j,r}, A_t^j)_{t=u_r^j}^{u_{r+1}^j-1}\) is a solution for the \(r\)-th instance. The value of all elements in the defined solutions for the cut instances is
\[
\sum_{r=0}^{k_j-1} \sum_{e \in \tilde{S}_{j,r}} v(e) = \sum_{e \in \tilde{S}_j} v(e) = \sum_{e \in X: U_j(e) = \{e\}} v(e) + \sum_{e \in Y} \sum_{(i, t_1, t_2) \in U_j(e): t_2 - t_1 \geq \phi} v(e')
\]
Thus, after including the value of gains \(g^-\) earned by solutions \(\tilde{S}_{j,r}\), we can bound the total value of optimal solutions for all cut instances \(Q_{U_j}\) by
\[
\sum_{e \in X: U_j(e) = \{e\}} v(e) + \sum_{e \in Y} \sum_{e'=(i, t_1, t_2) \in U_j(e): t_2 - t_1 \geq \phi} v(e') + \sum_{t \in [u_r^j, u_{r+1}^j-1]: u_r^j \in U_j} \sum_{i \in S_{j,r}(t)} g_{i,t}^{-} \quad (8.4)
\]
We define \(B\) as the total sum of values of optimal solutions for the cut instances
\((Q_{U_j})_{j=1}^{\frac{1}{\mu}}\). By utilizing Equation (8.4) we can bound \(B\) as follows.

\[
B \geq \sum_{j=1}^{\frac{1}{\mu}} \sum_{r=0}^{k_j-1} \left( \sum_{e \in \tilde{S}_{j,r}} v(e) + \sum_{t \in [u_j^i+1,u_j^i+1]} \sum_{i \in I_j \cap (t-1) \cup \tilde{S}_{j,r}(t)} g_{i,t} \right)
\]

\[
= \sum_{j=1}^{\frac{1}{\mu}} \sum_{e \in S_j} v(e) + \sum_{j=1}^{\frac{1}{\mu}} \sum_{t \in [2,T] \setminus \bigcup_{i \in I_j} (t-1) \cup \tilde{S}_{j,t}(t)} g_{i,t}
\]

\[
= \sum_{e \in X} \sum_{j=1}^{\frac{1}{\mu}} \sum_{e \in S_j} v(e) + \sum_{j=1}^{\frac{1}{\mu}} \sum_{e \in Y} \sum_{e' \in \bigcup_{i \in I_j} (t-1) \cup \tilde{S}_{j,t}(t)} v(e')
\]

\[
+ \sum_{j=1}^{\frac{1}{\mu}} \sum_{e \in X} \sum_{e' \in \bigcup_{i \in I_j} (t-1) \cup \tilde{S}_{j,t}(t)} g_{i,t}
\]

We bound the value of each of the three terms separately by comparing it to the value of the optimal solution.

Consider the first term, value earned from short elements, i.e., elements \(e = (i, t_1, t_2) \in X\). It holds that \(e \in \tilde{S}_j\) if and only if \(U_j(e) = \{e\}\) which means that \((t_1, t_2) \cap U_j = \emptyset\). Since for every \(j_1 \neq j_2\) it holds that \(U_{j_1} \cap U_{j_2} = \{1, T + 1\}\) and since there are \(\frac{1}{\mu}\) sets of cut point, for each element \(e \in X\) it holds that \(e \in \tilde{S}_j\) for at least \(\frac{1}{\mu} - \frac{\phi}{\epsilon}\) values of \(j \in [\frac{1}{\mu}]\). Thus,

\[
\sum_{e \in X} \sum_{j=1}^{\frac{1}{\mu}} \sum_{e \in S_j} v(e) \geq \left(\frac{1}{\mu} - \frac{\phi}{\epsilon}\right) \sum_{e \in X} v(e) = \frac{1}{\mu} (1 - \epsilon) \sum_{e \in X} v(e) \quad (8.6)
\]

Next, we bound the second term, the value earned from long elements, \(e \in Y\). Consider the set of cut points \(U_j\). Two operators are applied to each long element. First, it is cut and the subset \(U_j(e)\) is defined. Second, short elements are discarded from \(U_j(e)\). The resulting subset is \(\{(i, t_1, t_2) \in U_j(e) \mid t_2 - t_1 \geq \phi\}\) and therefore we
would like to bound the difference

\[
\sum_{e \in Y} \left( v(e) - \sum_{e' \in \{(i,t_1,t_2) \in U_j(e) \mid t_2 - t_1 \geq \phi\}}^{u \in \mathbb{U}} v(e') \right)
\]

Consider an element \( e = (i,t_1,t_2) \in Y \) cut by cut points set \( U_j \). As shown in Equation (8.3), the loss caused by cutting \( e \) at cut point \( u \in (t_1,t_2] \) is independent of other cuts that are applied to \( e \) and is equal to \( \ell(e,u) \). Thus we can consider each cut point separately.

As mentioned above, if \( e = (i,t_1,t_2) \in Y \), the second operator discards elements \( e' \in U_j(e) \) that are short. Since the distance between each pair of cut points in \( U_j \) is at least \( \frac{1}{\mu} = \frac{\phi^2}{\epsilon} > \phi \), each such short element \( e' \) is either \((i,t_1,u)\) or \((i,u,t_2)\) for some unique cut point \( u \in U_j \). In addition, it must hold that either \( u - t_1 < \phi \) or \( t_2 - u < \phi \). We associate the value lost by discarding \( e' \) to this unique cut point \( u \).

Let \( e = (i,t_1,t_2) \in Y \) and \( u \in U_j \) be a cut point such that \( u \in (t_1,t_2] \), i.e., \( u \) cuts \( e \). There are three cases to consider.

1. If \( u - t_1 < \phi \), element \( e' = (i,t_1,u-1) \in U_j(e) \) is discarded and a loss of \( v(e') \) is associated with \( u \) in addition to \( \ell(e,u) \). Thus the total loss is at most

\[
v(e') + \ell(e,u) = \sum_{t=t_1}^{u-1} p_t(i) + \sum_{t=t_1+1}^{u-1} g_{i,t}^+ - c_{i,t_1} - c_{i,u-1} + g_{i,u}^+ + c_{i,u}^- + c_{i,u-1}^- \leq \\
\leq \sum_{t=t_1}^{u-1} p_t(i) + \sum_{t=t_1+1}^{u} g_{i,t}^+ + c_{i,u-1}^- \leq \sum_{t=t_1}^{u} p_t(i) + \sum_{t=t_1+1}^{u+\phi} g_{i,t}^+
\]

where the equality is due to Equation (8.2) and the last inequality is due to the profit-cost ratio.

