The Transit Time Constrained Fixed Charge Multi-commodity Flow Problem

Erik Hellsten[a*], David Franz Koza[b], Ivan Contreras[c], Jean-François Cordeau[d], and David Pisinger[a]

[a] *Technical University of Denmark - Department of Technology, Management and Economics, Akademivej, Building 358, 2800 Kgs. Lyngby, Denmark*

[b] *Vattenfall A/S, Denmark*

[c] *Concordia University and Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), Montréal, QC, H3G 1M8, Canada*

[d] *HEC Montréal and GERAD, 3000 chemin de la Côte-Sainte-Catherine, Montréal, H3T 2A7 Canada*

Abstract

This paper introduces the transit time constrained fixed charge multi-commodity flow problem. Transit times are origin-to-destination time limits for the commodities, which appear for example in transport systems with perishable goods. We discuss how to model the problem and present three different formulations of it. The first formulation is an exponential size path formulation, which we solve with a branch-and-price algorithm. Several speed up techniques from the literature on fixed charge multi-commodity flow problems are implemented, such as lifted cover inequalities and the recently proposed deep dual-optimal inequalities. In an extensive set of computational experiments, we show that these inequalities significantly improve the performance of the algorithm. The other two formulations are of polynomial size: one uses path indices and the other uses time indices. While the branch-and-price algorithm outperforms solving the compact formulations with a general-purpose mixed-integer programming solver, the study of compact models helps better understand the problem, and we can use them as benchmarks. A detailed sensitivity analysis of the branch-and-price algorithm shows that longer transit times and an increased ratio of fixed charge to flow cost increase the difficulty of solving the problem whereas the arc capacity has less impact. We further discuss in-depth implementational details.

Keywords: Network Design, Branch-and-price, Fixed-charge multicommodity-flow, Transit times

* Corresponding author

email addresses: erik.o.hellsten@gmail.com (E. Hellsten), davidfranz.koza@vattenfall.com (D. F. Koza), ivan.contreras@concordia.ca (I. Contreras), jean-francois.cordeau@hec.ca (J.-F. Cordeau), dapi@dtu.dk (D. Pisinger)

1 Introduction

The *fixed charge multi-commodity flow problem* (FCMCFP), also known as the *capacitated network design problem*, is at the core of many transportation systems. The aim of the problem is to route a set of commodities, from their origins to their destinations, over a directed graph. Each arc has a capacity and carries a fixed charge that must be paid for the capacity to be used. The concept of opening a set of arcs, to route customers or commodities through the resulting network, is essential to model a wide spectrum of problems, for example in ship routing (Agarwal and Ergun, 2008, Brouer et al., 2014, Karsten et al., 2017, Christiansen et al., 2019), train scheduling, bus routing (Guihaire and Hao, 2008) and air transportation planning (Alibeyg et al., 2016). The practical meaning of opening an arc varies significantly between applications and ranges from designing infrastructure (Cordeau et al., 2006) to more short-term decisions, such as routing vehicles in city logistics (Crainic et al., 2009). By using space-time graphs, various scheduling problems can also be modeled as network design problems (Tong et al., 2015, Koza et al., 2020a).

While the FCMCFP has been studied widely, in this work we focus particularly on transit time constraints for the commodities. Transit times appear frequently in practice. In bus scheduling, for example, it is intuitive to consider upper time limits on travel between certain key locations, or else customers will resort to other modes of transportation. Further, in liner shipping, many commodities are perishable and have to be delivered to their destination before they expire. This constraint has been used in network design (Brouer et al., 2015, Akyüz and Lee, 2016) as well as in theoretical studies of multi commodity flow problems (Karsten et al., 2015, Trivella et al., 2020). We will further focus on the FCMCFP with split demand, i.e., where we allow the commodities to be split over multiple paths from their origins to their destinations. When allowing split demand, transit time constraints are of particular interest to study as they then have a significant impact on the properties of the problem.

The basic formulations of the network design problem are known to have very weak linear programming relaxations (Gendron et al., 1999) and many of the branch-and-bound methods rely heavily on cutting planes to strengthen the formulations. Several of those cutting planes are also applicable to the *transit time constrained fixed charge multi-commodity flow problem* (TTFCMCFP), as we will show. It is common to use various decomposition techniques such as Lagrangean relaxation (Gendron et al., 1999, Holmberg and Yuan, 2000, Kliewer and Timajev, 2005) and Dantzig-Wolfe decomposition. Hewitt et al. (2013) use a path-based branch-and-price algorithm to solve the fixed charge multi-commodity flow problem with integer flows. They use local search around the current best solution to find improved upper bounds. Yaghini et al. (2013) propose a hybrid method where the integer variables are updated using simulated annealing, and in each iteration the flow is created using column generation. Instead of using column generation to generate paths, Gendron and Larose (2014) develop a branch-and-price-and-cut algorithm that uses an inspection subproblem to generate the original flow variables. While Benders decomposition has been used for the capacitated FCMCFP (Costa et al., 2009), it is more commonly used for the uncapacitated version, where the subproblems decomposes by commodity (Zetina et al., 2019). Heuristic approaches have taken two main directions: meta-heuristics (Crainic et al., 2001, Crainic and Gendreau, 2007, Ghamlouche et al., 2003, Paraskevopoulos et al., 2016) and heuristics based on mathematical programming (Katayama et al., 2009, Rodríguez-Martín and Salazar-González, 2010, Hewitt et al., 2010, Katayama, 2015).

The TTFCMCFP, when allowing split demand, has no natural polynomial-size model. In this paper we study different approaches to model this problem and present a branch-and-price framework to solve the path formulation. The latter formulation, which is otherwise not commonly used for the FCMCFP, lends itself well to considering transit time constraints, as they can be included as part of the definition of feasible paths.

While time is likely the most common limiting resource for the commodities, the models and results in this paper further generalize to fixed charge multi-commodity flow problems with other resource limitations and multiple limited resources. For example, in liner shipping we might want to limit the number of transshipments, and in airline and bus planning, the number of transfers. In some applications we can also quantify the probability of successfully traversing an edge (Carraway et al., 1990), in which case we could impose a minimum probability of success for each path. An overview of complicating constraints can be found in Reinhardt and Pisinger (2011).

The main contributions of this paper are:

- We introduce and discuss an important extension to the FCMCFP. The transit time constraints make it non-trivial to model the problem in a compact form and we present two approaches to model it compactly under mild additional assumptions.

- The transit time constraints can easily be included into the path formulation and we present a branch-and-price framework to solve it, including the majority of the classic cuts and improvements from the literature on the standard FCMCFP.

- Based on the classic *Canad instances* Crainic et al. (2001), we generate a set of transit time constrained instances, and extensively test the two compact models and the branch-and-price algorithm. We further study how the fixed charges, the arc capacities and the transit time limits affect the solutions and the problem difficulty.

- We show how the use of the recently presented deep dual-optimal inequalities (Koza et al., 2020a) significantly reduces the number of columns generated throughout the branch-and-price tree, which results in greatly reduced run times.

- We do an in-depth sensitivity analysis, and study how particular properties of problem instances affect both the solution procedure and characteristics of the solution.

- Lastly, we discuss several implementational details.

In Section 2 we discuss the different approaches to model the problem and in Section 3 we present some of the core cuts from the FCMCFP literature to strengthen the models. In Section 4 we describe the branch-and-price framework in more detail. Section 5 describes the instances and presents the results. These results show that the path-based branch-and-price algorithm significantly outperforms solving the compact models with a general-purpose MIP solver. The computational experiments also show that the results can be improved significantly by using deep dual-optimal inequalities and lifted cover inequalities. Lastly, in Section 6 we conclude the paper and look at future research directions.

2 Models for the transit time constrained fixed charge multicommodity flow problem

In this section we begin with reviewing the standard FCMCFP and then we introduce different ways of handling the transit time constraints. We further present three models for the TTFCMCFP. First, we review the path formulation, in which the transit time constraints can be included as part of the subproblem, but which is potentially exponential in size. Then we present two polynomial-size formulations, which can be solved more directly with general-purpose MIP solvers.

There are two main formulations for the standard FCMCFP: the *arc formulation* and the *path formulation*. In the first, variables are used to model the flow of the commodities over the arcs. In the second, the arc flows are consolidated into paths, and instead variables representing how much of a commodity flows along specific paths are used.

The main variations of the problem regard whether we allow *rejecting commodities*, as well as various *integrality requirements* on the flows. Rejection is generally modeled by adding a rejection arc, with a large cost and infinite capacity, from each commodity's origin to its destination. Allowing rejection enlarges the feasible region but generally weakens the formulation. It also invalidates some of the strengthening inequalities that we present in Section 3. Integrality requirements most frequently come in two main variations: in the first, each commodity has to use a single path (Hewitt et al., 2013) and in the second the commodities are divided into fixed-sized bundles, where each bundle has to follow a single path (Lin and Kwan, 2013). When adding this kind of integrality constraints, in addition to the fact that the problem generally becomes harder to solve, it implies that the path formulation might have a stronger linear programming relaxation than the arc formulation.

In cases where the flow cost is independent of the commodity, it is also common to work with a so called *aggregated formulation*, where all commodities sharing the same origin, or all commodities sharing the same destination, are aggregated into a single commodity (Gendron et al., 1999, Chouman et al., 2016). This reduces the size of the problem but generally weakens many cuts, such as the strong linking constraints presented in Section 3. In this study we will work with *disaggregated formulations*.

2.1 Modeling the transit time constraints

The difficulty of modeling transit time constraints for the FCMCFP is highly dependent on the problem variation and model chosen. If we assume that each commodity has to follow a single path, then adding transit time constraints can be modeled by adding the following constraints to the arc formulation:

$$\sum_{a \in A} t_a^k x_a^k \leq T_k \qquad k \in K, \tag{1}$$

where t_a^k denotes the time it takes for commodity k to cross arc a, T_k is the maximum transit time for commodity k, and the variables $x_a^k \in \{0, 1\}$ denote whether commodity k flows across arc a.

Similarly, regardless of the level of integrality constraints on the flow variables, in the path formulation it suffices to add the maximum transit time limit to the definition of feasible paths. This has been utilized in standard multi-commodity flow problems by, for example, Karsten et al. (2015) and Holmberg and Yuan (2003).

However, when working with the arc formulation, without full integrality restrictions on the flow variables, constraints (1) are insufficient. They only limit the total sum of transit times for each commodity, instead of limiting the transit time for each individual path. Here we make an important distinction between *average transit time constraints* and *transit time constraints*. They are both useful modeling tools, and which one to use depends on the problem at hand. In this paper we focus on the second of the two: transit time constraints for individual paths.

2.2 The path formulation

The path formulation can be defined using the sets P_k, denoting all feasible paths p from o_k to d_k, for commodity $k \in K$. As we assume the flow costs to be *non-negative* we can further, without loss of generality, restrict the paths to be *elementary*. Let b_a^{kp} be binary coefficients denoting whether or not arc a is used in the path p for commodity k, and let c_{kp} be the costs for such paths. Lastly, let ω_{kp} be continuous variables denoting the proportion of the demand q_k that flows along path p for commodity k. We then define the path formulation as:

$$\min \sum_{k \in K} \sum_{p \in P_k} c_{kp} \omega_{kp} + \sum_{a \in A} d_a y_a \tag{2}$$

$$\text{s.t.} \quad \sum_{p \in P_k} \omega_{kp} \geq 1 \qquad k \in K \tag{3}$$

$$-\sum_{k \in K} \sum_{p \in P_k} q_k b_a^{kp} \omega_{kp} + u_a y_a \geq 0 \qquad a \in A \tag{4}$$

$$\omega_{kp} \geq 0 \qquad k \in K, p \in P^k \tag{5}$$

$$y_a \in \{0, 1\} \qquad a \in A. \tag{6}$$

Here, the objective function consists of the path flow costs and the design costs for opening the arcs. Constraints (3) are set cover constraints, which state that each commodity has to be routed. Constraints (4) are capacity constraints and constraints (5) and (6) define the domains of the variables.

While the arc formulation contains $\mathcal{O}(|A||K|)$ flow variables, the path formulation contains potentially an exponential number of variables, and hence this formulation is often solved with delayed column generation. When including transit time constraints, the subproblems can be formulated as resource constrained shortest path problems which can be solved in polynomial time using specialized algorithms.

