Multi-Dimensional Commodity Covering for Tariff Selection in Transportation*

Felix G. König[1], Jannik Matuschke[2], and Alexander Richter[2]

1 **TomTom International BV**
An den Treptowers 1, 12435 Berlin, Germany.
felix.koenig@tomtom.com

2 **Technische Universität Berlin, Institut für Mathematik**
Straße des 17. Juni 136, 10623 Berlin, Germany.
{matuschke,arichter}@math.tu-berlin.de

Abstract

In this paper, we study a multi-dimensional commodity covering problem, which we encountered as a subproblem in optimizing large scale transportation networks in logistics. The problem asks for a selection of containers for transporting a given set of commodities, each commodity having different extensions of properties such as weight or volume. Each container can be selected multiple times and is specified by a fixed charge and capacities in the relevant properties. The task is to find a cost minimal collection of containers and a feasible assignment of the demand to all selected containers.

From theoretical point of view, by exploring similarities to the well known SetCover problem, we derive $\mathcal{NP}$-hardness and see that the non-approximability result known for set cover also carries over to our problem. For practical applications we need very fast heuristics to be integrated into a meta-heuristic framework that—depending on the context—either provide feasible near optimal solutions or only estimate the cost value of an optimal solution. We develop and analyze a flexible family of greedy algorithms that meet these challenges. In order to find best-performing configurations for different requirements of the meta-heuristic framework, we provide an extensive computational study on random and real world instance sets obtained from our project partner 4flow AG. We outline a trade-off between running times and solution quality and conclude that the proposed methods achieve the accuracy and efficiency necessary for serving as a key ingredient in more complex meta-heuristics enabling the optimization of large-scale networks.

1998 ACM Subject Classification G.1.6 Optimization, G.2.3 Applications

Keywords and phrases Covering, Heuristics, Transportation, Tariff Selection

Digital Object Identifier 10.4230/OASIcs.ATMOS.2012.58

1 Introduction

One of the most important fields of application of combinatorial optimization is the subject of transportation logistics. Besides location and routing decisions, the correct choice of transportation modes and corresponding tariffs plays a crucial role in this context, as real-world transporation tariffs can be of quite complex nature, often depending on the specific features of the freight being transported.

* This work was supported by the European Regional Development Fund (ERDF) and is part of a joint research project with 4flow AG, Berlin, Germany.

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti; pp. 58–70

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we consider the *multi-dimensional commodity covering problem* (MDCC), an extension of the classical SetCover problem, which occurs as an important subproblem in a transportation model recently developed within a joint research project with supply chain management consulting company 4flow AG. In this model, commodities are defined by their extensions w.r.t. various properties, such as weight or volume, and have to be transported through a network. On each transport link in the network, a collection of different containers is available for transporting these commodities. The concept of containers models complex tariff structures and each container type varies in price and capacities for the respective properties. Compared to the global objective of optimizing the routes of flow in the network, MDCC takes a local perspective, optimizing transport costs on each individual link: Given a vector of demand for each commodity to be transported on a particular link, find a cost-minimal selection of containers and an assignment of the demand, such that all demand can be transported along the link without violating the capacity of any container.

From a theoretical point of view, MDCC is closely related to classical covering problems. We will see later that corresponding non-approximability results for these problems carry over to MDCC. From a practical point of view, analogies to minimum knapsack and multi-dimensional knapsack problems contributed to the terminology and design of the algorithms presented. In our algorithmic context, meta-heuristics concerned with optimizing the global flow pattern in the network are employing MDCC in two different scenarios: First, given a flow pattern in the network, a good selection of containers has to be found on every flow-carrying link. Second, while scanning for cost efficient routes to forward flow along, the meta-heuristics need good estimates on the cost incurred by sending a particular demand along a link. These latter scanning steps are performed very frequently (easily more than a million times during the optimization of a single network) and therefore need to be carried out even faster—however, note that a feasible solution does not need to be computed in this case. We will address both scenarios with different variants of greedy algorithms that provide an efficient balance of accuracy and speed.

Contribution and structure of the paper In the remainder of Section 1, we give a formal definition of MDCC and provide an overview of related work. In Section 2, we show that MDCC cannot be approximated better than by a factor logarithmic in the number of properties. In Section 3, we introduce several efficient heuristics for MDCC based on a unified greedy framework. The variants are tailored for fulfilling different requirements on solution quality and speed. These methods are evaluated in Section 4 within a computational study conducted both on an extensive set of instances arising from computations on real-world transportation networks, as well as an additional set of randomly generated instances. The results show that our algorithms enable the efficient evaluation of complex cost functions, which can be used to capture real world tariff systems more precisely in network flow based transportation models while maintaining the compuational tractability of such models.

1.1 Problem formulation

An instance of the multi-dimensinal commodity covering problem (MDCC) is given by a set of properties P, commodities K and container types J. Each commodity $i \in K$ has a demand d_i that has to be transported along the link. Its properties are defined by a vector $\alpha_i \in \mathbb{Q}_+^P$, i.e., one unit of commodity $i \in K$ requires a capacity of α_{ip} for property $j \in P$. Each container $j \in J$ is defined by its capacities $\beta_{pj} \in \mathbb{Q}_+$ w.r.t. each property $p \in P$ and a cost $c_j \in \mathbb{Q}_+$ that is incurred for each copy of container j in use.