2. If \( t_2 - u < \phi \), element \( e' = (i,u,t_2) \in U_j(e) \) is discarded and a loss of \( v(e') \) is
associated with \( u \) in addition to \( \ell(e, u) \). Thus the total loss is at most

\[
v(e') + \ell(e, u) = \sum_{t = u}^{t_2} p_t(i) + \sum_{t = u + 1}^{t_2} g_{i,t}^+ - c_{i,u}^+ - c_{i,t_2}^- + g_{i,u}^+ + c_{i,u}^+ + c_{i,u}^- \leq \\
\leq \sum_{t = u}^{t_2} p_t(i) + \sum_{t = u}^{t_2} g_{i,t}^+ + c_{i,u}^- \leq \sum_{t = u - \phi}^{t_2} p_t(i) + \sum_{t = u - \phi + 1}^{t_2} g_{i,t}^+
\]

where the equality is due to Equation (8.2) and the last inequality is due to the profit-cost ratio.

3. If \( t_2 - u \geq \phi \) and \( u - t_1 \geq \phi \), no elements are discarded from \( U_j(e) \). Thus the only loss is \( \ell(e, u) \) and can be bounded by

\[
\ell(e, u) = g_{i,u}^+ + c_{i,u}^+ + c_{i,u}^- \leq \sum_{t = u - \phi}^{u + \phi - 1} p_t(i) + \sum_{t = u - \phi + 1}^{u + \phi - 1} g_{i,t}^+
\]

Overall we can bound the loss induced by cutting long elements at cut points \( U_j \) (due to loss \( \ell(e, u) \) and discarded short elements) by

\[
\sum_{(i,t_1,t_2) \in Y} \sum_{u \in (t_1,t_2) \cap U_j} \left( \sum_{t = \max\{t_1, u - \phi\}}^{\min\{t_2, u + \phi - 1\}} p_t(i) + \sum_{t = \max\{t_1 + 1, u - \phi + 1\}}^{\min\{t_2, u + \phi - 1\}} g_{i,t}^+ \right)
\]
This means that the total value gained from elements that were originally in $Y$ is

$$
\sum_{e \in Y} \sum_{j=1}^{\frac{1}{\mu}} v(e') \geq
\frac{1}{\mu} \sum_{e \in Y} v(e) - \sum_{j=1}^{\frac{1}{\mu}} \sum_{(i,t_1,t_2) \in Y \cap U_j} \left( \min_{t \in \{t_1, u-\phi\}} p_t(i) + \sum_{t \in \{t_1, u-\phi+1\}} g_{i,t}^+ \right) \geq
\frac{1}{\mu} \sum_{e \in Y} v(e) - \sum_{(i,t_1,t_2) \in Y} \left( \min_{t \in \{t_1, u-\phi\}} p_t(i) + \sum_{t \in \{t_1, u-\phi+1\}} g_{i,t}^+ \right) \geq
\frac{1}{\mu} \sum_{e \in Y} v(e) - 2\phi \sum_{(i,t_1,t_2) \in Y} \left( \sum_{t \in \{t_1, t_2\}} p_t(i) + \sum_{t \in \{t_1+1, t_2\}} g_{i,t}^+ \right)
$$

where the second inequality follows from the fact that for every $j_1 \neq j_2$ it holds that $U_{j_1} \cap U_{j_2} = \{1, T + 1\}$. The last inequality is due to the fact that the profit and gains of element $(i, t_1, t_2)$ is lost at stage $t \in [t_1, t_2]$ if it is cut by a cut point $u$ such that $|u-t| \leq \phi$. Thus, its value is lost in at most $2\phi$ instances. Due to the profit-cost ratio, for each long element $e = (i, t_1, t_2) \in Y$ it holds that

$$
c_{i,t_1}^+ + c_{i,t_2}^- \leq 2\phi \cdot \frac{\sum_{t=t_1}^{t_2} p_t(i) + \sum_{t=t_1+1}^{t_2} g_{i,t}^+}{t_2 - t_1} \leq \epsilon \cdot \sum_{t=t_1}^{t_2} p_t(i) + \epsilon \cdot \sum_{t=t_1+1}^{t_2} g_{i,t}^+
$$
By substituting $4\phi(c_{i,t_1}^+ + c_{i,t_2}^-) \leq 4\phi\epsilon \left( \sum_{t=t_1}^{t_2} p_t(i) + \sum_{t=t_1+1}^{t_2} g_{i,t}^+ \right)$ we get that

$$\frac{1}{\mu} \sum_{e \in Y} v(e) - 2\phi \sum_{(i,t_1,t_2) \in Y} \left( \sum_{t=t_1}^{t_2} p_t(i) + \sum_{t=t_1+1}^{t_2} g_{i,t}^+ \right) =$$

$$\sum_{(i,t_1,t_2) \in Y} \left( \frac{1}{\mu} \left( \sum_{t=t_1}^{t_2} p_t(i) + \sum_{t=t_1+1}^{t_2} g_{i,t}^+ - c_{i,t_1}^+ - c_{i,t_2}^- \right) - 2\phi \left( \sum_{t=t_1}^{t_2} p_t(i) + \sum_{t=t_1+1}^{t_2} g_{i,t}^+ \right) \right) \geq$$

$$\sum_{(i,t_1,t_2) \in Y} \left( \frac{1}{\mu} - 2\phi - 4\phi\epsilon \right) \left( \sum_{t=t_1}^{t_2} p_t(i) + \sum_{t=t_1+1}^{t_2} g_{i,t}^+ \right) - \frac{1}{\mu} - 4\phi \left( c_{i,t_1}^+ + c_{i,t_2}^- \right) \geq$$

$$\left( \frac{1}{\mu} - 2\phi - 4\phi\epsilon \right) \sum_{e \in Y} v(e) \geq \left( \frac{1}{\mu} - \frac{\phi}{\epsilon} \right) \sum_{e \in Y} v(e)$$

where the last inequality follows the fact that $\epsilon < \frac{1}{4}$. Overall, we get that

$$\sum_{e \in Y} \sum_{j=1}^{\frac{1}{\mu}} \sum_{e'=(i,t_1,t_2) \in U_j(e), t_2-t_1 \geq \phi} v(e') \geq \left( \frac{1}{\mu} - \frac{\phi}{\epsilon} \right) \sum_{e \in Y} v(e) \quad (8.7)$$

Lastly, we bound the third term, gains $g^-$ earned in all cut instance solutions. Consider a cut point set $U_j$ and gain $g_{i,t}^-$ earned in solution $\tilde{S}^*$, i.e., $i \notin S_{t-1}^* \cup S_t^*$. Therefore, any element $(i,t_1,t_2)$ such that $t \in [t_1,t_2]$ or $t-1 \in [t_1,t_2]$ is not in $\tilde{S}_j$. Thus, unless $t \in U_j$, gain $g_{i,t}^-$ is earned in $\tilde{S}_j$ and in some solution $\tilde{S}_{i,r} \subseteq \tilde{S}_j$ such that $t \in [u_i^t,u_{i+1}^t-1]$ and $u_i^t \in U_j$. Again, we can use the fact that for every $j_1 \neq j_2$ it holds that $U_{j_1} \cap U_{j_2} = \{1,T+1\}$ and get that

$$\sum_{j=1}^{\frac{1}{\mu}} \sum_{t \in [2,T] \setminus U_j} \sum_{i \notin \tilde{S}_{j}(t-1) \cap \tilde{S}_j(t)} g_{i,t}^- \geq \left( \frac{1}{\mu} - 1 \right) \sum_{t=1}^{T} \sum_{i \notin \tilde{S}_{j}(t-1) \cap \tilde{S}_{j}(t)} g_{i,t}^- \quad (8.8)$$