2.3 Modeling transit time constraints with explicit path variables

Our first compact model handles the transit times by explicitly indexing the paths. For each commodity k we define an index set S_k for paths for that commodity, which are then designed using the binary decision variables z_a^{ks} denoting whether the path s for commodity k uses arc a. The continuous variables x_a^{ks} represent the flow of commodity k across arc a on path s. The parameters c_a^k denote the flow cost for arc a and commodity k, t_a represent the time it takes to traverse arc a, and T_k denote the maximum transit time for commodity k. The model can be written as follows:

$$\min \sum_{k \in K} \sum_{s \in S_k} \sum_{a \in A} c_a^k x_a^{ks} + \sum_{a \in A} d_a y_a \tag{7}$$

$$\text{s.t.} \quad \sum_{a \in \delta_i^+} x_a^{ks} - \sum_{a \in \delta_i^-} x_a^{ks} = 0 \qquad k \in K, s \in S_k, i \in N \setminus \{o(k), d(k)\} \tag{8}$$

$$\sum_{s \in S_k} \left(\sum_{a \in \delta_{o(k)}^+} x_a^{ks} - \sum_{a \in \delta_{o(k)}^-} x_a^{ks} \right) = 1 \qquad k \in K \tag{9}$$

$$\sum_{s\in S_k}\left(\sum_{a\in\delta^+_{d(k)}} x_a^{ks} - \sum_{a\in\delta^-_{d(k)}} x_a^{ks}\right) = -1 \qquad k\in K \tag{10}$$

$$-\sum_{k\in K}\sum_{s\in S_k} q_k x_a^{ks} + u_a y_a \geq 0 \qquad a\in A \tag{11}$$

$$z_a^{ks} - x_a^{ks} \geq 0 \qquad k\in K, s\in S, a\in A \tag{12}$$

$$-\sum_{a\in A} t_a z_a^{ks} \geq -T_k \qquad k\in K, s\in S \tag{13}$$

$$x_a^{ks} \in [0,1] \qquad k\in K, s\in S, a\in A \tag{14}$$

$$y_a \in \{0,1\} \qquad a\in A \tag{15}$$

$$z_a^{ks} \in \{0,1\} \qquad k\in K, s\in S, a\in A. \tag{16}$$

Constraints (8)-(10) are flow conservation constraints, constraints (11) are capacity constraints, constraints (12) define the relation between the flow variables and the path variables, and constraints (13) are transit time constraints. Lastly, constraints (14)-(16) define the domains of the variables.

This model is conceptually similar to the path formulation in Section 2.2, but it is a polynomial-size model, which can be directly solved with a general MIP solver, where the design of the paths is part of the problem definition. By including the integer path-design variables, we can limit the transit time for each individual path, bypassing the issue with average transit times.

This model has a number of disadvantages. The model contains a large number of binary variables and suffers from issues with symmetry. Further, to guarantee that the model correctly models the problem, the set S_k needs to be at least as large as the number of distinct paths used by commodity k in the optimal solution. However, in most practical applications, the number of distinct paths for each commodity is likely rather small.

As a small note, we observe that there is no guarantee that if a solution uses fewer than the available number of paths for each commodity, then it is optimal for the TTFCMCFP without restrictions on the number of paths.

2.4 Modeling transit time constraints using time indices

The second compact model uses time indices for the flows, as was proposed by Trivella et al. (2020). We introduce a set of time indices $\mathcal{T}$ and let $x_a^{k\tau}$ denote the flow of commodity k over arc a starting at time $\tau \in \mathcal{T}$. A commodity flowing onto arc a at time τ will arrive at the end of a at time $\tau + t_a$.

Next we limit the allowed values for τ. For each arc-commodity combination, let $\mathcal{T}_{ak} \subseteq \mathcal{T}$ be the set of time indices for which commodity k is allowed to flow onto arc a. This reduces the size of the model and the transit time constraints are enforced implicitly by limiting the largest element of $\mathcal{T}_{ak}$ to be less than or equal to $T_k - t_a$, $\forall a \in A, k \in K$. What remains is just to define the flow conservation constraints for the new flow variables, which results in the following model:

$$\min \sum_{k\in K}\sum_{a\in A}\sum_{\tau\in\mathcal{T}_{ak}} c_a^k x_a^{k\tau} + \sum_{a\in A} d_a y_a \tag{17}$$

$$\text{s.t.} \quad \sum_{a\in\delta_i^+:\tau\in\mathcal{T}_{ak}} x_a^{k\tau} - \sum_{a\in\delta_i^-:\tau\in\mathcal{T}_{ak}} x_a^{k(\tau-t_a)} = \xi_i^k \qquad i\in N\setminus\{o_k,d_k\}, k\in K, \tau\in\mathcal{T} \tag{18}$$

$$\sum_{a\in\delta_{o_k}^+} x_a^{k0} = 1 \qquad k\in K \tag{19}$$

$$\sum_{a\in\delta_i^-:\tau\in\mathcal{T}_{ak}} x_a^{k\tau} = 1 \qquad k\in K \tag{20}$$

$$-\sum_{k\in K}\sum_{\tau\in\mathcal{T}_{ak}} q_k x_a^{k\tau} + u_a y_a \geq 0 \qquad a\in A \tag{21}$$

$$x_a^{k\tau}\in[0,1] \qquad k\in K, a\in A, \tau\in\mathcal{T}_{ak} \tag{22}$$

$$y_a\in\{0,1\} \qquad a\in A, \tag{23}$$

where $\mathcal{T} = \cup_{a\in A,k\in K}\mathcal{T}_{ak}$. Note that, to reduce the symmetry of he model, each commodity is forced to leave its origin at time 0. The number of variables and constraints is $\mathcal{O}(|A||K||\mathcal{T}|)$ and $\mathcal{O}(|N||K||\mathcal{T}|+|A|)$, respectively, compared to $\mathcal{O}(|A||K|)$ and $\mathcal{O}(|N||K|+|A|)$ in the arc formulation for the FCMCFP.

The remaining question is what time step to use. As the size of the model is proportional to the number of time steps, we try to limit the size of $\mathcal{T}_{ak}$ for each $a\in A$ and $k\in K$. If there is a large smallest common divisor for the time parameters in the instance, that would generally be a natural choice for the time step. The most notable instance of this is when all time parameters are integer, in which case one is a common divisor, and the integers make for a natural set of time indices. This approach is used by Trivella et al. (2020).

We can further reduce the size of the problem by removing unnecessary indices. For any commodity k, we can remove every time-arc combination, for which either we cannot reach the arc at that time, or we cannot reach the destination in time from that arc at that time. Finding the minimum and maximum time for each arc-commodity pair amounts to solving a one-to-all shortest path problem from the commodity's origin and from its destination, respectively.

For instances where the smallest common divisor between the time parameters is very small, the size of the model becomes prohibitively large, and it becomes necessary to investigate other means of defining the time indices. One possible approach is to round all time parameters to integer values. To guarantee that the solution remains feasible to the original problem we need to round the arc transfer times upwards and the transit time limits downwards. By first multiplying the time parameters with a number strictly larger than one, we can reduce the proportional impact of the rounding at the cost of an increased model size. This can be improved slightly, by rounding down arc transfer times whose remainder is proportionally smaller than that of the transit time limit. In other words, if rounding down the transit time limit for a commodity reduces it by $x\%$, any arc transfer time which would be reduced by less than $x\%$, if rounded down, for that commodity, can be rounded down instead of up. This is easily seen, as in the worst case scenario, where all arc transfer times would be reduced by exactly $x\%$, this would just result in a flat scaling which does not affect the feasible region.

3 Valid inequalities for the fixed charge multi-commodity flow problem

The FCMCFP and other fixed charge network design problems are known for having very weak linear programming relaxations (Gendron et al., 1999) and a number of valid inequalities have been proposed to strengthen the formulations. The main families of cuts are valid for three different relaxations of the problem: the single-arc design, single-cutset, and single-cutset flow problems. A thorough description of the various valid inequalities can be found in Chouman et al. (2016). While flow-cover and flow-pack inequalities, which are valid for the single-cutset flow problem relaxation, significantly strengthen the model, they are difficult to separate and they have a distinct negative impact on the run time in Chouman et al. (2016). Hence, we have decided to only use cuts based on the single-arc design relaxation and the single-cutset relaxation. In the following we will briefly introduce those. All cuts presented here are valid for all three presented models of the TTFCMCFP, introduced in Section 2. We will present the cuts denoting the arc flow by x_a^k, as in the arc formulation for the FCMCFP, but this can then be replaced by $\sum_{p\in P_k} b_a^{kp}\omega_{kp}$, $\sum_{s\in S_k} x_a^{ks}$, or $\sum_{\tau\in\mathcal{T}_{ak}} x_a^{k\tau}$, depending on the formulation used.

3.1 Strong inequalities

Our first set of valid inequalities are the strong inequalities, defined as:

$$x_a^k \leq y_a, \ \forall a \in A, k \in K. \tag{24}$$

The strong inequalities are probably the most common cutting planes for the FCMCFP, and are often written as part of the original model (Gendron et al., 1999). While there is only a polynomial number of strong inequalities, adding them all to the model generally makes it highly degenerate. Hence, the more common approach is to add them dynamically (Chouman et al., 2016), as they are easily separated by inspection.

The strong constraints are facet-defining for the single arc design relaxation:

$$\left\{ y_a \in \{0,1\},\ x_a^k \geq 0\ \forall k \in K \ \middle|\ \sum_{k\in K} q_k x_a^k \leq u_a y_a,\ q_k x_a^k \leq d_k \right\}, \tag{25}$$

and together with the remaining constraints, they define the convex hull of that relaxation (Chouman et al., 2016).

3.2 Lifted cover inequalities

Lifted cover inequalities are mixed integer programming cuts for knapsack polytopes. For fixed charge multi-commodity problems, the most commonly used knapsack constraints are the single-cutset inequalities. Single-cutset inequalities are common practice in most transportation problems, and are based on the observation that if we split the nodes into two sets, there has to be sufficient capacity available between the two sets to cover all demand which has it origin in one set and its destination in the other. Given a set $S \subset N$, and its complement $\bar{S} = N \setminus S$, let us denote the arcs going from S to $\bar{S}$ by $A(S, \bar{S})$ and let $q(S, \bar{S})$ be the total quantity of the commodities with origin in S and destination in S'.

Then, the single-cutset inequalities can be defined as

$$\sum_{a\in A(S,\bar{S})} u_a y_a \geq q(S, \bar{S}). \tag{26}$$

The single-cutset inequalities by themselves are not strengthening the model, but they are the base from which we can then generate *cover inequalities* (CI) and *minimum cardinality inequalities* (MCI).

The well-known cover inequalities were first introduced by Balas (1975), Hammer et al. (1975) and Wolsey (1975). Given a single-cutset inequality, a cover is a set of arcs $C \subseteq A(S, \bar{S})$ such that the arcs in $A(S, \bar{S}) \setminus C$ have insufficient capacity to satisfy the demand, i.e., $\sum_{a\in C\setminus A(S,\bar{S})} u_a < q(S, \bar{S})$. For any cover, at least one of its arcs has to be open in any feasible solution, resulting in the following inequality:

$$\sum_{a\in C} y_a \geq 1. \tag{27}$$

A cover is said to be minimal if removing any arc from it invalidates the above inequality. Each cover inequality can then potentially be lifted. This means that, given a cutset inequality and a cover C, we change the cover inequality to:

$$\sum_{a\in C} y_a + \sum_{\hat{a}\in A(S,\bar{S})\setminus C} \alpha_{\hat{a}}(1 - y_{\hat{a}}) \geq 1. \tag{28}$$

If $\alpha_{\hat{a}} > 0$ for any $\hat{a}$, this strengthens the constraint, as $y_{\hat{a}} \leq 1$. The classic way to lift cover inequalities is to lift the variables sequentially, i.e., in some order of $\hat{a} \in A(S, S') \setminus C$ choose the maximum $\alpha_{\hat{a}}$ such that constraint (28) remains valid. Clearly, if $y_{\hat{a}} = 1$, the constraint is satisfied for any value of $\alpha_{\hat{a}}$, thus we just need to care about the case where $y_{\hat{a}} = 0$. This gives us $\alpha_{\hat{a}} \leq \sum_{a \in C} y_a - 1$, which in turn means that the strongest valid inequality will be obtained with:

$$\alpha_{\hat{a}} = \min \sum_{a \in C} y_a - 1 : \sum_{a \in A(S,\bar{S}) \setminus \{\hat{a}\}} u_a y_a \geq d(S, \bar{S}).$$

We see that this is a knapsack problem and thus generating the strongest lifting for a single arc can be done in pseudo-polynomial time. Nonetheless, if this is to be performed often, there are alternative, less computationally demanding, ways of lifting the cover inequalities (Balas, 1975). The problem is then only to decide in which order we would want to try to lift the various variables.

Balas (1975) and Wolsey (1975) also prove that such lifted minimal cover inequalities define facets for the polytope generated by the single-cutset inequalities and integer restrictions on the y-variables.

