Master's Thesis Econometrics & Operations Research

Optimization of two-phase methods using simple feedback mechanisms

Wouter Kool

1937189

VU Supervisor: Prof. Dr. J. Gromicho

VU Co-Reader: Prof. Dr. L. Stougie

ORTEC Supervisor: Dr. Ir. G. Post

VU University, Amsterdam

Faculty of Economics and Business Administration

Master Econometrics & Operations Research

De Boelelaan 1105

1081 HV Amsterdam

ORTEC

Houtsingel 5

2719 EA Zoetermeer

December 2014

ORTEC

PROFESSIONALS IN PLANNING

Abstract

In this thesis, we present a new framework for the optimization of two-phase methods using a simple feedback mechanism. The framework is based on the idea that there must exist a cost metric for the first phase, such that optimization of the first phase according to that cost metric will result in an intermediate solution, which leads to the optimal solution if it is used as input for the second phase. We present ideas for defining an a priori belief for this cost metric and we present an updating mechanism that can be used to update the belief according to observations. This process is repeated, until the belief is no longer updated, such that the method can be regarded as a special form of a fixed point iteration. We apply the framework to three different problems: the pallet matching problem, the capacitated vehicle routing problem, and the resource assignment problem. We observe that using the framework we are able to significantly improve on the initial solution quality within just a few iterations, but the applicability of the framework becomes increasingly more challenging from a modelling point of view alongside with the modelling complexity of the problem being solved.

Contents

1 Introduction

Optimization problems are often solved using multi-phase methods in practice, because optimization of the entire problem is computationally intractable. An example is the airline scheduling problem, where phases may consist of timetabling, fleet-assignment and crew-pairing. Typically these multi-phase methods are somewhat intuitive with respect to the problem and therefore align with the operational processes. In other cases, phases are less natural, but introduced to break a problem down into simpler subproblems. An example is the vehicle routing problem (VRP), for which many solution methods divide the problem in a clustering problem and a collection of independent (single vehicle) routing problems.

What characterizes a multi-phase approach is that each consecutive phase is optimized given the results of the preceding phase(s) and with respect to some cost metric defined for that phase. This means that, depending on the method used to solve the problem in each phase, the decisions made in each phase are at most *conditionally optimal* given the preceding phases. Since the objective is to optimize the overall problem, the decisions made in each phase should ideally correspond to the decisions required to find the global optimum. For each suboptimal decision in each of the phases, we may loose a part of the global optimality of the problem. Therefore, the cost metric in each phase should be such that good solutions with respect to that cost metric are also part of good solutions for the global problem.

In many applications, each phase is executed just once, which means that there is no feedback from subsequent phases. This makes the overall approach effectively a greedy heuristic with respect to the phases. There are a number of techniques in which the problems for each phase are solved repeatedly to find an overall solution, such as Column Generation (Barnhart et al., 1998) and Lagrange decomposition (Bellman, 1956). Using these techniques, subproblems are optimized with additional information in the form of shadow prices or Lagrange multipliers. These methods have proven to be very powerful, but their application is limited to special classes of problems. More generally applicable are metaheuristics such as Genetic Algorithms (Davis et al., 1991), Simulated Annealing (Van Laarhoven and Aarts, 1987) and Tabu Search (Glover and Laguna, 1999). All of these metaheurstics somehow 'learn' from earlier decisions in the sense that they generate new solutions based on what has been observed before. However, these methods do not explicitly take into account problems consisting of multiple phases.

In this thesis, we will introduce a general concept for a simple feedback mechanism for optimization of a two-phase method. The idea is that in each iteration the cost metric for the first

phase is updated in such a way that it converges to the 'true' cost metric for the overall problem. The mechanism is stopped if the cost metric no longer changes, which is why the method can be regarded as a special form of fixed point iteration (Hyvarinen, 1999). Using the true cost metric for the first phase means that solving the first phase optimally and the second phase conditionally optimal results in the optimal overall solution. We will illustrate how this is implemented for three different problems: the *pallet matching problem*, which is a variant of a three dimensional matching problem, the famous *capacitated vehicle routing problem* and the *resource assignment problem*.