By substituting inequalities (8.6),(8.7) and (8.8) in Inequality (8.5) we get that

$$B \geq \left( \frac{1}{\mu} - \frac{\phi}{\epsilon} \right) \left( \sum_{e \in \tilde{S}^*} v(e) + \sum_{t \in [2,T]} \sum_{i \notin \tilde{S}_{j}(t-1) \cap \tilde{S}_{j}(t)} g_{i,t}^- \right) = \left( \frac{1}{\mu} - \frac{\phi}{\epsilon} \right) f_{\mathcal{Q}}(S_{t}^*)_{t=1}^{T}$$

155
For each set of values, their average is smaller or equal to their maximum value. Thus there must exist at least one set of cut points $U_j$ such that the sum of values of the solutions $\left(\tilde{S}_{j^*,r}\right)_{r=0}^{k_j-1}$ for its cut instances, $Q_{U_j}$, is at least $\mu \cdot B$, the average value of a set of solutions for a set of cut instance $Q_{U_j}$ (for $j = 1, \ldots, \frac{1}{\mu}$). We get that

$$\sum_{e \in \tilde{S}_{j^*,r}} v(e) + \sum_{t \in [2,T]} \sum_{i \in U_j} g_{i,t} \geq \mu B = (1 - \epsilon) f_Q \left( (S_t^*)^T_{t=1} \right)$$

At iteration $j^*$, in which Algorithm 20 considers the cut instances $Q_{U_{j^*}}$, algorithm $A$ provides an approximate solution for each cut instance. Thus the value of the solution returned by $A$ for the $r$-th cut instance is at least

$$\alpha \cdot \left(\sum_{e \in \tilde{S}_{j^*,r}} v(e) + \sum_{t \in [u^j_{r+1},u^j_{r+1}-1]} \sum_{i \in U_j} g_{i,t}\right)$$

Summing over all cut instances in $Q_{U_{j^*}}$ provides a solution with value at least

$$\alpha \cdot \left(\sum_{e \in \tilde{S}_{j^*,r}} v(e) + \sum_{t \in [2,T]} \sum_{i \in U_j} g_{i,t}\right) \geq (1 - \epsilon) \cdot \alpha \cdot f_Q \left( (S_t^*)^T_{t=1} \right)$$

The correctness of Theorem 1.5.1 follows immediately from Theorem 8.1.7 and Lemma 8.1.5.

8.2 Approximation Scheme for Submodular $d$-GMK

In this section we present an approximation algorithm for Submodular $d$-GMK problem. Similarly to the approach used for $d$-GMK, we first present an approximation algorithm for instances with a fixed time horizon length and then extend it for general instances.
8.2.1 Bounded Time Horizon

In this section we provide a reduction from an instance of Submodular $q$-GMK to a generalization of Submodular $d$-MKCP (for specific values of $d$ and $q$). The generalization was presented in [37] and is called Submodular $d$-MKCP With A Matroid Constraint (Submodular $d$-MKCP+) and is defined by a tuple $(I, \mathcal{K}, p, \mathcal{I})$, where $(I, \mathcal{K}, p)$ forms an instance of Submodular $d$-MKCP. Also, the set $\mathcal{I} \subseteq 2^I$ defines a matroid constraint.

A feasible solution for Submodular $d$-MKCP+ is a set $S \subseteq I$ and a tuple of feasible assignments $A$ (w.r.t $\mathcal{K}$) of $S$. The goal is to find a feasible solution which maximizes $p(S)$.

We are now ready to provide the reduction. The following definition presents the construction of the reduction.

**Definition 8.2.1.** Let $Q = ((\mathcal{P}_t)_{t=1}^T, g^+, g^-)$ be an instance of Submodular $d$-GMK, where $\mathcal{P}_t = (I, \mathcal{K}_t, p_t)$ and $\mathcal{K}_t = (K_{t,j})_{j=1}^{d_t}$. Given instance $Q$, the operator $R$ returns the Submodular $dT$-MKCP+ instance $R(Q) = (E, \tilde{\mathcal{K}}, \tilde{p}, \tilde{\mathcal{I}})$ where

- $E = I \times 2^{|T|}$
- $\mathcal{I} = \{ S \subseteq E \mid \forall i \in I : |S \cap \{\{i\} \times 2^{|T|}| = 1 \}$
- For $t = 1, ..., T$, $j \leq d_t$ set MKC $\tilde{K}_{t,j} = (\tilde{w}_{t,j}, B_{t,j}, W_{t,j})$ over $E$, where $K_{t,j} = (w_{t,j}, B_{t,j}, W_{t,j})$ and
  
  \[
  \tilde{w}_{t,j}((i, D)) = \begin{cases} 
  w_{t,j}(i) & t \in D \\
  0 & \text{otherwise}
  \end{cases}
  \]

- For $t = 1, ..., T$, $d_t < j \leq d$ set MKC $\tilde{K}_{t,j} = (w_0, \{b\}, W_0)$ over $E$, where $w_0 : 2^E \to \{0\}$, $W_0(b) = 0$ and $b$ is an arbitrary bin (object).
- $\tilde{\mathcal{K}} = (\tilde{K}_{t,j})_{t \in [T], j \in [d]}$.
- For $t = 1, ..., T$: set $\tilde{p}_t(S) = p_t (\{i \in I \mid \exists (i, D) \in S : t \in D\})$
The objective function $\tilde{f}$ is defined as follows.

$$\tilde{p}(S) = \sum_{t=1}^{T} \tilde{p}_t(S) + \sum_{(i,D) \in S} \left( \sum_{t \in D: t-1 \in D} g_{i,t}^+ + \sum_{t \notin D: t-1 \notin D} g_{i,t}^- \right)$$

Each element $(i, D) \in E$ states the subset of stages in which item $i$ is assigned. I.e., $i$ is only assigned in stages $t \in D$. Thus any solution should include at most one element $(i, D)$ for each $i \in I$. This constraint is fully captured by the partition matroid constraint defined by the set of independent sets $\mathcal{I}$. Finally, if an element is $(i, D)$ is selected, we must assign $i$ in each MKC $K_{t,j}$ for $j \in [d_t], t \in D$. This is captured by the weight function $\tilde{w}$, as an element $(i, D)$ weighs $w_{t,j}(i)$ if and only if $t \in D$ (otherwise its weight is zero and it can be assigned for “free”).