It is also common to initially ignore some arcs which are close to being closed in the current solution. In that case the inequality first has to be downlifted to be valid (Chouman et al., 2016).

4 A branch-and-price implementation for solving the path formulation of the TTFCMCFP

Pushing the transit time constraints to the definition of feasible paths, in the path formulation, is one of the more straightforward ways to model an otherwise quite complicated constraint. Due to the exponential number of feasible paths, complete enumeration is generally infeasible and instead we develop a branch-and-price algorithm to solve the formulation. The subproblem is a resource constrained shortest path problem, which can be solved in pseudo-polynomial time using dynamic programming (Mehlhorn and Ziegelmann, 2000). In this section we will describe our branch-and-price algorithm along with some of the key implementational details.

One core choice when implementing a branch-and-price framework is whether to use a single global set of columns or to let each branching node have a set of local columns. When using separate sets of columns, some columns may be generated multiple times resulting in the need to solve the master problem more frequently. On the other hand, each master problem is likely to contain fewer variables, as we only use the columns generated for a particular branch. Separate column pools also make the framework more flexible, and we can add cuts tailored for each node as well as remove superfluous paths from individual nodes. From an implementation standpoint, if we use local column pools, we generally have to rebuild the model at each node, which is costly, whereas using the global columns pool we can work with a single model, just updating the branching constraints. In Section 5 we compare the two approaches.

Due to branching decisions the model frequently becomes infeasible, in which case we use the Farkas' certificate to generate new paths to try to break the infeasibility. When this is not possible, we prune the branch. When starting with an empty column pool, it is generally preferable to use artificial rejection variables for the commodities, until a feasible solution has been reached, as this tends to generate a more useful set of initial columns. Once a feasible solution has been found, the artificial variables are removed.

As mentioned earlier, the subproblems are resource-constrained shortest path problems, which we solve with a classic labeling algorithm (Mehlhorn and Ziegelmann, 2000). As we will see in Section 5.4.3, the time spent on solving the subproblem is relatively minor in comparison to the time it takes to solve the master problem. This is in contrast to most vehicle routing problems, where solving the subproblem is the more time consuming element.

4.1 Branching

In the TTFCMCFP, the only integer variables are the design variables. Hence, the branching decisions are limited to which arcs should be chosen to branch on, whether an aggregation of arcs should be used, and in which order to process the branching tree nodes. Some implementations branch on the inflow/outflow of nodes (Santini et al., 2018), but initial results showed that branching immediately on the design variables gives superior results for our problem. Regarding which arc to branch on, we tried two principles: the first one chooses only the most fractional arc in the current solution and the second further weights the distance between the current design variable value and the closest integer with the capacity of the arc. More complex branching rules, such as reliability branching, have also shown good results in previous work (Gendron and Larose, 2014).

For the branching node order, we choose nodes in order of increasing objective function value, to try to find good solutions early. This works well for instances for which the whole branching tree can be stored in memory, but when solving the largest instances, with split columns pools, it becomes too memory intensive. To handle this, we switch to depth-first search when the branching tree contains more than 250,000 open nodes.

4.2 Initial integer solutions

It is common practice to generate an integer solution from the columns generated in the root node. This can serve both as a good upper bound for the branching process, or in some cases as a good final heuristic solution to the problem. Depending on the structure of the problem, the quality of integer root node solutions varies significantly, but we will see that it may be a good alternative for some instances. Clearly, one of the benefits of using this is that it allows exploiting the MIP solver's full potential, with all the cuts and heuristics that come with it.

4.3 Deep dual-optimal inequalities

Deep dual-optimal inequalities denote dual inequalities that cut off feasible dual solutions and potentially some but not all optimal dual solutions (Ben Amor et al., 2006). Their primary goal is to reduce the dual solution space, thus reduce dual oscillations, and ultimately stabilize column generation algorithms. Koza et al. (2020b) recently proposed a set of deep dual-optimal inequalities for generalized capacitated fixed-charge network design problems that we apply in our branch-and-price framework. To understand how they act on the dual space, let us first present the dual restricted master problem.

Let $\tilde{P}_k$ be the set of currently generated paths for commodity k, in the restricted master problem. Let α_k be the duals for the set-covering constraints (3), and let β_a be the dual variables, corresponding to the capacity constraints (4). Further, let γ_a^k be the dual variables corresponding to the strong linking inequalities (5). In each node we have a set of lower and upper bounds for the design variables, either due to branching decisions or for defining the variable domains,

$$y_a \geq e_a \qquad a \in A \tag{29}$$

$$y_a \leq l_a \qquad a \in A, \tag{30}$$

where $e_a \in \{0, 1\}$ and $l_a \in \{0, 1\}$. Let μ_a be the dual variables for the lower bounds and η_a for the upper bounds. We also have a set of cover inequalities (28), which we index by $f \in F$. For ease of presentation, let us write them in the following form:

$$\sum_{a \in A} v_a^f y_a \geq h_f \qquad f \in F,$$

where v_a^f is the coefficient for arc a in cover inequality f, and lastly we let ξ_f be the dual variables for the cover inequalities. The dual of the restricted master problem can then be written as:

$$\max \sum_{k \in K} \alpha_k + \sum_{a \in A} (e_a \mu_a - l_a \eta_a) + \sum_{f \in F} h_f \xi_f \tag{31}$$

$$\text{s.t.} \quad \alpha_k - \sum_{a \in A} b_a^{kp} a(q_k \beta_a + \gamma_a^k) \leq c_{kp} \qquad k \in K, p \in \tilde{P}_k \qquad [w_{kp}] \tag{32}$$

$$u_a \beta_a + \sum_{k \in K} \gamma_a^k + \mu_a - \eta_a + \sum_{f \in F} v_a^f \xi_f \leq d_a \qquad a \in A \qquad [y_a] \tag{33}$$

$$\alpha, \beta, \gamma, \mu, \eta, \xi \in \mathbb{R}^+. \tag{34}$$

Here, constraints (32) and (33) are the dual constraints for the variables w_{kp} and y_a, respectively. The deep dual-optimal inequalities proposed by Koza et al. (2020b) correspond to a primal relaxation of the lower bound on the y-variables that is equivalent to replacing dual constraints (33) by the tighter constraint set

$$u_a\beta_a + \sum_{k\in K}\gamma_a^k + \mu_a - \eta_a + \sum_{f\in F} v_a^f \xi_f = d_a \qquad a \in A. \tag{35}$$

The primal relaxation is feasible because arc capacity constraints implicitly define a lower bound on the y-variables. In the dual space, replacing (33) by equality constraints (35) leads to a reduced dual solution space, which implicitly stabilizes the column generation algorithm.

4.4 Variable fixing

Variable fixing is a technique that exploits reduced cost information to fix integer variables, in the sub-tree of the node being processed (Gendron and Larose, 2014, Chouman et al., 2018). In our implementation, if $y_a \in \{0,1\}$ and $Z_{LB} + |d_a - u_a\beta_a - \sum_{k\in K}\gamma_a^k - \mu_a - \sum_{f\in F} v_a^f\xi_f| \geq Z^*$, where Z_{LB} and Z^* denote the lower and upper bounds on the objective value, in a node, we can fix y_a to its current value.

4.5 Generation of strong linking inequalities

We test two different methods for adding strong linking inequalities. In the first, we add them once a path is generated for the corresponding arc-commodity pair (*static* generation). In the second, we instead separate the strong linking inequalities that are violated each time the master problem is solved (*dynamic* generation).

In the case of dynamic generation, each time new constraints are added, the master problem has to be solved again. This new solution might again contain violated strong inequalities. But, as a trade-off between getting the tightest possible lower bound at each iteration, and minimizing the number of solver calls to solve the master problem, each time we only check for violated strong constraints for one iteration. The one exception is at the root node, where all strong inequalities are generated to keep the root node gap consistent.

4.6 Generation of lifted cover inequalities

Also for the lifted cover inequalities, we test two generation approaches, which we will denote *static* and *dynamic* generation. In both cases, we generate the cover inequalities from single-cutset inequalities, but they differ in *when* we generate them and *what* single-cutset inequalities we try to find violated inequalities from.

For a given cutset, in both approaches, the most violated cover inequality is found by solving the corresponding knapsack problem. The inequality is then lifted sequentially by solving a knapsack problem for each arc not in the original cover. As lifting variables with low values in the current solution is more likely to lead to a violated constraint, we try to lift the variables in order of increasing y-value. It is also possible to find violated inequalities from lifting non-violated inequalities, and while this increases the number of inequalities to lift, initial results showed that this indeed yields a minor improvement in overall run times. We use the algorithm from Pisinger (2000) to solve the knapsack problems.

In the static generation we generate the lifted cover inequalities at an initial stage, before any paths are generated, inspired by Barahona (1996). We start with a relaxation of the FCMCFP (and of the TTFCMCFP) including only the design variables, with their respective costs and with the integrality restrictions relaxed. We start without any constraints and solve the problem and then look for violated single-cutset inequalities. For each violated single-cutset inequality found, we then generate a corresponding lifted cover inequality, which we add to the model. Then we re-solve and iterate until no more violated single-cutset inequalities are found. This is done in two steps: First we look only at single-node single-cutset inequalities, and then we iteratively define and solve a set of max-cut problems to find the most violated single-cutset inequalities (Barahona, 1996).

In the dynamic generation, we instead separate inequalities dynamically throughout the branching tree. We assume that we have a feasible solution to the relaxed problem, and hence there are no violated single-cutset inequalities and thus the max-cut approach makes less sense. Instead, we revert to single-node single-cutsets inequalities, which are shown to be the best performing for the normal FCMCFP by Chouman et al. (2016).

5 Computational experiments

This section will be divided into four parts. The first part will introduce and discuss the instances used. In the second part the instances and their solutions are studied to see the impact of the instance characteristics on the solutions and the difficulty to solve them. In the third part the compact models are compared with the branch-and-price implementation. The last part is more of a dive into the details of the branch-and-price algorithm, to study how its various components impact its performance. All tests are run on a single core, using an Intel Xeon 2660v3 2.60GHz processor with 64GB of RAM. For the remainder of this section, whenever we mention the logarithm we mean the logarithm in base 10.

5.1 Instances

To test our algorithms we use the Canad instances, presented in Crainic et al. (2001), modified to include transit times. The transit times were generated as follows: for each instance, a feasible solution was found, by taking the first integer solution in the CPLEX branch-and-bound tree. For each arc a transit time was then added, proportional to the arc's fixed charge cost. The maximum transit time limit for each commodity k was then calculated as the maximum time used by any of that commodity's paths, scaled by a coefficient δ. This way, each new instance is guaranteed to have a feasible solution. Instead of defining a set of instances, each with a predefined transit time coefficient, we leave the transit time coefficient open. In other words, each instance will be written down with transit time coefficient one and then any instance can be obtained by just multiplying each maximum transit time by the sought coefficient. The new instances will keep the original name followed by the transit time coefficient in parenthesis. For example, we will denote the transit time constrained Canad instance *r01.1* with the transit time coefficient δ by *r01.1*(δ). Note that using a time coefficient less than one usually renders the instance infeasible as, in the feasible solution which the instance is built around, there tends to be at least one commodity that took the shortest path from its origin to its destination.

We note that the instance classes *r01.x*, *r02.x*, and *r03.x* contain infeasible instances, and *r06.x* does not follow the same pattern of increasing arc capacities and fixed charge costs as the others. Hence, those instance classes are not used in this work. Additionally, to be able to compare the different properties of the r-instances, we used the same transit times for each base-instance, and they were built from *rx.7*. The full instance set can be found at `https://zenodo.org/record/4050442` (Hellsten, 2020).

5.2 Instances analysis

In studying the instances we are mainly interested in the impact of three different factors: the transit time limits, the relation between fixed costs and flow costs, and the arc capacity. We are interested in how they affect the solutions and the difficulty of solving the problem, which we try to estimate by looking at the run times, number of columns generated, the number of nodes in the branch-and-bound tree, and the integrality gap. All experiments for this section were run with a time limit of two hours, using the branch-and-price framework with the base features, as will be presented in Section 5.4.1.1. In order to analyze how the three factors impact the solutions, we study the change in objective value, but also in the number of paths used by each commodity.

We begin with studying the impact of the transit times. All transit time limited Canad instances were run with the transit time coefficients $\delta = 1.1$, 1.2, 1.3, 1.6, and ∞. A summary of the results is presented in Table 1. We look at the run times, the numbers of nodes in the branching tree, the numbers of generated columns, the objective values, the integrality gaps and the average numbers of paths used by the commodities. The results are aggregated over each Canad instance that could be solved within two hours for each of the five transit time coefficients. First we look at the mean over each of those instances. However, as the mean poorly represents data that vary significantly in scale, we also look at the mean of the logarithm of the data and the mean of the data normalized by the lowest value found for each instance, among any of the five transit time coefficients used.