The goal is to find a minimum cost selection of container copies together with an assignment of the total demand to those containers such that the capacity of each container suffices to carry the shipment it has been assigned. More formally, for each conainer type $j \in J$, we need to determine the number of containers $y_j \in \mathbb{Z}_+$ to be used, and the amount $x_{ij} \in \mathbb{Q}_+$ of each commodity $i \in K$ to be packed into containers of type j. The solution is feasible, if all demand $d \in \mathbb{Q}_+^K$ is completely assigned, i.e.,

$$\sum_{j \in J} x_{ij} = d_i \quad \forall i \in K \tag{1}$$

and all capacity constraints

$$\sum_{i \in K} \alpha_{ip} x_{ij} \leq y_j \beta_{pj} \quad \forall j \in J,\ p \in P \tag{2}$$

of each container type are satisfied. Thus, we are looking for an optimal solution to the following mixed integer linear program (MIP).

$$\begin{aligned} \min \quad & c(y) := \sum_{j \in J} c_j y_j \\ \text{s.t.} \quad & x, y \text{ fulfill (1) \& (2)} \\ & x_{ij} \in \mathbb{Q}_+^{K \times J},\ y_j \in \mathbb{Z}_+^J \end{aligned}$$

▶ Notation. We call a vector $x \in \mathbb{Q}^K$ with amounts of commodities a *commodity vector*. Similarly, a vector $\kappa \in \mathbb{Q}^P$ with amounts for each property is called *property vector*.

For $x \in \mathbb{Q}^K$ we define the *aggregated properties* by $\kappa_p(x) := \sum_{i \in K} \alpha_{pi} x_i$ for all $p \in P$.

▶ Remark. We can assume that $\kappa_p(d) > 0$ for all properties $p \in P$ since otherwise we could delete such a property from the problem instance. We also assume w.l.o.g. $\beta_{pj} \leq \kappa_p(d)$, since no container needs to have more capacities than demanded. Furthermore, note that due to the fractional assignment of commodities there are two degrees of freedom w.r.t. scaling:

- For some single property $p \in P$, scaling at the same time all α_{pi} and β_{pj} by some factor $\mu > 0$ yields an equivalent MDCC instance.
- For some single commodity $i \in K$, scaling at the same time d_i by some factor $\nu > 0$ and all α_{pi} by $1/\nu$ yields an equivalent MDCC instance.

1.2 Related work

The special case of MDCC where each commodity has only one non-zero property is known as *covering integer programming*. In this case, the assignment of commodities to containers can be completely removed from the problem formulation by *aggregating* Equalities (1) and Inequalities (2) for each property:

$$\sum_{j \in J} \sum_{i \in K} \alpha_{pi} x_{ij} \leq \sum_{j \in J} y_j \beta_{pj} \underset{(1)}{\implies} \kappa_p(d) = \sum_{i \in K} \alpha_{pi} d_i \leq \sum_{j \in J} y_j \beta_{pj} \tag{3}$$

It then suffices to cover the aggregated properties of the demand with container capacities.

The most prominent special case of this setting is the well-known SetCover problem: Given a ground set S and a set family $\mathbb{S} \subseteq 2^S$ with costs $c : \mathbb{S} \to \mathbb{Q}_+$, find a minimum cost subset of $\mathbb{F} \subseteq \mathbb{S}$ such that $\bigcup_{F \in \mathbb{F}} F = S$. In the context of MDCC, this corresponds to the case where additionally all input data is restricted to be in $\{0, 1\}$. We establish a formal reduction of SetCover to MDCC in Section 2. Chvatal [2] analyzed a greedy algorithm

for SetCover that achieves an approximation ratio of $H(D)$ where D is the dilation of the instance, i.e., the size of the largest set in $\mathbb{S}$, and $H(n)$ denotes the nth harmonic number, i.e., $H(n) := \sum_{k=1}^{n} 1/k$. Feige [5] showed that this approximation ratio is essentially best possible from a theoretical point of view, as the existence of an $o(ln(|S|))$-approximation algorithm for SetCover implies $\mathcal{P} = \mathcal{NP}$.