Lemma 8.2.1. The reduced instance $R(Q)$ of a Submodular $d$-GMK instance $Q$ over time horizon $T$ is a valid Submodular $dT$-MKCP+ instance.

Proof. Let $Q = ((\mathcal{P}_t)_{t=1}^{T}, g^+, g^-)$ be an instance of Submodular $d$-GMK, where $\mathcal{P}_t = (I, K_t, p_t)$ and $K_t = (K_{t,j})_{j=1}^{d_t}$. Also, let $R(Q) = \left( E, \tilde{K}, \tilde{p}, \mathcal{I} \right)$ be the reduced instance of $Q$ as defined in Definition 8.2.1. It is easy to see that the set $\mathcal{I}$ is the independent sets of a partition matroid, as for each item $i$ at most one element $(i, D)$ can be chosen. Thus, $\mathcal{I}$ is the family of independent sets of a matroid as required.

Next, $\mathcal{K}$ defines a tuple of MKCs, so all that is left to prove is that $\tilde{p}$ is non-negative, monotone and submodular. For each element $(i, D) \in E$ we can define a fixed non-negative value

$$v((i, D)) = \sum_{t \in D: t-1 \in D} g_{i,t}^+ + \sum_{t \notin D: t-1 \notin D} g_{i,t}^-$$

Thus the function $h(S) = \sum_{e \in S} v(e)$ is non-negative and modular. For $t = 1, \ldots, T$ function $p_t$ is non-negative, monotone and submodular. Thus, due to Claim 3.1.5, functions $(\tilde{p}_t)_{t=1}^{T}$ are also non-negative, monotone and submodular. This means that $\tilde{p}$ is non-negative, monotone and submodular as it is the sum of non-negative, monotone and submodular functions. \qed

Next, we will prove that operator $R$ defines an approximation factor preserving
Lemma 8.2.2. Let $Q$ be an instance of Submodular $d$-GMK with time horizon $T$. For any feasible solution $(S_t, A_t)_{t=1}^T$ for $Q$ a feasible solution $(S, (\tilde{A}_{t,j})_{t\in[T], j\in[d]})$ for $R(Q)$ exists such that $f_Q((S_t)_{t=1}^T) = \tilde{f}(S)$.

Proof. Let $Q = (\mathcal{P}_t)_{t=1}^T, g^+, g^-$ be an instance of Submodular $d$-GMK, where $\mathcal{P}_t = (I, \mathcal{K}_t, p_t)$ and $\mathcal{K}_t = (K_{t,j})_{j=1}^{d_t}$. Also, let $R(Q) = (E, \tilde{\mathcal{K}}, \tilde{p}, T)$ be the reduced instance of $Q$, where $\tilde{\mathcal{K}} = (\tilde{K}_{t,j})_{t\in[T], j\in[d]}$ and $\tilde{K}_{t,j} = (\tilde{w}, B_{t,j}, W_{t,j})$ (see Definition 8.2.1). Consider some feasible solution $(S_t, A_t)_{t=1}^T$ for $Q$, where $A_t = (A_{t,j})_{j=1}^{d_t}$. In the following we define a solution $(S, (\tilde{A}_{t,j})_{t\in[T], j\in[d]})$ for $R(Q)$. Let

$$S = \{(i, D) \mid i \in I, \quad D = \{t \in [T] \mid i \in S_t\}\}$$

It can be easily verified that $S \in \mathcal{I}$. Since $\tilde{p}_t(S) = p_t(S)$ for every $t = 1, \ldots, T$ and $S \subseteq E$ it holds that

$$\tilde{p}(S) = \sum_{t=1}^T \tilde{p}_t(S_t) + \sum_{(i, D) \in S} \left( \sum_{t \in D, t-1 \not\in D} g_{i,t}^+ + \sum_{t \in D, t-1 \not\in D} g_{i,t}^- - \sum_{t \in D, t \not\in D} c_{i,t}^+ - \sum_{t \in D, t \not\in D} c_{i,t}^- \right) =$$

$$\sum_{t=1}^T p_t(S_t) + \sum_{t \in [2, T]} \left( \sum_{i \in S_{t-1} \cap S_t} g_{i,t}^+ + \sum_{i \not\in S_{t-1} \cup S_t} g_{i,t}^- \right) - \sum_{t=1}^T \left( \sum_{i \in S_t \setminus S_{t-1}} c_{i,t}^+ + \sum_{i \not\in S_t \setminus S_{t+1}} c_{i,t}^- \right) = f_Q((S_t)_{t=1}^T).$$

Next, we need to show an assignment $\tilde{A}_{t,j}$ of $S$ for each MKC in $\tilde{\mathcal{K}}$. Let $t \in [T]$ and $j \in [d]$, and consider the following two cases:

1. If $j > d_t$, recall $\tilde{K}_{t,j} = (w_0, \{b\}, W_0)$ where $w_0(i, D) = 0$ for all $(i, D) \in E$ and $W_0(b) = 0$ by definition. We define $\tilde{A}_{t,j}$ by $\tilde{A}_{t,j}(b) = S$. It thus holds that $w_0(\tilde{A}_{t,j}(b)) = w_0(S) = 0 = W_0$. That is, $\tilde{A}_{t,j}$ is feasible.
2. If \( j \leq d_t \), then let \( b^* \in B_{t,j} \) be some unique bin in \( B_{t,j} \) and we define the assignment 
\( \tilde{A}_{t,j} : B_{t,j} \to 2^E \) by

\[
\tilde{A}_{t,j}(b) = (A_{t,j}(b) \times 2^{[T]}) \cap S \\
\tilde{A}_{t,j}(b^*) = ((A_{t,j}(b^*) \times 2^{[T]}) \cap S) \cup \{(i, D) \in S \mid t \not\in D\}. \tag{8.9}
\]

The assignment \( \tilde{A}_{t,j} \) is a feasible assignment w.r.t \( \tilde{K}_{t,j} \) since for each bin \( b \in B_{t,j} \) it holds that

\[
\sum_{(i, D) \in \tilde{A}_{t,j}(b)} \tilde{w}_{t,j}((i, D)) \leq \sum_{i \in \tilde{A}_{t,j}(b)} \tilde{w}_{t,j}(i) \leq W_{t,j}(b).
\]

Let \( (i, D) \in S \). If \( i \in S_t \) there is \( b \in B_{t,j} \) such that \( i \in A_{t,j}(b) \), hence \( (i, D) \in \tilde{A}_{t,j}(b) \) by Equation (8.9). If \( i \not\in S_t \) then \( t \not\in D \) and thus \( (i, D) \in \tilde{A}_{t,j}(b^*) \). Overall, we have \( S \subseteq \bigcup_{b \in B_{t,j}} \tilde{A}_{t,j}(b) \). By Equation (8.9) it follows that \( S \supseteq \bigcup_{b \in B_{t,j}} \tilde{A}_{t,j}(b) \) as well, thus \( S = \bigcup_{b \in B_{t,j}} \tilde{A}_{t,j}(b) \). I.e, \( \tilde{A}_{t,j} \) is an assignment of \( S \).