We see that the run times increase significantly with higher transit time coefficients and so do the numbers of columns generated and numbers of nodes in the branch-and-bound trees. Looking at the normalized mean, we see that the lowest transit times almost always result in the lowest numbers of columns but far from always result in the lowest numbers of branch-and-bound nodes and run times. While the mean and log-mean clearly point towards that more slack transit times generally increase the number of nodes in the branching tree, looking at the normalized mean, we see that it is not completely clear-cut. We note that there is a big gap between $\delta = 1.6$ and using no transit times at all, not only in run times but also in objective values, meaning that we are dealing with relatively tight transit time constraints. From the very high values of the normalized mean, with only moderate values of the log-mean, we draw the conclusion that there are a few instances where the run times and numbers of branching nodes increase dramatically when removing the transit time constraints. The largest difference seen is for *r14.2*, which was solved in about 0.3 seconds with tight time constraints and a bit over 6000 seconds without. On the other hand, the numbers of columns generated is much more stable and seems to be more monotonically increasing with the transit time constraints. This is what would be expected, as loosening the transit time constraints increases the number of feasible columns.

The objective values increase by about 6% on average when adding tight transit time coefficients, and looking at the difference in mean objective value, it instead increases by about 7%. We also see that the average number of paths used for each commodity is about 1.40, which is rather low and seems to be more or less unaffected by the transit time constraints. Note that this, and the other numbers, are deflated by the large number of small instances. In terms of instances solved, 113, 113, 112, 110 and 97 out of the 126 instances where solved within the time limit, with $\delta = 1.1$, 1.2, 1.3, 1.6 and ∞, respectively.

Next we study the impact of arc capacity and the fixed charge cost. For the *r-instances*, each base instance class is divided into three tiers of arc capacity and three tiers of fixed charge costs, resulting in nine instances for each base instance class. There are a total of 18 base instance classes, but four of them are removed as explained earlier, which means that there are a total of 14 instances for each arc capacity tier, and for each fixed charge tier. We solve all instances for the four levels of transit time constraints presented above, $\delta \in \{1.1, 1.2, 1.3, 1.6\}$. This gives us a total of 56 instances for each class, and 168 instances for each arc capacity tier and fixed charge cost tier, respectively. For the *high* arc capacity tiers, all 56 instances are solved within the two-hour limit, whereas for the *medium* capacity tier, two instances were not solved. The remaining 54 unsolved instances all

Table 1: *Sensitivity results for different transit time coefficients*

Measure	δ	Run Times	BB Nodes	Columns	Objective Value	Integrality Gap	Avg. No of Paths
Mean	1.1	46.74	2831.67	769.05	1271746.24	2.61	1.40
	1.2	73.64	2821.60	843.46	1259983.16	2.63	1.40
	1.3	65.33	3110.81	885.82	1251790.44	2.62	1.40
	1.6	87.57	4000.98	1067.73	1233360.51	2.79	1.40
	∞	491.72	8236.19	2168.02	1188533.48	3.22	1.40
Mean (Log)	1.1	0.68	2.42	2.57	5.62	0.43	0.37
	1.2	0.72	2.43	2.59	5.61	0.43	0.37
	1.3	0.74	2.44	2.61	5.61	0.43	0.37
	1.6	0.83	2.53	2.67	5.60	0.45	0.37
	∞	1.13	2.76	2.84	5.59	0.51	0.37
Normalised	1.1	2.16	5.12	1.04	1.06	1.17	1.02
Mean	1.2	2.62	7.63	1.10	1.05	1.29	1.02
	1.3	3.28	6.20	1.16	1.04	1.38	1.02
	1.6	3.73	3.85	1.38	1.02	1.53	1.03
	∞	388.87	147.98	2.71	1.00	4.31	1.02

belong to the *low* capacity tier. Out of those 54 instances, 14 belong to the *low* fixed charge cost tier, 20 to the *medium* fixed charge cost tier and 20 to the *high* fixed charge cost tier.

More detailed results are shown in Tables 2 and 3. The results shown are for the complete instance-tier combinations, i.e., the results for varying the fixed charge cost are for each combination of base-instance class and capacity tier which was solved for each of the fixed charge tiers, and the opposite for the capacity results. Again we report the mean, the mean of the logarithm and the mean of the normalized values. Here, the normalization is by the lowest value among the three tiers of fixed charge cost in Table 2 and among the three tiers of arc capacity in Table 3.

In Table 2 we see that the difficulty of the instances increase with higher fixed cost ratios. Increasing the fixed charge part of the cost significantly increases the integrality gap, as it shifts the cost to the integer variables, which generally increases the number of branching nodes and columns needed. Looking at the normalized mean, we see that the version with the lowest ratio between fixed charge and flow cost almost always has the lowest run time and always has the lowest number of columns. The number of branching nodes vary more, but there is still a definite trend. We also see that the average number of paths used decreases with increased fixed charge cost. Increasing the cost of opening arcs naturally aggregates the flow onto fewer arcs and hence also fewer paths.

For the arc capacity, the results are less clear-cut. Lowering the arc capacity naturally forces the flow to be split among more arcs and paths, which we also see in Table 3. Using more arcs in the solution also means that there would be more fractional arcs and hence more branching is needed to find integer solutions. We also see that the number of branching nodes is significantly higher for the low capacity instances. The impact on the number of columns generated is less clear. We see that the mean is actually higher for the high capacity instances, but both the log-mean and the normalized mean are quite significantly higher for the low capacity instances. This points towards lower arc capacities increasing the number of columns, but a few instances with lots of columns go

Table 2: *Sensitivity results for different fixed charge costs*

Measure	**Fixed Charge Cost**	Run Times	BB Nodes	Columns	Objective Value	Integrality Gap	Avg. No of Paths
Mean	Low	7.60	876.52	438.73	737398.03	1.09	1.40
	Medium	135.46	3815.56	1192.11	1331900.37	2.66	1.35
	High	213.74	4904.37	1828.22	1961121.93	3.63	1.34
Mean (Log)	Low	0.42	2.07	2.36	5.47	0.26	0.37
	Medium	0.95	2.58	2.78	5.72	0.45	0.36
	High	1.16	2.78	2.97	5.88	0.55	0.36
Normalised	Low	1.12	1.71	1.00	1.00	1.01	1.05
	Medium	33.54	11.17	3.19	1.78	4.11	1.03
	High	60.52	16.30	5.62	2.59	5.19	1.02

Table 3: *Sensitivity results for different arc capacities*

Measure	**Arc Capacity**	Run Times	BB Nodes	Columns	Objective Value	Integrality Gap	Avg. No of Paths
Mean	Low	144.32	25922.48	788.30	2318308.42	3.85	2.03
	Medium	126.94	4908.09	1060.10	1070336.62	3.65	1.27
	High	179.70	2985.32	1107.44	846470.26	1.30	1.08
Mean (Log)	Low	0.82	2.98	2.73	5.85	0.56	0.48
	Medium	0.97	2.93	2.67	5.56	0.58	0.36
	High	0.76	1.95	2.41	5.49	0.28	0.32
Normalised	Low	2436.18	7406.81	4.25	10.56	10.34	1.90
	Medium	83.51	153.38	2.86	1.18	11.88	1.18
	High	73.71	19.38	1.47	1.00	4.37	1.00

against this trend. It is interesting to see that the gaps increase for low capacity instances. Looking at capacity scaling heuristics (Katayama et al., 2009), for example, it is a commonly used idea to lower the arc capacities to improve the bound. But if they are reduced too much, we instead get the effect that the flow needs to use more arcs.

5.3 The compact models

We now compare the same basic version of the branch-and-price algorithm with the two presented compact models: the explicit-path model presented in Section 2.3, and the time-indexed model presented in Section 2.4. As the compact models are slower to solve, we restrict ourselves to look at the small *r-instances* with only 10 nodes, and we use the transit time parameter $\delta = 1.2$. This constitutes a total of 45 instances. Additionally, as both models are alterations of the true problem – the explicit-path formulation uses an upper bound on the number of paths per commodity, and the time-indexed formulation requires integer time indexes – we study how those alterations impact the solutions. For the compact models we tried adding strong linking inequalities upfront, as well as adding them dynamically as user-cuts, both locally and globally. The results showed that adding them upfront was significantly faster than any of the alternatives. Hence, the results shown in this section will be for the compact models with all strong linking inequalities. The compact models

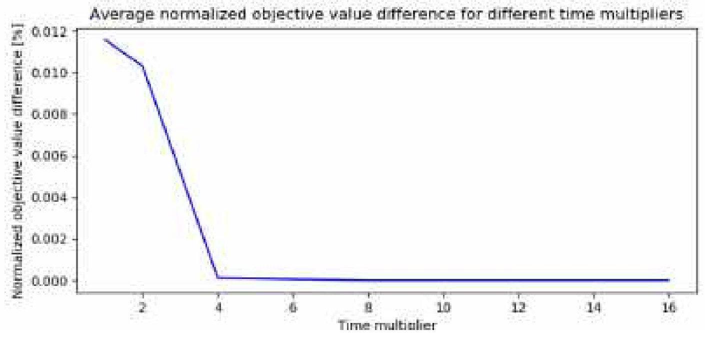

Figure 1: *Average objective value difference from rounding time parameters as a function of the time multipliers for the time-indexed formulation*

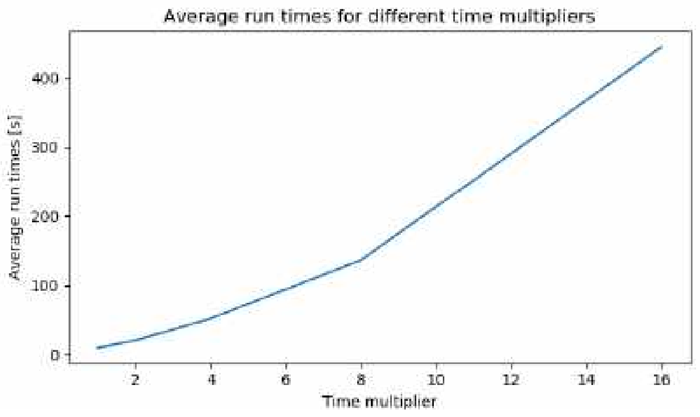

Figure 2: *Average run time for increasing time multipliers for the time-indexed formulation*

were run with a time limit of one hour.

For the time-indexed formulation, we use integer time steps and do the rounding as proposed in Section 2.4. In Figure 1 we see the mean of the absolute difference in objective value from rounding the time parameters for different time multipliers. We see that the error from rounding is marginal for those small instances, and that it disappears slowly when increasing the time multiplier. On the other hand, in Figure 2 we see the average run times for increasing time multipliers. Not only does this show the effect of using time multipliers, but it gives an idea about how this model would scale towards instances with large time parameter values.

For the explicit-path model, we ran the model for the 45 instances using a maximum of 1 to 4 paths for each commodity. In Table 4 we see the number of feasible instances, the number of instances which were solved to optimality within the time limit of one hour, and, out of those, how many have the same objective value as the problem without limiting the maximum number of

Table 4: *Statistics for the solutions to the explicit-path model with different number of maximum paths*

Maximum Number of Paths	Feasible Instances	Solved Instances	Correct Obj Instances
1	36	36	16
2	45	36	33
3	45	38	38
4	45	36	36

paths. We see that using a single path seems insufficient, even for those small instances, whereas already using up to three paths seems completely sufficient. But as we would expect, the number of instances we can solve decreases when we increase the maximum number of paths.

In Table 5, we see how the two compact models compare to the branch-and-price implementation. The results are presented for different values for the time multiplier for the time-indexed formulation and for the maximum number of paths for the explicit-path formulation as indicated in parenthesis. We see that in terms of run time, the branch-and-price algorithm significantly outperforms the other two methods, and the time-indexed formulation is significantly faster than the explicit-path formulation. It should be noted, however, that the time-index sets are rather small for those instances. In terms of the number of nodes, the time-indexed formulation performs the best. This is partly due to the additional cuts and heuristics implemented in CPLEX, that our branch-and-price framework does not use. On the other hand, we see that a tremendous number of nodes are used to solve the explicit-path formulation. Regarding the impact of the parameters, we see that increasing the time multiplier has a dramatic effect on the run time for solving the time-indexed formulation, but as the time multiplier only affects the number of continuous variables, it has very little effect on the number of branching nodes. For the maximum number of paths, we see a relatively small increase in mean run time, but a large increase in the normalized mean. Part of this is explained by the upper run time limit. In general, the mean increases when we spend a lot of time on difficult instances, whereas the normalized mean is affected equally by all instances. So when we cap the run time for the larger instances, we would see this behavior.