For general covering integer programs, Dobson [4] devises a combinatorial algorithm that achieves an approximation factor of $\max_{j \in J} \left\{ \log \left(\sum_{p \in P} \beta_{pj} \right) + 1 + H(n_j)) \right\}$, where n_j is the number of nonzero capacities in container j. In [11] and [12], Srinivasan proposes a different algorithm based on randomized rounding involving the width of the problem defined as $W := \min_{p \in P,\, j \in J} \{\kappa_p(d)/\beta_{pj} : \beta_{pj} > 0\}$ and an adapted definition of the dilation D as $D := \max_{j \in J} n_j$. The author derives a $(1+O(\max\{\ln(D+1)/W, \sqrt{\ln(D+1)/W}\}))$-approximation algorithm and also gives "pessimistic estimators" that allow for derandomization of the rounding scheme. Exact algorithms based on dynamic programming are known [6] but suffer from a running time growing exponentially in $|P|$. Yet more different bounds for variations of the problem and for further assumptions on input data have been attained in the last decades, a recent overview is given for example in [8]. In the case of only one property ($|P| = 1$) the problem is referred to as a minimum knapsack problem and a greedy algorithm with performance ratio of 3/2, which can be extended to a (not fully) polynomial time approximation scheme [3]. Also a primal-dual algorithm with performance ratio 2 has been recently explored [1]. Note that all these results only apply to the case where each commodity has only one non-zero property, while for the more general problem studied in this paper, to the best of our knowledge, no results are known up to this point.

2 (Non-)approximability of MDCC

In this section, we derive $\mathcal{NP}$-hardness and non-approximability for MDCC by reduction from SetCover. However, we will also show that under certain conditions on the input data the known results for covering problems can be used to obtain an approximation algorithm for MDCC with a factor depending logarithmically on the number of properties and on a particular variance measure for the instance.

2.1 Hardness of MDCC

Given a SetCover instance $(S, \mathbb{S})$, we can construct a corresponding MDCC instance (K, P, J) such that for any solution for the former instance there is a solution for the latter one with same cost value and vice versa. Indeed, we can model each ground element by introducing a commodity that has only one nonzero associated property, i.e., we chose $K = P = S$. Furthermore, we can model sets $S_j \in \mathbb{S}$ by introducing containers j that have only nonzero capacities for those properties $p \in P$ that are associated with those ground elements contained in S_j. This way, selecting a container $j \in J$ and assigning some commodity $i \in K$ to it corresponds to selecting a set $S_j \in \mathbb{S}$, that covers ground element $e_i \in S$.

▶ **Theorem 1.** *There is a constant $c > 0$ such there is no $c \ln(|P|)$-approximation algorithm for* MDCC *unless $\mathcal{P} = \mathcal{NP}$.*

Proof. Observe that the construction described above is cost preserving, and thus any $f(|P|)$-approximation algorithm for MDCC for some function f immediately implies an $f(|S|)$-approximation algorithm for SetCover. As shown in [5], there is a constant $c > 0$

such that there is no $c\ln(|S|)$ approximation for set cover unless $\mathcal{P} = \mathcal{NP}$. Accordingly, for the same c, there also is no $c\ln(|P|)$-approximation for MDCC unless $\mathcal{P} = \mathcal{NP}$. ◀

Approximation with bounds on input data

In order to achieve an approximation guarantee for the MDCC, we again consider the aggregation of the capacity constraints (3) from Section 1.2. We obtain the following covering integer program as relaxation of the problem, which we denote by A-MDCC:

$$\min\left\{c(y) \;:\; y \in \mathbb{Z}_+^J,\ \sum_{j\in J} \beta_{pj} y_j \geq \kappa_p(d) \quad \forall\, p \in P\right\}.$$

For the special case of each commodity having only one nonzero property discussed Section 1.2, the aggregation results in a more compact but equivalent formulation of the problem. However, in the general case, this is no longer true. In fact, it is easy to observe that the relaxation might allow for feasible solutions even if the original problem is infeasible. Yet we will show that for a certain class of instances, it is possible to construct a feasible solution of MDCC from a feasible solution of A-MDCC. To this end, we definde the *capacity variance* δ of an instance of MDCC by

$$\delta := \max\left\{\frac{\kappa_q(d)\beta_{pj}}{\kappa_p(d)\beta_{qj}} \;:\; p,q \in P,\ j \in J\right\}.$$

Note that $\delta \in [1,\infty]$ and the capacity variance is only finite if $\beta_{pj} > 0$ for all $p \in P, j \in J$. In this case, we can show that the containers selected in a solution of A-MDCC suffice to cover at least an $\frac{1}{\delta}$-fraction of the demand of the original MDCC instance. Combining this result with an $(1 + O(\ln|P|))$-appoximation algorithm for A-MDCC obtained from [12, 11], we derive an $\lceil\delta\rceil(1 + O(\ln|P|))$-approximation to MDCC instances with finite capacity variance.

▶ **Lemma 2.** *For an instance of* MDCC *with finite capacity variance, let y be a solution to the corresponding* A-MDCC *instance. Then there is an $x \in \mathbb{Q}_+^{K\times J}$ such that $\kappa_p(x_j) \leq \beta_{pj} y_j$ for all $j \in J$, $p \in P$ and $\sum_{j\in J} x_{ij} \geq \frac{1}{\delta} d_i$ for all $i \in K$.*

▶ **Theorem 3.** *There is an $\lceil\delta\rceil(1+O(\ln|P|))$-approximation algorithm for* MDCC *restricted to instances with finite capacity variance.*

A detailed proof of Lemma 2 and Theorem 3 can be found in the extended version of this paper [9].