Note that the assignments can be constructed in polynomial time. We can conclude that 
\( (S, (\tilde{A}_{t,j})_{t \in [T], j \in [d]}) \) is a feasible solution for \( R(Q) \), and its value is \( f_Q ((S_t)_{t=1}^T) \). \( \square \)

**Lemma 8.2.3.** Let \( Q \) be an instance of Submodular \( d \)-GMK with time horizon \( T \). For any feasible solution \( (S, (\tilde{A}_{t,j})_{t \in [T], j \in [d]}) \) for \( R(Q) \), a feasible solution \( (S_t, A_t)_{t=1}^T \) for \( Q \) can be constructed in polynomial time such that \( f_Q ((S_t)_{t=1}^T) = \tilde{f}(S) \).

**Proof.** Let \( Q = ((I, T^+, T^-), P_t) \) be an instance of Submodular \( d \)-GMK, where \( P_t = (I, CK_t, p_t) \) and \( CK_t = (K_{t,j})_{j=1}^{d_t} \). Also, let \( R(Q) = \left( E, \tilde{K}, \tilde{p}, \tilde{T} \right) \) be the reduced instance of \( Q \). and \( (S, (\tilde{A}_{t,j})_{t \in [T], j \in [d]}) \) be a feasible solution for \( R(Q) \). We define the solution \( (S_t, A_t)_{t=1}^T \) for \( Q \) as follows. For every stage \( t \) we set \( S_t = \{ i \in I \mid \exists (i, D) \in S : t \in D \} \). For every \( t = 1, \ldots, T, j = 1, \ldots, d_t \), and bin \( b \in B_{t,j} \) (the set of bins in MKC \( K_{t,j} \)) let \( A_{t,j}(b) = \{ i \in I \mid \exists (i, D) \in \tilde{A}_{t,j}(b) : t \in D \} \). Observe that the sets \( (S_t)_{t=1}^T \) and assignments \( (A_{t,j})_{t \in [T], j \in [d]} \) can be constructed in polynomial time as at most \( |I| \) elements can be chosen due to the matroid constraint. The assignment \( A_{t,j} \) is an
assignment of $S_t$ since

$$S_t = \{ i \in I \mid \exists (i, D) \in S : t \in D \} = \bigcup_{b \in B_{t,j}} \{ i \in I \mid (i, D) \in \tilde{A}_{t,j}(b) : t \in D \} = \bigcup_{b \in B_{t,j}} A_{t,j}(b),$$

where the second equality follows the feasibility of the solution for $R(Q)$. In addition, $A_{t,j}$ is a feasible assignment for MKC $K_{t,j}$ since for every bin $b \in B_{t,j}$ it holds that

$$\sum_{i \in A_{t,j}(b)} w_{t,j}(i) = \sum_{(i, D) \in \tilde{A}_{t,j}(b) : t \in D} \tilde{w}_{t,j}((i, D)) = \sum_{(i, D) \in \tilde{A}_{t,j}(b)} \tilde{w}_{t,j}((i, D)) \leq W_{t,j}(b)$$

Thus $(S_t, A_t)_{t=1}^T$ is a feasible solution for $Q$.

Lastly, consider the value of the solution for $Q$. It holds that

$$f_Q((S_t)_{t=1}^T) =$$

$$\sum_{t=1}^T p_t(S_t) + \sum_{(i, D) \in S} \left( \sum_{i \in S_{t-1} \cap S_t} g^+_{i,t} + \sum_{i \notin S_{t-1} \cup S_t} g^-_{i,t} \right) - \sum_{t \in [T]} \left( \sum_{i \in S_t \setminus S_{t-1}} c^+_{i,t} + \sum_{i \in S_t \setminus S_{t+1}} c^-_{i,t} \right) =$$

$$\sum_{t=1}^T \tilde{p}_t(S) + \sum_{(i, D) \in S} \left( \sum_{t \in D : t-1 \in D} g^+_{i,t} + \sum_{t \notin D : t-1 \notin D} g^-_{i,t} - \sum_{t \in D : t-1 \notin D} c^+_{i,t} - \sum_{t \in D : t+1 \notin D} c^-_{i,t} \right) = \tilde{p}(S)$$

\[\square\]

For any Submodular $d$-GMK instance with a fixed time horizon $T$, $R(Q)$ can be constructed in polynomial (as $|E| = |I| \cdot 2^{|T|}$). The next corollary follows from this observation and from the last two lemmas.

**Corollary 8.2.4.** For any fixed $T \in \mathbb{N}$, there exists an approximation factor preserving reduction (see Definition 3.3.1) from Submodular $d$-GMK with a time horizon bounded by $T$ to Submodular $dT$-MKCP+.

In [37] a $(1 - \frac{1}{e})$-approximation algorithm for Submodular $d$-MKCP+ is presented. Thus, the next lemma follows from the above corollary.
Lemma 8.2.5. For any fixed $T \in \mathbb{N}$ there exists a randomized $(1 - \frac{1}{e})$-approximation algorithm for $d$-GMK with a time horizon bounded by $T$.

8.2.2 General Time Horizon

In this section we present an algorithm for Submodular $d$-GMK with a non-constant time horizon $T$. In order to do so the time horizon is cut at several stages into sub-instances. Each sub-instance is optimized independently, and then the solutions are combined to create a solution for the complete instance. A similar technique was used in [8], though they considered a model without change costs.

The following algorithm is identical to the algorithm presented in Section 20. We present it here as well for completeness.

Algorithm 21: Non Constant Time Horizon

Input: A Submodular $d$-GMK instance $Q$ with time horizon $T$, $0 < \epsilon < \frac{1}{2}$ and $\alpha$-approximation algorithm $A$ for Submodular $d$-GMK with time horizon $T \leq \frac{2\phi Q}{\epsilon^2}$.

1. Set $\mu = \frac{\epsilon^2}{\phi Q}$.
2. for $j = 1, \ldots, \frac{1}{\mu}$ do
3. Set $U_j = \left\{ \frac{a}{\mu} + j - 1 \mid a \in \mathbb{N}, a \geq 1, \frac{a}{\mu} + j - 1 \leq T - \frac{1}{\mu} \right\} \cup \{0, T+1\}$.
4. Find a solution for each cut instance in $Q_{U_j}$ using algorithm $A$ and set $S_j$ as the respective cut solution.
5. Return the solution $S_j$ which maximizes the objective function $f_Q$.

Lemma 8.2.6. Let $0 < \epsilon < \frac{1}{4}$, $\phi \geq 1$ and $A$ be an $\alpha$-approximation algorithm for Submodular $d$-GMK with time horizon $T \leq \frac{2\phi}{\epsilon^2}$. Also, let $Q$ be an instance of Submodular $d$-GMK such that $\phi_Q \leq \phi$. Algorithm 21 approximates $Q$ within a factor of $(1 - \epsilon)\alpha$.