In the next section we will study the branch-and-price algorithm more in depth.

5.4 The branch-and-price algorithm

Here we want to study the performance of the branch-and-price algorithm for future benchmarks, and more importantly, we want to study the impact of its various parts. This section is organized in two parts: first we study the algorithm in detail, and try to find an optimal parameter setting, then we run the final experiments. The instance set varies significantly in difficulty, from trivial to extremely difficult, and neither edge case is particularly interesting to study. We will hence focus on a smaller subset of instances. Lastly, for all experiments in this section we will use a maximum

Table 5: *Statistics for the solutions to the compact models as well for 'r-instances' with up to 10 nodes and* $\delta = 1.2$

	Run Time			BB Nodes		
Method	Mean	Log (Mean)	Normalized Mean	Mean	Log (Mean)	Normalized Mean
Time-Indexed *(1)*	9.94	0.68	14.61	112.80	1.26	1.27
Time-Indexed *(2)*	20.46	0.91	30.25	134.55	1.30	1.40
Time-Indexed *(4)*	52.00	1.16	64.90	129.41	1.33	1.66
Time-Indexed *(8)*	136.54	1.46	139.79	126.43	1.35	1.95
Explicit-Path *(2)*	547.51	1.05	1227.37	43606.68	2.72	4490.24
Explicit-Path *(3)*	634.53	1.15	4776.18	51940.84	2.83	23169.26
Explicit-Path *(4)*	732.70	1.27	5039.39	55953.32	2.89	23358.66
Branch-and-price	2.31	0.28	1.42	1324.45	2.10	40.52

run time of two hours, if not otherwise written.

5.4.1 Algorithm analysis and parameter tuning

There are numerous design choices in developing an algorithm, and in the following we will study the impact of the ones we consider the most relevant. Features range from being almost implicitly assumed to being considered speed-up techniques, depending both on how close they are to the core of the algorithm and how common they are in previous work. We split the features into two groups. The *base features* make up the basic settings for the model. They are essential to the model and will be used in most experiments. The *core features* are the features we are mostly interested in testing and they will be tested individually as well as together to see how they interact with each other. The chosen base features consist of creating a new model in each node, storing the LP-basis between nodes, branching on the most fractional arc weighted by the arc capacity, and using dynamic strong linking inequalities. Together they make up the base setting. The decision to implement the framework with separate models is not due to performance, but it allows more interesting experiments for the remainder of the features. The impact of this will be studied later on. The core features are to use a price-and-branch upper bound, to use deep dual-optimal inequalities, and to use static or dynamic cover inequalities, as described in Section 4.6. We also tried the traditional LP-based variable fixing, but it turned out to have a significant negative impact on the run times.

5.4.1.1 Base features

The base features include what is common practice in most branch-and-price implementations, and thus we will not study them in detail. We study their impact by starting with all of them and then see the impact of removing them individually. This is done over a smaller set of medium size instances. Removing the dynamic strong constraints increased the average run time by 68% and the run time was increased for 39 out of 45 instances. Not transferring the basis status between nodes increased the run time on average by 45% and only two instances were solved faster when not using this feature. Using the basic arc-selection criteria, always choosing the most fractional arc, and not weighting it by capacity, increased the run time on average by around 1700%, but this is mostly due to a massive difference for a few instances. Overall, 30 instances were solved faster when including the arc capacity when choosing which arc to branch on, and 15 were solved faster otherwise.

5.4.1.2 Core features

For the main four core features, we first study their individual impact, by adding them to the base features. Those experiments where run over a slightly larger set of instances. In Figures 3 and 4, we see how the run times, numbers of nodes in the branch-and-bound tree, and the numbers of columns generated are affected when adding the core features. Each graph plots the quotient between the result when using a feature and the base case for each instance. On the x-axis the run time in logarithmic scale is shown. Naturally, results further to the right in the graphs carry more weight. We see that each of the core features, except for the price-and-branch upper bound, have a significant positive impact on the run times. We see that the price-and-branch upper bound significantly reduces the number of nodes in the branching trees, but has absolutely no impact on the number of columns generated. This is an interesting phenomenon. The reason why it happens is likely that, as we do best-first branching and the linear relaxation is rather weak, all necessary columns are generated before any pruning takes place, even with the improved upper bound. On the other hand, deep dual-optimal inequalities have in principle no impact on the number of nodes in the branching tree but a large impact on the number of columns generated. As the deep dual-optimal inequalities increase the cost of new columns, fewer columns are generally needed to solve each master problem. The slight variations in the number of branching nodes are due to that the deep dual-optimal inequalities can affect which primal optimal solution is found in a node, which in turn affects the branching decisions.

The cover inequalities have been shown to be effective for solving the FCMCFP before, and the results clearly show that it is also the case for the TTFCMCFP. Both the numbers of columns and the numbers of branching nodes are drastically reduced, leading to significant improvements in terms of run times. Comparing the static and dynamic generation, it is interesting to see that they show very similar patterns. It seems that there are a few cover inequalities, generated both in the static and the dynamic case, which dictate the numbers of columns and branching nodes, and hence also the run times. All in all, we also see a slight improvement going from static to dynamic generation.

To further understand the behavior of the cover inequalities, we look at the number of cover inequalities generated and lifted, the time spent on generation and the change in gap at the root

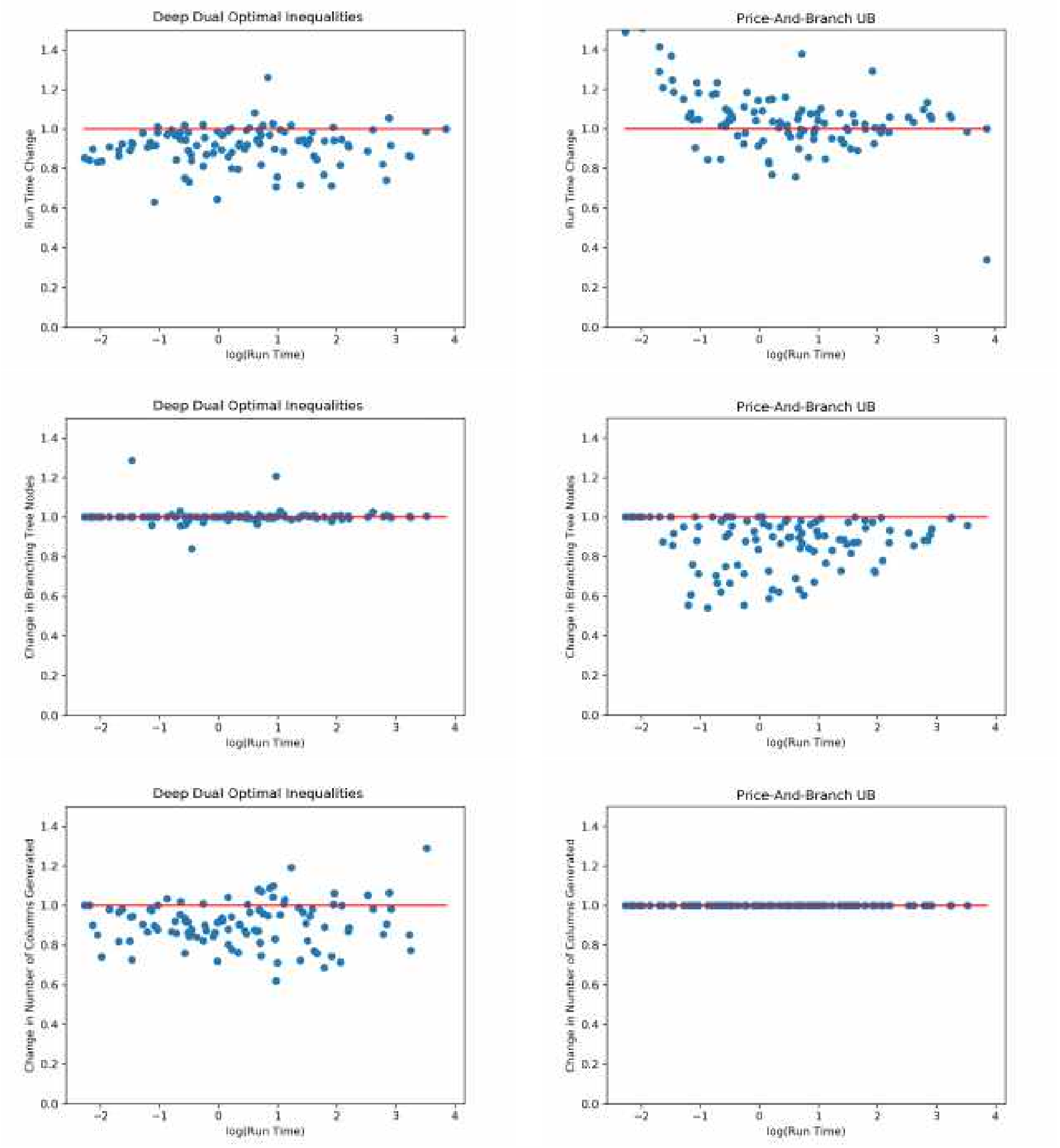

Figure 3: *Change in run time, number of branching nodes and number of columns generated, when adding deep dual-optimal inequalities or a price-and-branch upper bound to the base settings. Each dot represents an instance solved - on the y-axis is the quotient between the measure when and when not using the feature. On the x-axis is the logarithm of the run time using the base features. Any value below the solid red line means that the algorithm performed better using the feature. Note how the deep dual-optimal inequalities more or less only affects the number of columns and the upper bound only affects the number of nodes.*

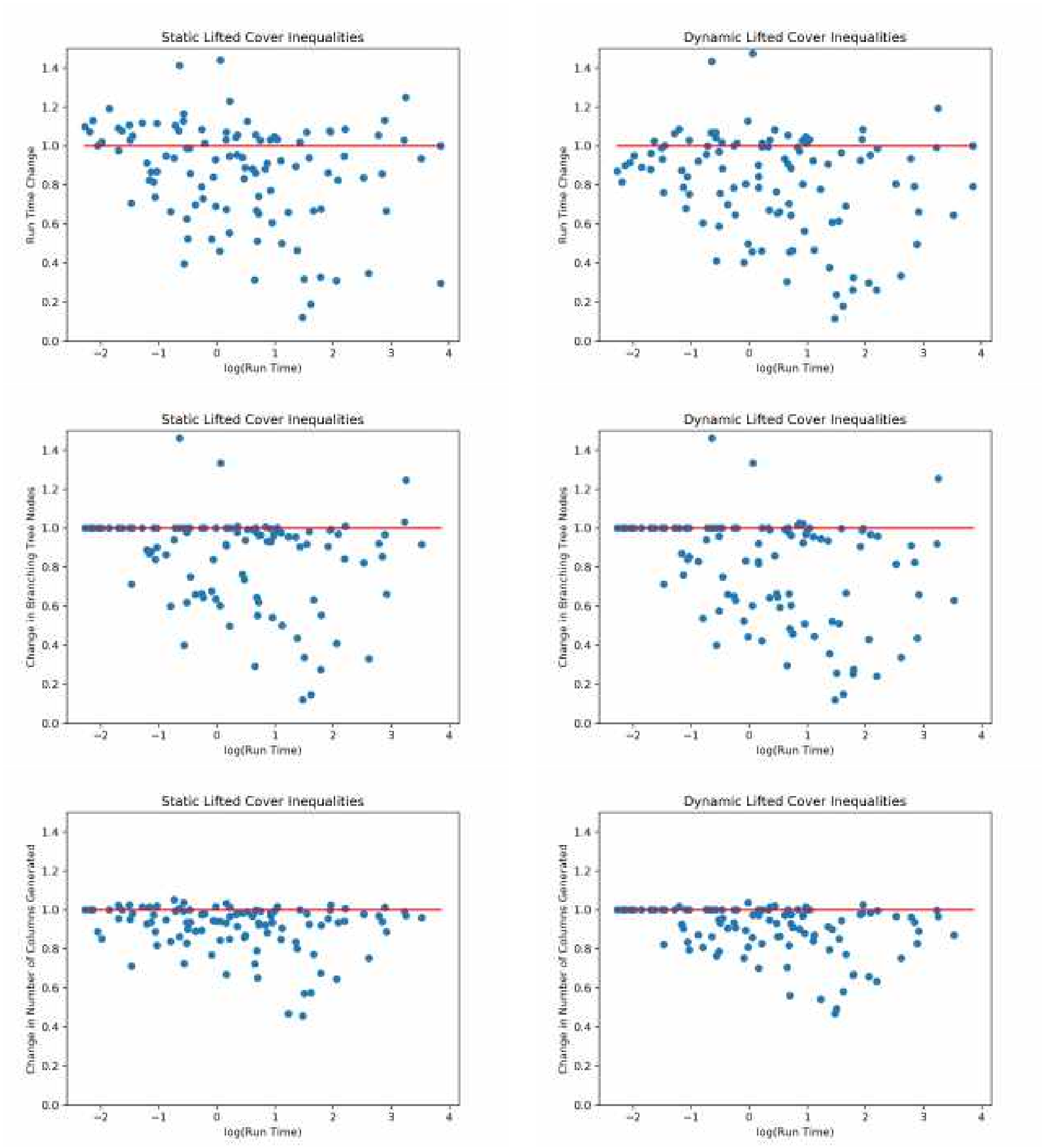

Figure 4: *Change in run time, number of branching nodes and number of columns generated, when adding statically and dynamically generated lifted cover inequalities to the base settings. Note the similar effects of the two in all three regards.*

node. In Figure 5 we see the number of cover inequalities generated using static and dynamic generation. Note that this is the total number of inequalities generated, across all branches, double counting inequalities generated multiple times in different branches. We see that the number of cover inequalities generated statically is increasing slowly with the difficulty of the instance, whereas the number generated dynamically increases dramatically. As the dynamic generation generates all violated cover inequalities at the root node, this also shows that the number of violated inequalities at the root node is significantly smaller than the total number of inequalities generated by the static method.