3 A heuristic greedy framework

In this section we present a framework of greedy algorithms to heuristically produce near-optimal solutions for instances of MDCC within very short running-time. Algorithmic requirements are twofold: Due to the integration into a meta-heuristic framework our algorithms have to solve up to 2 million MDCC instances or at least estimate their cost within one meta-heuristic run, which requires them to be extremely fast. Furthermore, our project partner favors algorithmic variants that run without any third party licensed software, such as MIP- or LP-solvers. We emphasize that some of our methods are specifically designed to cope well with the given practice instances: In those instances all properties and capacities are strictly positive, they are always feasible and the number of properties is small. Though some of the following algorithms also work with zero-valued properties or capacities and

feasibility tests could be easily incorporated, we omit the explicit treatment of these issues for the sake of readability.

The general outline follows a natural greedy approach that shares similarities with the concepts applied to integer covering problems as presented for example in [4]. We denote with $\bar{d}$ the remaining commodity demand to be covered. The generic greedy algorithm (formally denoted as Algorithm 1 below) repeatedly selects an "efficient" container $j \in J$ to cover portions of, or the whole remaining demand $\bar{d}$. It uses generic methods `Score` and `Fill` for which we will devise different variants: `Score`$(j,\bar{d})$ returns a value that reflects a measure of efficiency of container j with respect to the remaining demand $\bar{d}$, i.e., the amount of demand covered by the container compared against its cost. `Fill`$(j,\bar{d})$ returns a vector Δ of commodities $0 \leq \Delta \leq \bar{d}$ that uses container j at maximal capacity and can then be assigned to it. These two methods represent the computational bottleneck: Both performance and efficiency of the generic greedy algorithm depend on the detailed implementation of these methods. Note that for a given `Fill` implementation, a corresponding `Score` method can be defined as the amount of aggregated properties of covered commodities per container cost:

$$\texttt{Score}(j,\bar{d}) := \frac{1}{c_j} \cdot \sum_{p \in P} \kappa_p(\texttt{Fill}(j,\bar{d}))\,.$$

However, this makes the task of scoring a container as computationally expensive as the task of filling a container and we will discuss other possibilities in the following paragraphs.

Algorithm 1 repeats until $\bar{d}$ reduces to zero and all demand is covered. However, at some point in the execution there might be containers large enough to cover $\bar{d}$ as well as containers that cover only fractions of $\bar{d}$. In such situations both outcomes are considered: we branch a separate complete solution with a large container selected, if it improves upon the best known solution (lines 5 to 8), and continue considering the incomplete partial solution with a smaller container selected, if its cost does not exceed the best known cost. To speed up the algorithm we can assign the computed mix of commodities Δ multiple times to copies of the same container, as long as there is enough remaining demand $\bar{d}$ to assign Δ completely (line 11). This is convenient if the `Score` function depends on the computed filling but for heuristic `Score` functions this might imply that some of the container copies are selected although there was another container with higher score. To simplify notation we associate a multiset Y over J with a possible solution vector $y \in \mathbb{Z}_+^J$ that contains y_j copies of container $j \in J$ and denote with $c(Y)$ the respective selection cost $c(y)$. In the following paragraphs we introduce and discuss different `Score` and `Fill` methods including a variant where `Score` does not depend on `Fill`.

3.1 LP-based filling of containers

The most versatile approach for designing a `Fill` method is to use linear programming. The objective function for the linear program (LP) is then to minimize the sum of slack in all capacity constraints. We introduce slack variables s_p for each property $p \in P$, and obtain the following LP-formulation for a fixed container $j \in J$ and a given remaining demand vector $\bar{d}$:

$$\begin{aligned} \min \quad & \sum_{p \in P} s_p \\ \text{s.t.} \quad & \sum_{i \in K} \alpha_{pi} \Delta_i + s_p = \beta_{pj} \quad \forall\, p \in P \\ & 0 \leq \Delta_i \leq \bar{d}_i \quad \forall\, i \in K, \qquad s_p \geq 0 \quad \forall\, p \in P \end{aligned} \tag{4}$$

Algorithm 1: Generic Greedy Algorithm (GGA)

Input: MDCC instance (K, P, J) with initial demand d
Output: assignment commodity vectors $x'_j \in \mathbb{Q}_+^K$, $j \in J$, multiset Y' over J