Proof. Let $Q = (\{P_t\}_{t=1}^T, g^+, g^-)$ be an instance of Submodular $d$-GMK with time horizon $T$ for which we denote its profit-cost ratio by $\phi$ and assume for simplicity it is integral. Let $0 < \epsilon < \frac{1}{4}$ and $\mu = \frac{\epsilon^2}{\phi}$. Also, let $A$ be an $\alpha$-approximation algorithm for Submodular $d$-GMK with time horizon $T' \leq \frac{2\phi}{\epsilon^2} = \frac{2}{\mu}$. Note, if $T \leq \frac{2}{\mu}$, the cut points...
set $U_0$ will be an empty set. Thus $A$ will return an $\alpha$-approximation solution for $Q$ as required.

We show there exists a set of cut points $U_j$ and a tuple of solutions \((S_t, A_t)_{t=ur}^{ur+1-1}\) for each cut instance in $Q_{U_j} = (q_j^r)_{r=0}^{k-1}$, such that the sum of values of the solutions, $\sum_{r=0}^{k-1} f_{q_j^r}\left((S_t)_{t=ur}^{ur+1-1}\right)$, is sufficiently large. From Corollary 8.1.6 it follows that the value of a cut solution is larger than the sum of its parts (due to lost gains and change costs saved if an item is assigned in adjacent instances). Thus, this will prove that the maximum cut solution found is sufficiently large as well.

Let \((S^*_t, A^*_t)_{t=1}^{T}\) be an optimal solution for $Q$. Also, let $Q_{U_j} = (q_j^r)_{r=0}^{k-1}$ be the tuple of cut instance w.r.t cut points set $U_j = \{u_j^0, \ldots , u_j^k\}$. For $r = 0, \ldots , k - 1$, the tuple \((S^*_t, A^*_t)_{t=u_j^r+1}^{u_j^{r+1}-1}\) is a feasible solution for cut instance $q_j^r$ due to the feasibility of the optimal solution. The sum of values of these solutions is

$$
\sum_{r=0}^{k-1} f_{q_j^r}\left((S_t)_{t=ur}^{ur+1-1}\right) = \sum_{r=0}^{k-1} \left( \sum_{t=ur}^{ur+1-1} p_t(S^*_t) + \sum_{t=ur+1}^{ur+1-1} \left( \sum_{i \in S_{t-1} \cap S_t} g_{i,t}^+ + \sum_{i \notin S_{t-1} \cup S_t} g_{i,t}^- \right) \right)
$$

$$
= \sum_{t=1}^{T} p_t(S^*_t) + \sum_{t=[2,T] \setminus U_j} \left( \sum_{i \in S_{t-1} \cap S_t} g_{i,t}^+ + \sum_{i \notin S_{t-1} \cup S_t} g_{i,t}^- \right)
$$

$$
= f_Q\left((S_t^*)_{t=1}^{T}\right) - \sum_{t \in U_j} \left( \sum_{i \in S_{t-1} \cap S_t} g_{i,t}^+ + \sum_{i \notin S_{t-1} \cup S_t} g_{i,t}^- \right)
$$

For any set of cut points $U_1, \ldots , U_{\frac{p}{n}}$, such a tuple of solution can be constructed. Their
total value is

$$\sum_{j=1}^{k-1} \sum_{r=0}^{k-1} f_{q_j^r} \left( (S_t^*)^{u_{r+1}}_{t=u_r} \right) = \sum_{j=1}^{k-1} \left( f_{Q} \left( (S_t^*)^{T}_{t=1} \right) - \sum_{t \in U_j} \left( \sum_{i \in S_{t-1} \cap S_t} g_{i,t}^+ + \sum_{i \notin S_{t-1} \cup S_t} g_{i,t}^- \right) \right)$$

$$\geq \frac{1}{\mu} \cdot f_{Q} \left( (S_t^*)^{T}_{t=1} \right) - \sum_{t=2}^{T} \left( \sum_{i \in S_{t-1} \cap S_t} g_{i,t}^+ + \sum_{i \notin S_{t-1} \cup S_t} g_{i,t}^- \right)$$

$$\geq \left( \frac{1}{\mu} - 1 \right) \cdot f_{Q} \left( (S_t^*)^{T}_{t=1} \right)$$

There must be a value \( j^* \) for which the sum of values of solutions \( \sum_{r=0}^{k-1} f_{q_j^r} \left( (S_t^*)^{u_{r+1}}_{t=u_r} \right) \) is at least as high as the average sum of values of solutions. It follows that

$$\sum_{r=0}^{k-1} f_{q_j^r} \left( (S_t^*)^{u_{r+1}}_{t=u_r} \right) \geq (1 - \mu) \cdot f_{Q} \left( (S_t^*)^{T}_{t=1} \right)$$

Let \( \left( (S_t, A_t)_{t=u_r}^{u_{r+1}-1} \right)_{r=0}^{k-1} \) be the solutions returned by \( A \) for the cut instances. Since Algorithm \( A \) returns \( \alpha \)-approximation solution for the problem, it holds that

$$\sum_{r=0}^{k-1} f_{q_j^r} \left( (S_t)^{u_{r+1}}_{t=u_r} \right) \geq (1 - \mu) \cdot \alpha \cdot f_{Q} \left( (S_t^*)^{T}_{t=1} \right)$$

Finally, from Corollary 8.1.6 we can conclude that the value of \( S_j^* \) is at least as high as the sum of values of solution for the cut instances. Thus, its value is at least \( (1 - \mu) \cdot \alpha \cdot f_{Q} \left( (S_t^*)^{T}_{t=1} \right) \) as required.

The correctness of Theorem 1.5.4 follows immediately from Lemma 8.2.6 and Lemma 8.2.5.
8.3 Hardness Results

In this section we present two hardness results for 1-GMK. First, we show no constant approximation ratio exists for 1-GMK (with unbounded profit-cost ratio), even if there is only one bin per stage. Then, we show that even if we wither down the model by removing the change costs, limiting the time horizon length to $T = 2$, and only having a single bin per stage, the problem still does not admit an EPTAS.

The above results are proved by showing an approximation preserving reduction from $d$-Dimensional Knapsack ($d$-KP) and Multidimensional Knapsack. For $d \in \mathbb{N}$, in $d$-KP we are given a set of items $I$, each equipped with a profit $p_i$, as well as a $d$-dimensional weight vector $w^i \in [0,1]^d$. We denote $j$-th coordinate of $w^i$ by $w^i_j$. In addition, we are given a single bin equipped with a $d$-dimensional capacity vector $W \in \mathbb{R}^d_{\geq 0}$. A subset $S \subseteq I$ is a feasible solution if $\sum_{i \in S} w^i \leq W$. The objective is to find a feasible solution $S$ which maximizes $\sum_{i \in S} p_i$.