For the static generation, the generation time makes up on average 2.2% of the total run time. The reason that the average is this high is further that it makes up a large portion for a few instances, where the run times are less than 0.1 seconds. For the dynamic lifted cover inequalities, the generation time mostly ranges from 0 to 8%, with an average of 2.5%. The main reason why the generation times are this low, is that the knapsack algorithm by Pisinger (2000) is very fast, taking up but a fraction of the time used by the traditional dynamic programming approach. In terms of gap, the average integrality gap over all the tested instances were 2.506%, which was reduced to 2.093% using static cover inequalities and 2.010% using dynamic cover inequalities. The gap here is measured between the initial lower bound and the optimal objective value. Out of all dynamically generated cover inequalities, 13.8% were lifted.

Further, solving the price-and-branch integer master problem to generate an initial upper bound used up on average 9.3% of the run time. In the same manner as for the cover inequality generation, the integer master problem took up proportionally more time for the smaller problems and less for the larger. The average proportion reduces to 3.9% for instances which took longer than 10 seconds to solve. On average, the difference in objective value between the price-and-branch solution and the optimal solution is 0.51%. This is deflated by the number of small instances in the test set, but it is still significantly smaller than the average root node gap, for comparison. Disregarding the run time for solving the price-and-branch integer problem, using this improved upper bound reduces the overall run times by on average 9.5%. This suggests that there might be hope of achieving improved performance by finding good initial solutions using heuristics.

From those results it looks like dynamically generated lifted cover inequalities and deep dual-optimal inequalities yield the best results. Hence, lastly, we will look at how they work together. In Figure 6 we see the run time change when adding one to the other. In general, the results are good and they work well together. The effect the two have on the run time is similar to when added individually. It is interesting though that it has a large negative impact for the smallest instances. This is not only seen in the run time, but the smallest instances also have larger branching trees and more columns when using both features.

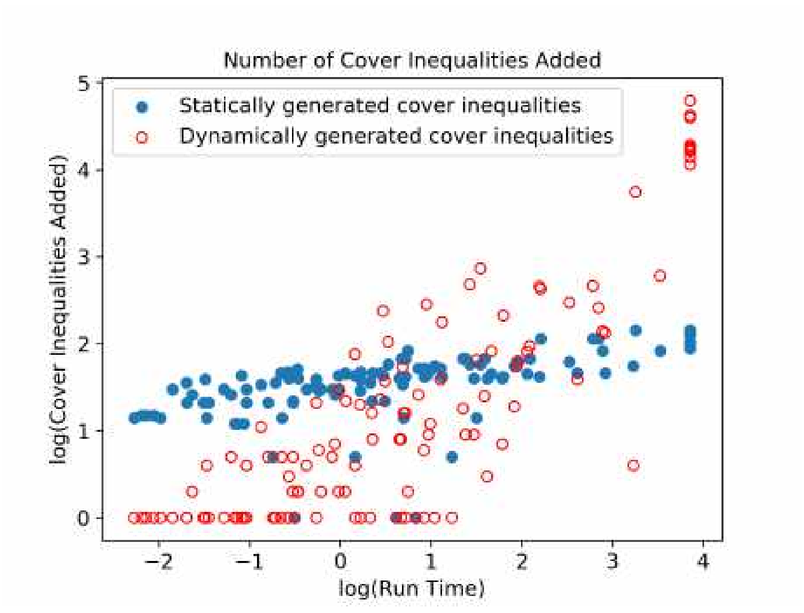

Figure 5: *The number of generated cover inequalities using static and dynamic generation against the run time for the solution. Note that both axes are in logarithmic scale. The number presented is the total number, double counting across different branching nodes.*

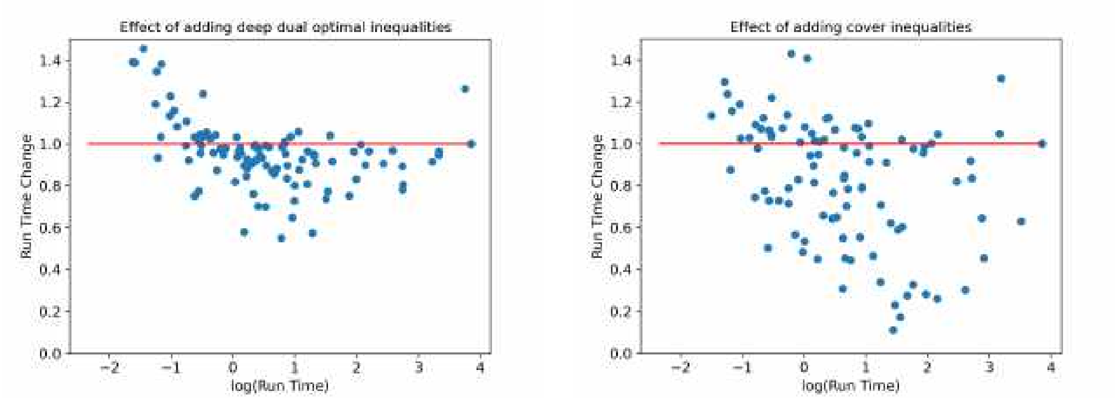

Figure 6: *Change in run time when adding dynamic lifted cover inequalities to a setting using the base features and deep dual-optimal inequalities, and when adding deep dual-optimal inequalities to a setting using the base features and cover inequalities. Note how it has a negative effect for the smallest instances.*

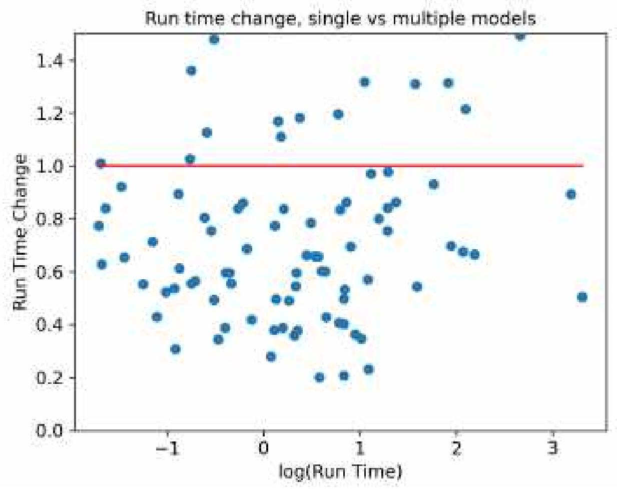

Figure 7: *Run time difference between the single model and multi model implementations, plotted against the logarithm of the run time. In general we see that the single model outperforms the multimodel, even when not using dynamically generated inequalities.*

5.4.2 Implementation using a single model

Up until now, all results shown have been using a framework which generates a separate model in each node. This has many advantages, such that as the possibility to have individual cuts and columns in each node, but it comes at the cost of also having to generate the problem at each node. An alternative is to use a single model, and then only update the branching rules at each node. This naturally works well with using a single global column pool, and global cuts. In our comparison between the single-model implementation and the previously used multi-model implementation, we use dynamic cover inequalities and strong linking inequalities for the multi-model version, and static cover inequalities and strong linking inequalities for the single model. We also use deep dual-optimal inequalities for both. The average number of branching nodes is 8% higher for the single model and the number of columns 1% higher. The run times are, however, significantly lower, as we can see in Figure 7. As we will see in the next section, part of this is due to that it is costly to rebuild the problem in each node. The reason that we mainly have used the multi-model version, is that it better allows the study of the cuts and speed-up methods, presented in this work.

5.4.3 Final results

For the final results, we will use the branch-and-price framework, with separate models for the nodes, with all base features, deep dual-optimal inequalities and dynamically generated lifted cover inequalities. We will look at some properties of the solution process, such as the division of time

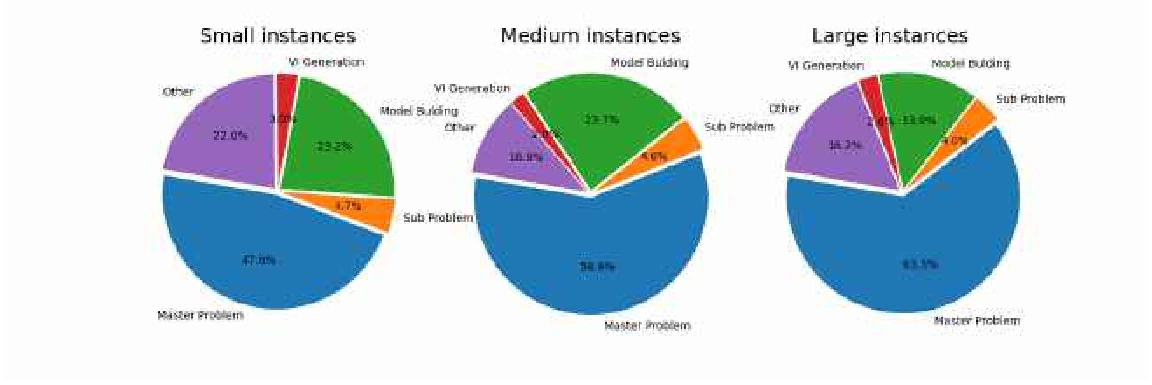

Figure 8: *Run time distribution for small, medium and large instances, classified as instances solved in less than one second, instances solved in more than one but less than a hundred seconds, and instances solved in more than a hundred seconds. Note how the master problem takes a proportionally larger time for larger instances.*

between the different parts of the algorithm, number of times the master problem is solved, etc. We will also present a table with results for future benchmarks.

If we look at the run time distribution between different elements of the algorithm in Figure 8, we see that the majority of the run time is spent on solving the master problem. Yet, significant time is used by the interaction with the solver. Both building the problem in each node, as well as retrieving values, such as dual variables, is rather slow. Naturally, not having to rebuild the master problem at each node is one of the main advantages with the single model approach. The *small* instances are instances solved in less than a second, the *medium* instances, those that were solved in between one and a hundred seconds, and the *large* instances are the rest.

Adding cuts dynamically requires us to re-solve the master problem to get the correct dual variables. As we just saw, solving the master problem is the most time consuming part of the algorithm, which means that this could be costly. To study this effect, we look at how many additional times we solve the master problem due to adding cuts. On average the master problem is solved 104,812 times, out of which on average 19,308, or 22 %, are additional due to cuts. If we instead look at the average fraction of master problems which are from adding cuts over the instances, we get 26%. This is a rather large portion, but we have seen, in the previous results, that the gain clearly outweighs this cost, in terms of reduced number of nodes and columns.

Lastly, in Table 6, we see a sample of the results as a benchmark for future projects. Here we increase the run time to 10 hours. The full set of benchmark results are attached in the appendix. There, we also append the results to the *C-instances.* Comparing our results to exact methods for the FCMCFP, we see that we solve significantly more of the transit time constrained Canad instances, than what state-of-the-art solution frameworks are able to solve of the corresponding

Canad instances, without transit time constraints (Chouman et al., 2018, Gendron and Larose, 2014, Koza et al., 2020b).