$\bar{d} \leftarrow d$; // remaining uncovered demand
$(x_j)_{j\in J} \leftarrow 0$; $Y \leftarrow \emptyset$; // current partial solution
$(x'_j)_{j\in J} \leftarrow 0$; $Y' \leftarrow \emptyset$; // current best complete solution
while *there is uncovered demand* $\bar{d}$ **do**
  // consider separate complete solution
  if *there exists* $j_F = \operatorname{argmin}_{j\in J:\kappa(\bar{d})\le\beta_j} c_j$ **then**
    if $Y' = \emptyset$ *or* $c(Y \cup j_F) < c(Y')$ **then** // found new best solution?
      replace Y' with $Y \cup j_F$ and $(x'_j)_{j\in J}$ with $(x_j)_{j\in J}$;
      $x'_{j_F} \leftarrow x'_{j_F} + \bar{d}$; // update new best solution
  $j_B \leftarrow \operatorname{argmax}_{j\in J}$ `Score`$(j, \bar{d})$; // pick most efficient container
  $\Delta \leftarrow$ `Fill`$(j_B\ ,\ \bar{d})$; // compute mix of commodities to assign
  $\mathsf{n} \leftarrow \left\lfloor \min_{i\in K:\Delta_i\neq 0} \frac{\bar{d}_i}{\Delta_i} \right\rfloor$; // compute multiplicity of assignment
  $Y \leftarrow Y \cup_{i=1\cdots\mathsf{n}} \{j_B\}$; // add container copies
  $x_{j_B} \leftarrow x_{j_B} + \mathsf{n} \cdot \Delta$; // update assigned commodities
  $\bar{d} \leftarrow \bar{d} - \mathsf{n} \cdot \Delta$; // compute remaining uncovered demand
  if $Y' \neq \emptyset$ *and* $c(Y) \ge c(Y')$ **then**
    return x'_j, Y'; // complete solution dominates partial solution

This method is the most versatile because it easily copes with properties or capacities of value 0 and infeasible instances can be detected whenever there is remaining demand and no container produces a nonzero filling. Furthermore, the minimization of slack leads to efficient utilization of containers. However, a drawback of the method lies in the computational effort, since a distinct LP for each container has to be solved in every iteration of Algorithm 1.

3.2 Greedy filling method

In order to achieve good container utilization without solving LPs, we devise a two-phase greedy method. Both phases successively select and add commodities to a given container j so as to approximate the capacity vector β_j of the container with the aggregated properties $\kappa(\Delta)$ of the assigned commodity vector Δ. While Phase 1 can deal with zero-entries in property and capacity vectors and can be used to detect infeasible instances, Phase 2 can only be applied to instances with all nonzero properties and capacities.

The first phase adds commodities that minimize the residual capacity until one of the capacity constraints becomes tight or the demand of every commodity is depleted. Assuming that some commodity demands have already been added to Δ let $\bar{\beta}_j$ be the vector of residual capacities of container j w.r.t. Δ. For any given vector of commodities $\delta \in \mathbb{Q}_+^K$, we denote with `linFrac` the maximal fraction of δ that can be feasibly and uniformly assigned to a container with residual capacities $\bar{\beta}_j$, defined by:

$$\texttt{linFrac}(\delta,\ \bar{\beta}_j) := \min_{p\in P:\kappa_p(\delta)\neq 0} \bar{\beta}_{pj}/\kappa_p(\delta)\,.$$

Now the algorithm successively chooses a commodity i that minimizes the Euclidian norm of

the vector of slacks after maximal feasible assignment of this commodity, i.e.,

$$i = \operatorname*{argmin}_{i' \in K} \|\bar{\beta}_j - \min\{\texttt{linFrac}(\bar{d}^{i'}, \bar{\beta}_j), 1\} \cdot \kappa(\bar{d}^{i'})\|$$

where $\bar{d}^i = (0, \dots, \bar{d}_i, \dots, 0)$, and adds this amount of commodity i to the current vector Δ.

Phase 1 might incur an unnecessary slack in some capacities due to the greedy choice of commodities. To improve on this, Phase 2 minimizes slack by focusing on a good mix of assigned commodities: It adjusts the current Δ so as to approximate the ray induced by the capacity vector $\beta_j \in \mathbb{Q}^P$ with a conic combination of property vectors α_i of the available commodities. More formally, we decompose the property space $\mathbb{Q}^P = V(\beta_j) + V(\beta_j)^\perp$ into the linear subspace $V(\beta_j)$ spanned by the capacity vector β_j and its orthogonal complement and consider for each commodity i the unique decomposition of its property vector $\alpha_i = v_i + u_i$ with $v_i \in V(\beta_j)$ and $u_i \in V(\beta_j)^\perp$. The current commodity mix $\Delta \in \mathbb{Q}_+^K$ induces the property vector $\sum_{i \in K} \Delta_i \alpha_i = \sum_{i \in K} \Delta_i v_i + \sum_{i \in K} \Delta_i u_i \in \mathbb{Q}_+^P$. Our goal of approximating the ray spanned by β_j corresponds to minimizing the orthogonal deviation $\|\sum \Delta_i u_i\|$. For commodity $\ell \in K$, we define $\lambda_\ell := \langle \sum \Delta_i u_i, u_\ell \rangle / \|u_\ell\|^2$. Note that $\lambda_\ell u_\ell$ corresponds to the projection of $\sum \Delta_i u_i$ on $V(u_\ell)$. If $\lambda_\ell < 0$, we augment Δ by $\min\{-\lambda_\ell, \bar{d}_\ell\}$ units of commodity ℓ, which leads to a decrease of the orthogonal deviation. We iteratively augment Δ in this way until no additional improvement can be achieved by any commodity. Note that the resulting vector Δ might violate container capacities. We therefore scale Δ down to feasibility.