The Multidimensional Knapsack problem is the generalization of $d$-KP in which $d$ is not fixed. That is, the input for the problem is a $d$-KP instance for some $d \in \mathbb{N}$. The solutions and their values are the solution and values of the $d$-KP instance.

Note that $d$-KP is a special case of $d$-MKCP, where the set of MKCs is $\mathcal{K} = (K_j)_{j=1}^d$. The $j$-th MKC is $K_j = (w_j, B_j, W_j)$, where $w_j(i) = w^i_j$, $B_j = \{b\}$ and $W_j(b) = W_j$, where $W_j$ is the $j$-th coordinate of the capacity vector $W$. Finally, the profit function $p : I \rightarrow \mathbb{R}_{\geq 0}$ is defined as $p(i) = p_i$ for any $i \in I$. For simplicity we will use this notation for $d$-KP and Multidimensional Knapsack throughout this section.

**Lemma 8.3.1.** There is an approximation preserving reduction from the Multidimensional Knapsack problem to 1-GMK with a single bin in each stage.

**Proof.** Let $Q = (I, \mathcal{K}, p)$ be an instance of Multidimensional Knapsack, where $\mathcal{K} = (K_j)_{j=1}^d$. We define an instance of 1-GMK as follows. Define $T = d$, and for $j = 1, \ldots, d$ define $P_j = (I, (K_j), h)$ with $h(i) = \frac{p(i)}{d}$ for all $i \in I$. The gains vectors are defined as zero vectors, $g^+ = g^- = 0$. Finally, we define the change cost vectors. For all $i \in I$ we set

\[
c_{i,t}^+ = \begin{cases} p(i) & t \in [2, d] \\ 0 & \text{otherwise.} \end{cases}, \quad c_{i,t}^- = \begin{cases} p(i) & t \in [1, d - 1] \\ 0 & \text{otherwise.} \end{cases}
\]
The tuple $\tilde{Q} = ((P_t)_{t=1}^d, g^+, g^-, c^+, c^-)$ is a 1-GMK instance with time horizon $T = d$.

Let $(S, A)$ be a feasible solution for $Q$, where $A = (A_j)_{j=1}^d$. We can easily construct a solution for $\tilde{Q}$ by setting $A_j = (\tilde{A}_j)$ for $j = 1, \ldots, d$. Then, $(S_t, A_t)_{t=1}^d$, where $S_j = S$ for $j = 1, \ldots, d$, is a solution for $\tilde{Q}$. Note that all items are either assigned or not assigned in all stages. Thus the value of the solution is

$$f_{\tilde{Q}}((S_t)_{t=1}^d) = \sum_{t=1}^d h(S_t) - \sum_{t=1}^T \left( \sum_{i \in S_t \setminus S_{t+1}} c^-_{i,t} + \sum_{i \in S_t \setminus S_{t-1}} c^+_{i,t} \right) = d \cdot h(S) = p(S)$$

The solution is also feasible as for every $j \in [d]$ it holds that $A_j$ is a feasible assignments for MKC $K_j$.

Next, let $(S_t, A_t)_{t=1}^d$ be a feasible solution for $\tilde{Q}$, where $A_j = (\tilde{A}_j)$ for $j = 1, \ldots, d$. For $j = 1, \ldots, d$ let $B_j = \{b_j\}$. We define the selected items set as $S = \bigcap_{j \in [d]} S_j$ and define the assignments accordingly, $A_j(b_j) = S$ for $j = 1, \ldots, d$ and $A = (A_j)_{j=1}^d$.

Consider some $j \in [s]$, the assignment $A_j$ is feasible as $A_j(b_j) \subseteq \tilde{A}_j(b_j)$ and

$$\sum_{i \in A_j(b_j)} w_j(i) \leq \sum_{i \in A_j(b_j)} w_j(i) \leq W_j(b_j)$$

The value of the solution is

$$p(S) = d \cdot h(S) \geq \sum_{t=1}^d h(S_t) - \sum_{t=1}^T \left( \sum_{i \in S_t \setminus S_{t+1}} c^-_{i,t} + \sum_{i \in S_t \setminus S_{t-1}} c^+_{i,t} \right) = f_{\tilde{Q}}((S_t)_{t=1}^d)$$

where the inequality follows from the construction of $\tilde{Q}$. Furthermore, note that $S$ can be constructed in polynomial time, which concludes the proof. 

In [19] Chekuri and Kahanna showed that Multidimensional Knapsack does not admit any constant approximation ratio unless $NP = ZPP$. Theorem 1.5.3 follows immediately from the hardness result of [19] and Lemma 8.3.1.

We now proceed to the second hardness result.

**Lemma 8.3.2.** There is an approximation preserving reduction from 2-Dimensional
Knapsack problem to 1-GMK with time horizon $T = 2$, no change costs and a single bin per stage.

Proof. Let $Q = (I, K, p)$ be an instance of 2-dimensional knapsack, where $K = (K_1, K_2)$. Also, since $p$ is modular, it holds that $p(S) = \sum_{i \in S} p_i$.

We define an instance of 1-GMK as follows. Set $T = 2$, $\mathcal{P}_1 = (I, (K_1), h)$ and $\mathcal{P}_2 = (I, (K_2), h)$, where $h$ is the zero function, i.e., $h : I \rightarrow \{0\}$ such that $\forall i \in I$ it holds that $h(i) = 0$. Since there are only two stages, gains exists only for stage for $t = 2$. Set $g_{i,2}^+ = p(i)$ and $g_{i,2}^- = 0$ for each item $i \in I$. Finally, we set the change cost vectors as $c^+ = c^- = 0$. The tuple $\tilde{Q} = ((\mathcal{P}_1, \mathcal{P}_2), g^+, g^-, c^+, c^-)$ is a 1-GMK instance with time horizon $T = 2$. Note that since all profits, change costs and gains $g^-$ are zero we can write the objective function as

$$f_{\tilde{Q}}((S_1, S_2)) = \sum_{i \in S_1 \cap S_2} g_{i,2}^+.$$

Let $(S, A)$ be a feasible solution for $Q$, where $A = (A_1, A_2)$. We can easily construct a solution for the $\tilde{Q}$ by setting $A_1 = (\tilde{A}_1)$ and $A_2 = (\tilde{A}_2)$. Then, $(S_t, A_t)_{t=1}^2$, where $S_1 = S_2 = S$, is a solution for $\tilde{Q}$. Note that all items are either assigned in both stages or not assigned in both stages. Thus the value of the solution is

$$f_{\tilde{Q}}((S, S)) = \sum_{i \in S_1 \cap S_2} g_{i,2}^+ = \sum_{i \in S} g_{i,2}^+ = \sum_{i \in S} p_i = p(S)$$

The solution is also feasible as $A_1$ and $A_2$ are feasible assignments of $K_1$ and $K_2$ (respectively).