6 Conclusions

Transit time constraints for the commodities in the fixed charge multi-commodity flow problem, or other network design problems, appear commonly in practice. There is an important distinction between *transit time constraints* and *average transit time constraints*. If a problem can equally well be modeled using average transit time constraints, the formulation can be solved using the vast machinery developed for the traditional fixed charge multi-commodity flow problem, with only minor changes. In this work we have instead studied the transit time constraints applied to individual paths. We have shown that those additional constraints significantly change the structure of the problem, making it difficult to model using compact formulations. With some additional assumptions, we presented two compact models, based on splitting the flow variables over a predefined number of paths or over a set of time indices, but ultimately showed that the problem is more effectively handled if modeled using a path formulation and solved with a branch-and-price algorithm.

We generated a set of instances for the transit time constrained multi-commodity flow problem, which we used to show that many of the classic cuts and speed up techniques, for the normal fixed charge multi-commodity flow problem, also yield good results for the transit time constrained version. We also show that the new deep dual-optimal inequalities from Koza et al. (2020b) greatly reduce the number of columns generated, which significantly reduces the run times.

To further understand the problem, we study how different instance properties affect the performance of the solution algorithm. We observe that higher ratios between fixed charge cost and flow cost tend to increase the initial gap and the size of the branching tree, increasing the run times. The arc capacity has a less straightforward impact, but in general lower arc capacities force the flow to be split over more arcs, which again increases the initial gap, the size of the branching tree and the run times. For the path formulation, the transit time constraints in general make the problem easier to solve, as they reduce the number of feasible routes. Adding the transit time constraints makes the subproblem no longer solvable in polynomial time, but the results show that by simply using a traditional label-setting algorithm, the run time for the subproblem remains close to negligible.

A few observations stand out from our computational results. Maybe most surprising is that the single model significantly outperform the multi-model implementation. Also that the classic LP-based variable fixing greatly increases the run times.

While branch-and-price based algorithms seem to be the most promising direction for solving the problem optimally, many real-life applications require solutions to problems of sizes which are

Table 6: *Results for running the multimodel implementation using the base settings with dynamically generated cover inequalities and deep dual-optimal inequalities. For instances which were not solved in the given two hours the remaining gap is shown.*

Instance	Run Time	Obj Value	Branching Nodes	Columns	Root Node Gap [%]	Remaining Gap [%]
r15.1(1.2)	11.99	1122929.0	1092	886	0.41	0.0
r15.2(1.2)	1535.35	2379790.0	17452	4537	1.45	0.0
r15.3(1.2)	57.63	3580850.0	542	2991	1.05	0.0
r15.4(1.2)	153.76	1233428.0	10330	1393	0.75	0.0
r15.5(1.2)	459.12	2701550.0	7280	3495	1.36	0.0
r15.6(1.2)	435.55	4208750.0	4840	5511	2.12	0.0
r15.7(1.2)	19.36	2300560.0	1438	1718	0.69	0.0
r15.8(1.2)	39.06	5982840.0	2416	2042	0.89	0.0
r15.9(1.2)	116.12	10333880.0	5804	2561	0.95	0.0
r16.1(1.2)	0.09	140822.0	2	114	0	0.0
r16.2(1.2)	0.48	246851.0	18	480	0.33	0.0
r16.3(1.2)	1.4	336252.0	20	769	0.32	0.0
r16.4(1.2)	0.3	142669.0	62	164	0.27	0.0
r16.5(1.2)	6.22	254426.0	256	826	1.53	0.0
r16.6(1.2)	12.93	351625.0	388	1448	2.28	0.0
r16.7(1.2)	36000	-	-	-	-	21.59
r16.8(1.2)	36000	-	-	-	-	77.1
r16.9(1.2)	36000	-	-	-	-	32.7
r17.1(1.2)	0.12	422439.0	2	220	0	0.0
r17.2(1.2)	1.18	813651.0	36	673	0.7	0.0
r17.3(1.2)	0.89	1198281.0	20	713	0.23	0.0
r17.4(1.2)	0.36	425627.0	44	268	0.19	0.0
r17.5(1.2)	7.98	839583.0	560	860	0.84	0.0
r17.6(1.2)	7.18	1236717.0	286	1071	0.94	0.0
r17.7(1.2)	2027.23	515404.0	274148	1057	1.8	0.0
r17.8(1.2)	16434.9	1143413.0	660126	4478	3.48	0.0
r17.9(1.2)	36000	-	-	-	-	10.85
r18.1(1.2)	1.11	1026239.0	112	585	0.14	0.0
r18.2(1.2)	1.8	2059575.0	44	1053	0.52	0.0
r18.3(1.2)	3.34	3131660.0	122	1157	0.58	0.0
r18.4(1.2)	2.57	1063242.0	214	682	0.19	0.0
r18.5(1.2)	0.88	2132925.0	24	820	0.36	0.0
r18.6(1.2)	11.1	3304429.0	484	1356	0.97	0.0
r18.7(1.2)	36000	-	-	-	-	2.54
r18.8(1.2)	36000	-	-	-	-	6.41
r18.9(1.2)	36000	-	-	-	-	13.77

way out of reach for exact approaches. Hence, an interesting research direction is to look into heuristic approaches for the problem. The combination between branching and capacity scaling presented in Katayama (2015) seems to be a promising avenue.

Acknowledgments

The first author wishes to thank the Danish Maritime Fund for sponsoring this project.

References

Agarwal, R. and Ergun, Ö. (2008). Ship scheduling and network design for cargo routing in liner shipping. *Transportation Science*, 42(2):175–196.

Akyüz, M. H. and Lee, C.-Y. (2016). Service type assignment and container routing with transit time constraints and empty container repositioning for liner shipping service networks. *Transportation Research Part B: Methodological*, 88:46–71.

Alibeyg, A., Contreras, I., and Fernández, E. (2016). Hub network design problems with profits. *Transportation Research Part E: Logistics and Transportation Review*, 96:40–59.

Balas, E. (1975). Facets of the knapsack polytope. *Mathematical Programming*, 8(1):146–164.

Barahona, F. (1996). Network design using cut inequalities. *SIAM Journal on Optimization*, 6(3):823–837.

Ben Amor, H., Desrosiers, J., and Valério de Carvalho, J. M. (2006). Dual-optimal inequalities for stabilized column generation. *Operations Research*, 54(3):454–463.

Brouer, B. D., Alvarez, J. F., Plum, C. E. M., Pisinger, D., and Sigurd, M. M. (2014). A base integer programming model and benchmark suite for liner-shipping network design. *Transportation Science*, 48(2):281–312.

Brouer, B. D., Desaulniers, G., Karsten, C. V., and Pisinger, D. (2015). A matheuristic for the liner shipping network design problem with transit time restrictions. In *International Conference on Computational Logistics*, pages 195–208. Springer.

Carraway, R. L., Morin, T. L., and Moskowitz, H. (1990). Generalized dynamic programming for multicriteria optimization. *European Journal of Operational Research*, 44(1):95–104.

Chouman, M., Crainic, T. G., and Gendron, B. (2016). Commodity representations and cut-set-based inequalities for multicommodity capacitated fixed-charge network design. *Transportation Science*, 51(2):650–667.

Chouman, M., Crainic, T. G., and Gendron, B. (2018). The impact of filtering in a branch-and-cut algorithm for multicommodity capacitated fixed charge network design. *EURO Journal on Computational Optimization*, 6(2):143–184.

Christiansen, M., Hellsten, E., Pisinger, D., Sacramento, D., and Vilhelmsen, C. (2019). Liner shipping network design. *European Journal of Operational Research*.

Cordeau, J.-F., Pasin, F., and Solomon, M. M. (2006). An integrated model for logistics network design. *Annals of Operations Research*, 144(1):59–82.

Costa, A. M., Cordeau, J.-F., and Gendron, B. (2009). Benders, metric and cutset inequalities for multicommodity capacitated network design. *Computational Optimization and Applications*, 42(3):371–392.

Crainic, T. G., Frangioni, A., and Gendron, B. (2001). Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics*, 112(1-3):73–99.

Crainic, T. G. and Gendreau, M. (2007). A scatter search heuristic for the fixed-charge capacitated network design problem. In *Metaheuristics*, pages 25–40. Springer.

Crainic, T. G., Ricciardi, N., and Storchi, G. (2009). Models for evaluating and planning city logistics systems. *Transportation Science*, 43(4):432–454.

Gendron, B., Crainic, T. G., and Frangioni, A. (1999). Multicommodity capacitated network design. In *Telecommunications Network Planning*, pages 1–19. Springer.

Gendron, B. and Larose, M. (2014). Branch-and-price-and-cut for large-scale multicommodity capacitated fixed-charge network design. *EURO Journal on Computational Optimization*, 2(1-2):55–75.

Ghamlouche, I., Crainic, T. G., and Gendreau, M. (2003). Cycle-based neighbourhoods for fixed-charge capacitated multicommodity network design. *Operations Research*, 51(4):655–667.

Guihaire, V. and Hao, J.-K. (2008). Transit network design and scheduling: A global review. *Transportation Research Part A: Policy and Practice*, 42(10):1251–1273.

Hammer, P. L., Johnson, E. L., and Peled, U. N. (1975). Facet of regular 0–1 polytopes. *Mathematical Programming*, 8(1):179–206.

Hellsten, E. (2020). Transit time constrained canad instances.

Hewitt, M., Nemhauser, G., and Savelsbergh, M. W. P. (2013). Branch-and-price guided search for integer programs with an application to the multicommodity fixed-charge network flow problem. *INFORMS Journal on Computing*, 25(2):302–316.

Hewitt, M., Nemhauser, G. L., and Savelsbergh, M. W. P. (2010). Combining exact and heuristic approaches for the capacitated fixed-charge network flow problem. *INFORMS Journal on Computing*, 22(2):314–325.

Holmberg, K. and Yuan, D. (2000). A lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. *Operations Research*, 48(3):461–481.

Holmberg, K. and Yuan, D. (2003). A multicommodity network-flow problem with side constraints on paths solved by column generation. *INFORMS Journal on Computing*, 15(1):42–57.

Karsten, C. V., Brouer, B. D., and Pisinger, D. (2017). Competitive liner shipping network design. *Computers & Operations Research*, 87:125–136.

Karsten, C. V., Pisinger, D., Røpke, S., and Brouer, B. D. (2015). The time constrained multi-commodity network flow problem and its application to liner shipping network design. *Transportation Research Part E: Logistics and Transportation Review*, 76:122–138.

Katayama, N. (2015). A combined capacity scaling and local branching approach for capacitated multi-commodity network design problem. *Far East Journal of Applied Mathematics*, 92(1):1–30.

Katayama, N., Chen, M., and Kubo, M. (2009). A capacity scaling heuristic for the multicommodity capacitated network design problem. *Journal of Computational and Applied Mathematics*, 232(1):90–101.

Kliewer, G. and Timajev, L. (2005). Relax-and-cut for capacitated network design. In *European Symposium on Algorithms*, pages 47–58. Springer.

Koza, D. F., Desaulniers, G., and Røpke, S. (2020a). Integrated liner shipping network design and scheduling. *Transportation Science*, 54(2):512–533.

Koza, D. F., Hellsten, E. O., and Pisinger, D. (2020b). Deep dual-optimal inequalities for generalized capacitated fixed-charge network design problems. Technical report. `https://ssrn.com/abstract=3704539`.

Lin, Z. and Kwan, R. S. K. (2013). An integer fixed-charge multicommodity flow (FCMF) model for train unit scheduling. *Electronic Notes in Discrete Mathematics*, 41:165–172.

Mehlhorn, K. and Ziegelmann, M. (2000). Resource constrained shortest paths. In *European Symposium on Algorithms*, pages 326–337. Springer.

Paraskevopoulos, D. C., Bektaş, T., Crainic, T. G., and Potts, C. N. (2016). A cycle-based evolutionary algorithm for the fixed-charge capacitated multi-commodity network design problem. *European Journal of Operational Research*, 253(2):265–279.

Pisinger, D. (2000). A minimal algorithm for the bounded knapsack problem. *INFORMS Journal on Computing*, 12(1):75–82.

Reinhardt, L. B. and Pisinger, D. (2011). Multi-objective and multi-constrained non-additive shortest path problems. *Computers & Operations Research*, 38(3):605–616.

Rodríguez-Martín, I. and Salazar-González, J. J. (2010). A local branching heuristic for the capacitated fixed-charge network design problem. *Computers & Operations Research*, 37(3):575–581.

Santini, A., Plum, C. E. M., and Røpke, S. (2018). A branch-and-price approach to the feeder network design problem. *European Journal of Operational Research*, 264(2):607–622.

Tong, L., Zhou, X., and Miller, H. J. (2015). Transportation network design for maximizing space–time accessibility. *Transportation Research Part B: Methodological*, 81:555–576.