3.3 Fraction based scoring

From practical point of view, another computational bottleneck concerning running time is the `Score` method, because `Score` has to be computed for every container in each iteration of Algorithm 1. As mentioned above, a `Score` method can be defined depending on any of the previous `Fill` methods, making `Score` as computationally expensive as `Fill`. Instead one can define a less accurate but significantly faster method by $\texttt{Score}(j,\bar{d}) := (1/c_j)\texttt{linFrac}(\bar{d},\beta_j)$. Note that this scoring only makes sense if all container capacities are strictly positive and recall that $\texttt{linFrac}(\bar{d},\beta_j)$ is the fraction of the remaining demand $\bar{d}$ that can be uniformly assigned to the container. This fraction can be small compared to the maximum assignable value of a single commodity. We will outline the tradeoff between running time and solution quality due to a less accurate `Score` in Section 4.

3.4 Theoretical bound on running time

In order to obtain a theoretical bound on the running time of the heuristic, we again restrict to instances with finite capacity variance and observe that every of the presented `Fill` methods either returns a filling with at least one tight capacity constraint or all remaining demand $\bar{d}$ is used. This observation can be used to bound the total number of chosen containers and to show pseudo-polynomial running time (see [9] for a formal proof).

▶ **Theorem 4.** *The generic greedy algorithm (Algorithm 1) has pseudo-polynomial running time for instances finite capacity variance.*

Note that this theoretical worst-case bound does not yield any information on the practical running time of the greedy algorithm on real world instances. In fact, our computational results in Section 4 reveal this practical running time to be extremely low.

3.5 Cost estimators

In some applications of MDCC it is not important to compute an assignment of commodities to containers, but merely an estimate on the incurred cost. Examples include shortest path type algorithms where nodes are to be labeled with the cost of forwarding a given amount flow to them. Hence, it is useful to investigate whether an algorithm for MDCC can be accelerated when setting the actual choice of containers aside and only focusing on (approximate) cost. Again, both presented estimators rely on all strictly positive properties and capacities, as present in practice instances.

Covering relaxation (`CR`) We again consider relaxation A-MDCC from Section 2 to obtain a very fast cost estimation. Recall its formulation:

$$\min \left\{ c(y) \;:\; y \in \mathbb{Z}_+^J,\ \sum_{j \in J} \beta_{pj} y_j \geq \kappa_p(d) \quad \forall\, p \in P \right\}.$$

We can heuristically solve this problem very efficiently by adjusting Algorithm 1 to directly operate on a property vector $\bar{\kappa} \in \mathbb{Q}_+^P$ of residual uncovered properties instead of on the residual commodity demand vector $\bar{d}$, i.e., we reduce $\bar{\kappa}$ by β_j for each selected container copy of type j. An appropriate scoring function with respect to $\bar{\kappa}$ can be defined by $\texttt{Score}(j, \bar{\kappa}) := 1/c_j \cdot \min_{p \in P} \left\{ \beta_{pj}/\bar{\kappa} \;:\; \bar{\kappa}_p > 0 \right\}$. As mentioned before, a solution to A-MDCC does not necessarily yield a feasible solution for MDCC. However, for many real-world instances, applying the adjusted variant of Algorithm 1 to CR proves to be a reasonable estimate (see Section 4).

One dimensional covering restriction (`CR1D`) Like the covering relaxation, the one dimensional covering restriction eliminates the computational effort arising from flow assignment to containers. Differently from the above, however, we restrict a container's filling to the maximal uniformly assignable fraction of the total demand d. More formally, each container $j \in J$ is assigned a weight $w_j := \min(1, \texttt{linFrac}(d,\, \beta_j))$ and the resulting one dimensional covering problem, also known as minimum knapsack problem, can be written as:

$$\min_{y \in \mathbb{Z}_+^J} \left\{ \sum_{j \in J} c_j y_j \mid \sum_{j \in J} y_j w_j \geq 1 \right\}.$$

We observe that by the definition of w_j, a feasible solution to the above formulation always yields a feasible solution to MDCC. To quickly obtain a heuristic solution, we can apply a greedy algorithm given in [3] that is very similar to Algorithm 1 and has performance guarantee two. Also, we benefit from significant speedups: At first, all containers can be presorted in order of non-increasing score values defined as c_j/w_j and therefore need to be considered exactly once in this order. Second, we omit any calls to `Score` or `Fill` methods. While there also is an approximation scheme for the covering restriction [3], we emphasize that running time is our major interest here, and therefore restrict to the basic version of the greedy algorithm.

4 Computational study

We now evaluate the efficiency and solution quality attained by the algorithms presented in the preceding section within a computational study on both, instances arising from real-world logistics networks as well as randomly generated instances.