Next, let $(S_t, A_t)_{t=1}^2$ be a feasible solution for $\tilde{Q}$, where $A_1 = (\tilde{A}_1)$ and $A_2 = (\tilde{A}_2)$. Let $B_1 = \{b_1\}$ and $B_2 = \{b_2\}$. We define the selected items set as $S = S_1 \cap S_2$ and define the assignments accordingly, $A_1(b_1) = A_2(b_2) = S$ and $A = (A_1, A_2)$. Assignment $A_1$ is feasible as $A_1(b_1) \subseteq \tilde{A}_1(b_1)$ and

$$\sum_{i \in A_1(b_1)} w_1(i) \leq \sum_{i \in \tilde{A}_1(b_1)} w_1(i) \leq W_1(b_1)$$
A similar statement shows that assignment $A_2$ is feasible as well. The value of the solution is

$$p(S) = \sum_{i \in S} p(i) = \sum_{i \in S} g_{i,2}^+ = \sum_{i \in S_1 \cap S_2} g_{i,2}^+ = f_Q((S, S)),$$

which concludes the proof.

In [60] Kulik and Shachnai showed that there is no EPTAS for 2-KP unless $W[1] = FPT$. Theorem 1.5.2 follows immediately from the hardness result of [60] and Lemma 8.3.2.
Chapter 9

Discussion

In this thesis we considered multiple allocation problems, in which we assign a given set of clients (or requests) to a set of servers with limited resources. The goal is either to minimize cost of fully satisfying all clients (for example, the Dynamic NFV Placement Problem), or maximize revenue without violating the resource constraints (for example, the Capacitated MEC Problem). As discussed in Chapter 1, addressing these problems has an immediate impact on the development of new networks, by enabling an efficient and optimized usage of novel network paradigms.

We show that these problems are NP-hard, and provide proven lower bounds. In some cases, these bounds hold even for simpler special cases of the problems. We accompany these results with algorithms with analytically proven approximation ratios. In addition, for some of the problems we present an empiric evaluation based on extensive simulation, proving the advantage of our solutions over currently used heuristic in realistic scenarios.


180


A group of customers and a group of servers. In addition, we have a function of distance between customers and servers and a price function for the initial operation, or startup, of servers. In solving the problem, a subset of the servers will open, and every customer will be connected to an open server. The goal is to find a solution that minimizes the total startup cost and the total distance between customers and servers to which they are connected.

For the Dynamic Facility Location problem, we present an algorithm with a constant approximation ratio. Additionally, we extend this model so that it includes multiple services. We define the Dynamic UNFV problem, where an upper limit on the servers is imposed, thereby limiting the number of services that can be installed on each server. In the Dynamic CNFV problem, in addition to the capacity limit, we define a capacity constraint that limits the number of customers that each service can serve. For both problems, we present algorithms with constant approximation ratios and polynomial running times.

In addition, we present empirical experiments that prove the advantage of our algorithm over existing heuristics and previous solutions to the problem.

The problem of Capacitated MEC is defined by a similar generalization for the Maximum Matching problem. In this problem, we are given a graph with two sides, and the goal is to select a maximum matching of edge-disjoint edges from the graph. In the Capacitated MEC graph, the sides are defined as a group of customers and a group of servers. Each customer has a profit and a service he requires, and each server has a capacity limit that limits the number of services that can be installed in it. Additionally, each service has a capacity limit that limits the number of customers it can serve. In the solution to the problem, we need to choose which services will be installed in each server, without violating its capacity limit, and connect customers to the services, without violating their capacity limits. The goal is to maximize the profit from the customers who are served.

We present three algorithms for the problem with constant approximation ratios and running times.

In addition, we present empirical experiments proving the advantage of our algorithms over existing heuristics and previous solutions to the problem. In the LastStage, we deal with the problem of VM Scheduling, a generalization of the Bin Packing problem. In this problem, we are given a group of products that need to be stored in the minimum number of bins. Each product has a size, and each bin can store exactly one product.

First, we will consider the offline VM Scheduling problem, which can also be defined as a multistage Bin Packing problem, where the cost of changing solutions between successive stages is infinite. Therefore, we cannot apply changes. For this problem, we present an algorithm with a nearly optimal approximation ratio, which depends on the total size of the products.
The recent generations of cellular networks promise wide coverage and short latency, features that provide better service levels and enable the development of new businesses. The development of this was partially enabled by the development of new communication paradigms such as virtualization of communication functions (NFV) and edge computing (MEC).

The core idea behind NFV is to separate the functional from the dedicated material to create flexible networks where service location can change dynamically. On the other hand, MEC allows to move services from large central locations to network edge points where resources are limited due to short time delays.

In order to exploit these paradigms and harness their potential, new resource allocation algorithms are needed. In this work, we focus on two general categories of problems to be considered:

1. The first category, Many-Service, deals with the division of limited resources between existing services (required by customers).
2. The second category, Dynamic, focuses on the nature of the network and its demands.

We present two different ways of expanding classical problems for the management of many services. In the first approach, we define a function of services for each customer, who decides what services they require. In addition, there is a dependency between services, so resources cannot be divided between services independently. For example, the definition of a location for servers utilizes the limited resources assigned to each server and prevents installation (impossible in real-time) of each service on a single server.

A different way is to define a non-linear revenue function that can represent more complex tariff models for multiple services. Specifically, the revenue function is submodular, a natural function that appears in many practical problems.

We use two popular models for the second category, Dynamic. First, we use the Multistage (also known as Dynamic) model, an offline model where all data about the problem is known in advance along the time axis. The goal is to provide a solution for each event in a sequence of static optimization problems that also minimizes the deviation between solutions of successive events.

The second model, the online model, is defined along the time axis. At first, problem data is not available, but becomes available with the passage of time. Each new data point represents a part of the optimization problem that needs to be solved before additional data is received. This model fully models the dynamic world because it does not assume knowledge of the future, which can often be estimated (for example through statistical methods).

In this work, we present algorithmic solutions for several problems that fall into these categories. First, we extend the Facility Location problem to a dynamic world using the Multistage model. In the Facility Location problem, data is known, and the goal is to choose the locations of servers in order to minimize the overall cost. In the dynamic case, data is not available, and the goal is to choose locations over time, taking into account the changes in data over time.
המחק נעשה בהנחיית פרופ' ספי זאэр ופרופ' דני רז בפקולטה למדעי המחשב.

אני מודה לטכניון על התמיכה הכספית הנדיבה למдуш של המחשה.

אני מודד לטכניו על ההמכו הכספים הגדירות בהשכלה הרציטית.
דיימויו והשירותים מרובים ברשתות

히ובר על מחקר
לשם مليוי חלק של הדרישות לקבלת התואר דוקטור לפילוסופיה

ירון פיירסטיין

הוגש לסנט הטכניון, מכון טכנולוגי לישראל
כחליל התשש"ב, חיפה, נובמבר 2021