Trivella, A., Corman, F., Koza, D. F., and Pisinger, D. (2020). The multi-commodity network flow problem with soft transit time constraints. *Available at SSRN 3537743*.

Wolsey, L. A. (1975). Faces for a linear inequality in 0–1 variables. *Mathematical Programming*, 8(1):165–178.

Yaghini, M., Rahbar, M., and Karimi, M. (2013). A hybrid simulated annealing and column generation approach for capacitated multicommodity network design. *Journal of the Operational Research Society*, 64(7):1010–1020.

Zetina, C. A., Contreras, I., and Cordeau, J.-F. (2019). Exact algorithms based on benders decomposition for multicommodity uncapacitated fixed-charge network design. *Computers & Operations Research*, 111:311–324.

Appendix: Complete benchmark results

Complete benchmark results for running the multi-model implementation using the base settings with dynamically generated cover inequalities and deep dual-optimal inequalities. For instances which were not solved in the given two hours the remaining gap is shown.

Instance	Run Time	Obj Value	Branching Nodes	Columns	Root Node Gap [%]	Remaining Gap [%]
r04.1(1.2)	0.26	34910.0	2	13	0	0.0
r04.2(1.2)	0.02	48933.0	2	18	0	0.0
r04.3(1.2)	0.02	63767.0	2	23	0	0.0
r04.4(1.2)	0.1	37607.0	70	33	1.52	0.0
r04.5(1.2)	0.12	55424.0	86	50	2.2	0.0
r04.6(1.2)	0.2	75998.0	136	74	3.51	0.0
r04.7(1.2)	0.4	68291.7	418	94	3.01	0.0
r04.8(1.2)	0.75	113004.0	814	144	5.7	0.0
r04.9(1.2)	0.46	163208.0	456	120	5.78	0.0
r05.1(1.2)	0.03	123653.0	16	50	0.38	0.0
r05.2(1.2)	0.07	171229.0	36	98	1.41	0.0
r05.3(1.2)	0.67	224790.0	282	295	4.55	0.0
r05.4(1.2)	0.13	133093.0	98	103	2.06	0.0
r05.5(1.2)	0.44	205866.0	210	202	5.56	0.0
r05.6(1.2)	3.39	290156.0	1218	488	7.61	0.0
r05.7(1.2)	0.08	278372.0	38	134	1.35	0.0
r05.8(1.2)	0.06	445810.0	40	122	2.19	0.0
r05.9(1.2)	0.12	625879.0	42	127	1.45	0.0
r07.1(1.2)	0.02	32807.0	2	11	0	0.0
r07.2(1.2)	0.02	47252.0	4	20	0.05	0.0
r07.3(1.2)	0.04	62962.0	10	43	1.31	0.0
r07.4(1.2)	0.34	37432.0	266	46	2.45	0.0
r07.5(1.2)	2.09	56475.0	1770	131	5.47	0.0
r07.6(1.2)	6.77	77249.0	4912	259	7.57	0.0
r07.7(1.2)	1.28	59947.0	1242	153	3.33	0.0
r07.8(1.2)	2.25	99194.0	1818	259	6.63	0.0
r07.9(1.2)	4.47	141692.0	3598	360	8.23	0.0
r08.1(1.2)	0.05	103155.0	24	52	0.57	0.0
r08.2(1.2)	0.03	145314.0	2	72	0	0.0
r08.3(1.2)	0.05	184706.0	6	101	0.66	0.0
r08.4(1.2)	0.07	109325.0	40	57	0.79	0.0
r08.5(1.2)	0.29	157527.0	148	156	2.52	0.0
r08.6(1.2)	0.4	207540.0	168	175	3.47	0.0
r08.7(1.2)	2.2	154316.0	1444	251	3.01	0.0
r08.8(1.2)	15.71	277614.0	7614	557	7.36	0.0
r08.9(1.2)	19.58	422798.0	9258	700	9.15	0.0
r09.1(1.2)	0.03	196939.0	2	97	0	0.0
r09.2(1.2)	0.11	358765.0	28	174	0.72	0.0
r09.3(1.2)	0.61	523181.0	92	318	1.26	0.0
r09.4(1.2)	0.24	208463.0	136	149	0.92	0.0
r09.5(1.2)	2.15	402549.0	710	330	2.24	0.0
r09.6(1.2)	2.8	592600.0	556	491	2.66	0.0
r09.7(1.2)	0.31	345188.0	180	204	0.98	0.0
r09.8(1.2)	1.83	668786.0	890	352	1.58	0.0

r09.9(1.2)	1.34	1019379.0	674	333	1.65	0.0
r10.1(1.2)	0.13	205624.0	44	97	0.63	0.0
r10.2(1.2)	1.31	356423.0	294	359	1.98	0.0
r10.3(1.2)	3.1	518530.0	394	669	2.48	0.0
r10.4(1.2)	6.05	229251.0	2626	265	1.9	0.0
r10.5(1.2)	19.32	414217.0	3944	697	3.88	0.0
r10.6(1.2)	37.48	621225.0	5616	1308	5.14	0.0
r10.7(1.2)	8.88	486895.0	3004	700	4.19	0.0
r10.8(1.2)	6.78	956975.0	2146	776	6.32	0.0
r10.9(1.2)	10.29	1456225.0	3718	767	7.09	0.0
r11.1(1.2)	0.54	721506.0	88	376	0.17	0.0
r11.2(1.2)	123.62	1292385.0	5422	2158	2.09	0.0
r11.3(1.2)	489.93	1902536.0	10652	3977	4.33	0.0
r11.4(1.2)	87.7	877784.0	12604	984	1.25	0.0
r11.5(1.2)	82.02	1646457.0	4850	1817	2.43	0.0
r11.6(1.2)	245.09	2461419.0	10516	2935	3.44	0.0
r11.7(1.2)	0.17	2294911.0	32	543	0.22	0.0
r11.8(1.2)	1.19	3547810.0	312	588	0.34	0.0
r11.9(1.2)	12.26	4840860.0	3016	756	0.46	0.0
r12.1(1.2)	2.53	1660054.0	234	839	0.62	0.0
r12.2(1.2)	371.98	3446690.0	6742	3267	3.1	0.0
r12.3(1.2)	2077.57	5343420.0	27360	7073	5.26	0.0
r12.4(1.2)	6.88	2311304.0	540	1205	0.52	0.0
r12.5(1.2)	3.99	4725130.0	266	1372	0.79	0.0
r12.6(1.2)	3.7	7246910.0	198	1560	0.42	0.0
r12.7(1.2)	0.18	7635271.0	12	970	0.02	0.0
r12.8(1.2)	3.79	10447170.0	602	1126	0.13	0.0
r12.9(1.2)	6.76	13562030.0	900	1160	0.24	0.0
r13.1(1.2)	0.03	151993.0	2	83	0	0.0
r13.2(1.2)	0.18	278147.0	12	287	0.16	0.0
r13.3(1.2)	1.62	390683.0	90	629	1.43	0.0
r13.4(1.2)	1.58	159562.0	584	137	1.02	0.0
r13.5(1.2)	23.67	301707.0	3558	636	2.49	0.0
r13.6(1.2)	26.54	427826.0	1952	1154	3.21	0.0
r13.7(1.2)	3221.96	211999.0	548196	990	3.45	0.0
r13.8(1.2)	36000	-	-	-	-	0.42
r13.9(1.2)	36000	-	-	-	-	36.09
r14.1(1.2)	0.29	457481.0	60	248	0.19	0.0
r14.2(1.2)	0.21	869124.0	8	453	0.15	0.0
r14.3(1.2)	1.5	1271242.0	60	766	0.44	0.0
r14.4(1.2)	4.34	479541.0	966	346	0.64	0.0
r14.5(1.2)	2.36	927462.0	146	677	0.99	0.0
r14.6(1.2)	5.89	1368722.0	226	1082	0.99	0.0
r14.8(1.2)	36000	-	-	-	-	0.55
r14.9(1.2)	36000	-	-	-	-	9.17
r15.1(1.2)	11.99	1122929.0	1092	886	0.41	0.0
r15.2(1.2)	1535.35	2379790.0	17452	4537	1.45	0.0
r15.3(1.2)	57.63	3580850.0	542	2991	1.05	0.0
r15.4(1.2)	153.76	1233428.0	10330	1393	0.75	0.0
r15.5(1.2)	459.12	2701550.0	7280	3495	1.36	0.0
r15.6(1.2)	435.55	4208750.0	4840	5511	2.12	0.0
r15.7(1.2)	19.36	2300560.0	1438	1718	0.69	0.0
r15.8(1.2)	39.06	5982840.0	2416	2042	0.89	0.0
r15.9(1.2)	116.12	10333880.0	5804	2561	0.95	0.0

r16.1(1.2)	0.09	140822.0	2	114	0	0.0
r16.2(1.2)	0.48	246851.0	18	480	0.33	0.0
r16.3(1.2)	1.4	336252.0	20	769	0.32	0.0
r16.4(1.2)	0.3	142669.0	62	164	0.27	0.0
r16.5(1.2)	6.22	254426.0	256	826	1.53	0.0
r16.6(1.2)	12.93	351625.0	388	1448	2.28	0.0
r16.7(1.2)	36000	-	-	-	-	21.59
r16.8(1.2)	36000	-	-	-	-	77.1
r16.9(1.2)	36000	-	-	-	-	32.7
r17.1(1.2)	0.12	422439.0	2	220	0	0.0
r17.2(1.2)	1.18	813651.0	36	673	0.7	0.0
r17.3(1.2)	0.89	1198281.0	20	713	0.23	0.0
r17.4(1.2)	0.36	425627.0	44	268	0.19	0.0
r17.5(1.2)	7.98	839583.0	560	860	0.84	0.0
r17.6(1.2)	7.18	1236717.0	286	1071	0.94	0.0
r17.7(1.2)	2027.23	515404.0	274148	1057	1.8	0.0
r17.8(1.2)	16434.9	1143413.0	660126	4478	3.48	0.0
r17.9(1.2)	36000	-	-	-	-	10.85
r18.1(1.2)	1.11	1026239.0	112	585	0.14	0.0
r18.2(1.2)	1.8	2059575.0	44	1053	0.52	0.0
r18.3(1.2)	3.34	3131660.0	122	1157	0.58	0.0
r18.4(1.2)	2.57	1063242.0	214	682	0.19	0.0
r18.5(1.2)	0.88	2132925.0	24	820	0.36	0.0
r18.6(1.2)	11.1	3304429.0	484	1356	0.97	0.0
r18.7(1.2)	36000	-	-	-	-	2.54
r18.8(1.2)	36000	-	-	-	-	6.41
r18.9(1.2)	36000	-	-	-	-	13.77
c33(1.2)	0.28	425117.0	26	74	0.13	0.0
c35(1.2)	4.71	372254.0	1408	136	0.75	0.0
c36(1.2)	45.2	644011.0	11550	239	1.56	0.0
c37(1.2)	5.33	105127.0	80	1225	0.65	0.0
c38(1.2)	1.32	178851.0	66	793	0.31	0.0
c39(1.2)	4.21	110486.0	116	1005	0.38	0.0
c40(1.2)	29.12	159543.0	658	1512	0.57	0.0
c41(1.2)	0.06	430116.0	12	64	0.36	0.0
c42(1.2)	1.9	594506.0	304	152	1.02	0.0
c43(1.2)	5.84	464509.0	1552	148	0.72	0.0
c44(1.2)	15.95	619290.0	4202	207	1.51	0.0
c45(1.2)	2.17	84314.0	82	803	0.3	0.0
c46(1.2)	15.98	139044.0	322	1325	0.8	0.0
c47(1.2)	3.98	86317.0	218	792	0.35	0.0
c48(1.2)	32.51	119670.0	708	1378	0.85	0.0
c49(1.2)	35.86	56036.0	3116	489	0.48	0.0
c50(1.2)	16969.3	103242.0	349486	3336	1.99	0.0
c51(1.2)	23329.3	52778.0	1621234	2272	0.88	0.0
c52(1.2)	1945.88	103876.0	67060	1294	1.22	0.0
c53(1.2)	99.67	120994.0	1330	1807	0.27	0.0
c54(1.2)	19637.0	169769.0	188368	3996	0.71	0.0
c55(1.2)	3099.4	124865.0	46234	2206	0.39	0.0
c56(1.2)	7662.24	170137.0	52300	3250	0.57	0.0
c57(1.2)	26.39	49072.0	1774	536	0.54	0.0
c58(1.2)	167.55	64247.0	6598	929	0.81	0.0
c59(1.2)	36000	-	-	-	-	0.11
c60(1.2)	36000	-	-	-	-	0.56