4.1 Test instances

As mentioned in the introduction, the real world instances of MDCC considered in this article arise as subproblems in a fixed charge multi-commodity network flow model for tactical logistics optimization. Our project partner 4flow AG provided an extensive library comprised by 145 networks aggregated from four recent and on-going customer projects in three different industries (automotive, chemical, retail). Given a fixed flow pattern in such a network, an optimal tariff selection for each link that carries flow has to be found and each such link contributes an individual MDCC subproblem. Naturally, meta-heuristics consider a vast amount of different intermediate flow patterns in their solving process, but we could observe that the final, i.e., locally optimal flow patterns already introduce a representative subset of those instances. We therefore restrict our study to MDCC instances arising from final flow patterns obtained by various meta heuristics.

We observed that many of those instances are very easy to solve in the sense that they allow for an optimal solution with only a single container. Note that for those instances an optimal solution is always considered (and hence found) in line 5 of Algorithm 1. In order to prevent numerous easy instances from dominating the outcome of the study we removed these instances from the test set. The resulting instance sets are called Auto1, Auto2, Chemical and Retail with 581, 647, 2867 and 4600 MDCC instances respectively. Though the number of commodities present in the networks varies between 50 and 500, in extracted MDCC instances only 10 commodities are found in averages. The number of different container types varies from 6 to 38. In each instance two properties—mass and volume—are present.

We also generated three sets of random instances following some of the steps outlined in [10] to test the performance of our algorithms on instances with more than two properties and different other modifications. The main differences in performance are visible when considering different numbers of properties. We therefore aggregate all instance sets with the same number of properties and obtain sets “RandP2”, “RandP4” and “RandP8” with 2, 4 and 8 properties respectively, each containing 960 instances (see [9] for more details).

4.2 Tested configurations

We tested three promising configurations of the greedy framework, denoted by `1P/LP`, `1P/2P`, and `Fr/2P`, as well as the two cost estimators `CR` and `CR1D` from Section 3.5. From previous tests we could infer that using either LP-Based filling from Section 3.1 or the two-phase greedy `Fill` method described in Section 3.2 for the scoring task exceeds acceptable running times. Thus, Configurations `1P/LP` and `1P/2P` employ only the first phase of the greedy filling algorithm to compute `Score` from the resulting filling. `1P/LP` then computes a feasible filling using the LP-based method, while `1P/2P` uses the two-phase greedy method for this task. The third configuration, `Fr/2P`, uses the fraction based scoring method together with two-phase greedy filling. We compare the solutions computed by our solvers to the corresponding near-optimal solutions obtained by solving the MIP formulation of MDCC and report the gap in percent. Note that for the `CR` configuration, it is possible to underestimate the optimal cost value. Whenever this happens, we compute the negative gap to the `CPLEX` solution cost and consider its absolute value for averaging.

Algorithms have been implemented in C++ and compiled with gcc 4.6.2 on SuSE 12.1 Linux with kernel 3.1.0-1.2. Computations have been performed on a desktop machine with an Intel Core Duo CPU (3.00 GHz, 64 bit) and 4 GB of memory (note that our implementations make use of only one thread). We measure the running time as CPU user time and the given values denote seconds. Since solving time for each individual instance lies within a few

Table 1 Optimality gaps and computation times of solvers and estimators on practice instances.

	1P/LP		1P/2P		Fr/2P		CR		CR1D	
Instance Set	Av-Gap	Time	Av-Gap	Time	Av-Gap	Time	Av-Gap	Time	Av-Gap	Time
Auto1	2.144	1.96	3.153	0.73	17.900	0.18	18.695	0.02	28.442	<0.01
Auto2	0.222	1.67	0.228	0.49	0.334	0.13	0.548	0.01	0.801	0.01
Chemical	0.053	6.56	0.053	0.56	0.053	0.43	0.053	0.03	0.053	0.03
Retail	0.074	9.17	0.074	1.72	0.074	1.11	0.074	0.04	0.074	0.05

Table 2 Optimality gaps and computation times of solvers and estimators on random instances.

	1P/LP		1P/2P		Fr/2P		CR		CR1D	
Instance Set	Av-Gap	Time	Av-Gap	Time	Av-Gap	Time	Av-Gap	Time	Av-Gap	Time
RandP2	1.801	63.59	4.176	50.25	3.736	5.56	3.731	0.13	7.341	0.03
RandP4	5.501	70.07	11.140	100.19	9.950	11.28	5.191	0.20	21.016	0.03
RandP8	8.128	102.77	33.751	159.67	33.509	14.83	8.273	0.36	46.291	0.03

hundredths of a second, we measure only the accumulated running time needed to solve a whole instance set to avoid that inaccuracies distort the outcome. Furthermore, to rule out possible errors due to system fluctuations we run every test ten times and report the average time needed. The MIP formulation as well as the filling LP from Section 3.1 are solved with CPLEX 12.1 [7]. Since proving optimality of MIPs is very time consuming for some of the given instances, we impose a time limit of 10 seconds per instance which turned out to be sufficient to achieve optimality gaps close to zero (see [9] for more details).

4.3 Results

Table 1 shows the averages over all gaps achieved by the respective solver for practice instance sets. We can observe that the 1P/LP configuration achieves best solution quality with an average gap of less than three percent on any set but has an up to 10 times longer running time compared to the similar 1P/2P configuration. The latter one performs equally on practice instances except for the Auto1 set. Fr/2P can yet improve on this running time by up to 50% but looses significantly in solution quality on Auto1 instances. When looking closer on Auto1 instances we found a larger variance in the densities of the commodities as well as the highest average number of container types. While 1P/LP and 1P/2P cope well with this more challenging setting and still produce reasonably small gaps, the estimators and Fr/2P are less robust. We also note that all solver configurations perform equally well on Chemical and Retail instances and the differences on the Auto2 set are marginal. The two estimators CR and CR1D perform well on Chemical and Retail instances with results close to the optimum cost value, while they achieve speed up factors of roughly ten compared to the fastest heuristic solver. Deviations are marginally larger on Auto2 instances, however they become obvious with average deviation of up to 30% on Auto1.

On the random instance sets (cf. Table 2), again solver 1P/LP achieves best solution quality and is even faster than 1P/2P for sets with more than two properties. One reason might be that Phase 2 of the greedy filling employed in 1P/2P considers all commodities before it chooses one to be added to the current filling and that the average number of commodities is much larger on random instances than on practice instances. Surprisingly the roughly ten times faster Fr/2P solver achieves even slightly better solution quality than 1P/2P. We conclude that on random instances the more sophisticated scoring method used

for 1P/2P does not pay off compared to heuristic scoring of Fr/2P. But in general, the impact of less accurate heuristic filling on solution quality compared to the accurate LP based filling grows with the number of properties. Estimators are significantly faster and achieve speed up factors between 10 and 100 compared to the fastest heuristic Fr/2P. Despite this enormous savings in running time, CR still achieves reasonably good cost estimates.

5 Conclusions and outlook

MDCC is a generalized covering problem that serves as a key component in a larger transportation logistics model. We have studied this problem from theoretical and practical perspective and developed different algorithmic approaches, whose quality we validated on a broad set of practice as well as random instances.

By exploring connections to other covering problems, we established both a lower bound on the achievable approximation factor of this problem, as well as an approximation algorithm whose factor depends on the numerical structure of the input. In order to solve the problem in practice, we developed a framework of greedy algorithms that is configurable for various needs of the meta heuristic solvers it serves as a sub-routine:

- If accurate solutions are desired, the configuration 1P/LP has the best performance on most instance sets but relatively slow running time.
- If good solution quality is sufficient, the configurations 1P/2P and Fr/2P run significantly faster and produce good solutions on most instance sets.
- If only estimates of the optimal solution values are needed, CR produces such estimates with acceptable deviation of cost in very fast running times.

The MDCC in conjunction with our heuristic toolbox enables the design of transportation models that capture complex tariff structures and at the same time remain accessible to established optimization procedures.

Outlook While the reduction from SetCover in Section 2 resulted in a high number of properties, this number is usually low in practical applications. Future research will investigate the possibility of better—possibly constant factor—approximations for the special case of a fixed property dimension. In order to capture more complex cost functions such as graded linear tariffs, we propose two extensions to the model: shipping commodities may incur an additional linear cost depending on the container they are assigned to, and the number of copies of a particular container type might be bounded. While we hope that the algorithms presented in this work are sufficiently robust to be adaptable to this generalized setting, further experiments are needed to verify this claim.

References

1 T. Carnes and D. Shmoys. Primal-dual schema for capacitated covering problems. In *Integer Programming and Combinatorial Optimization*, pages 288–302. Springer, 2008.

2 V. Chvátal. A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 4(3):pp. 233–235, 1979.

3 J. Csirik, J. B. G. Frenk, M. Labbé, and S. Zhang. Heuristics for the 0–1 min-knapsack problem. *Acta Cybern.*, 10(1-2):15–20, 1991.

4 G. Dobson. Worst-case analysis of greedy heuristics for integer programming with nonnegative data. *Math. Oper. Res.*, 7(4):pp. 515–531, 1982.

5 U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

6 Q. Hua, Y. Wang, D. Yu, and F. C. M. Lau. Dynamic programming based algorithms for set multicover and multiset multicover problems. *Theor. Comput. Sci.*, 411(26–28):pp. 2467–2474, 2010.

7 IBM ILOG CPLEX 12.1. Ref. Manual, 2009. `http://www.ilog.com/products/cplex/`.

8 S. G. Kolliopoulos and N. E. Young. Approximation algorithms for covering/packing integer programs. *J. Comput. Syst. Sci.*, 71(4):495 – 505, 2005.

9 F. G. König, J. Matuschke, and A. Richter. A multi-dimensional multi-commodity covering problem with applications in logistics. *Preprint 009-2012, TU Berlin*, 2012.

10 J. Puchinger, G. R. Raidl, and U. Pferschy. The multidimensional knapsack problem: Structure and algorithms. *INFORMS J. Comput.*, 22:250–265, 2010.

11 A. Srinivasan. An extension of the Lovàsz local lemma, and its applications to integer programming. In *Proc. ACM-SIAM Sympos. Discrete Algorithms*, pages 6–15, 1996.

12 A. Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM J. Comput.*, 29(2):648–670, 1999.