The remainder of this thesis is structured as follows. In Section 2 we will develop the theoretical framework that we use to implement our iterative feedback mechanism. In Section 3 we will apply it to the pallet matching problem, in Section 4 we do the same for the capacitated vehicle routing problem and in Section 5 we apply the framework to the resource assignment problem. The thesis concludes with a discussion in Section 6.

2 Theoretical framework

In this section, we develop the theoretical framework which we use to optimize in general a multi-phase approach. We focus on two-phase methods only using the observation that any multiple-phase method can be regarded as multiple nested two-phase methods.

2.1 Notation

Let x be an instance of the problem. Let $F_1(x, c_1)$ be the method that executes phase 1 of the optimization problem using the cost metric c_1. Let $y = F_1(x, c_1)$ be the intermediate result after phase 1, so $F_1(x, c_1) = \arg\min_{y \in Y(x)} c_1(x, y)$. $Y(x)$ is the set of all possible intermediate solutions for input x and the metric c_1 is defined as a function of x and y giving the costs for intermediate solution y for instance x. For phase 2, we have $z = F_2(x, y)$, where z is the result of phase 2. Also, we have a cost metric c_2 for phase 2, which typically matches the overall cost metric c and which we assume is known. In other words, we assume that for any global solution z in the domain $Z(x)$, we are able to calculate $c_2(x, z) = c(x, z)$ as the 'cost' of the solution. Again, F_2 can be written as $F_2(x, y, c_2) = \arg\min_{z \in Z(y)} c_2(x, z)$, where $Z(y)$ is the set of feasible solutions which have y as intermediate solution, and we write $F_2(x, y) = F_2(x, y, c_2)$ because we assume c_2 is known and fixed. Let $Z(x)$ be the set of all feasible solutions for the instance x. It is clear that $Z(y) \subseteq Z(x)$. We assume that for every solution z there is a exactly one intermediate solution y, which can be observed as the projection of $Z(y)$ on the domain $Y(x)$. As a result it holds that $Z(y) \cap Z(y') = \emptyset \quad \forall y, y' \in Y(x), y \neq y'$ and that $Z(x) = \bigcup_{y \in Y(x)} Z(y)$. See Figure 1 for an illustration.

2.2 Optimality of the solution

We assume that there is a unique optimal solution z^* for the problem and we define $F(x)$ to be the utopian method that gives this optimal solution z^* for any instance x, so $z^* = F(x) = \arg\min_{z \in Z(x)} c(x, z)$. In order to find z^* using the two phase approach we need that $z^* \in Z(y)$ for $y = F_1(x, c_1)$. Since for every z there is only one y such that $z \in Z(y)$, there is a unique optimal intermediate solution which we denote with y^*. This means that we need to find a cost metric c_1 such that $y^* = F_1(x, c_1)$. If this is the case, we will say that c_1 *represents the optimum.*

Definition 1. *A cost metric c_1 represents the optimum if $y^* = F_1(x, c_1)$.*

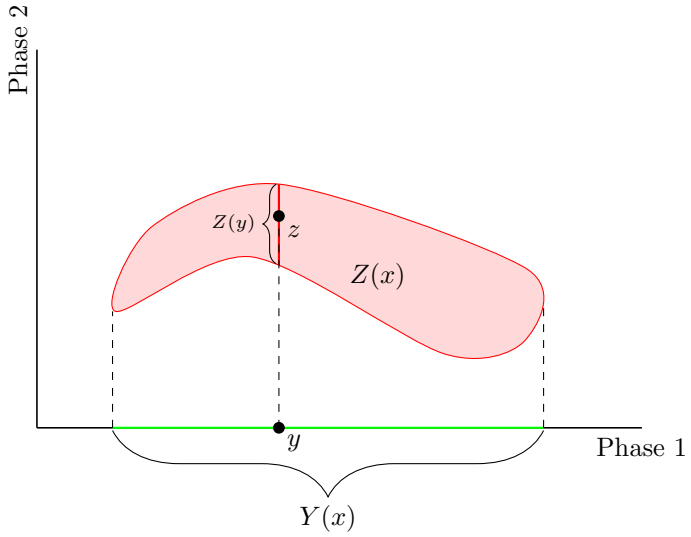


Figure 1: Visualization of domains

If we do not require any special structure of c_1, we can show that there always exists a cost metric that satisfies this property.

Proposition 1. *A cost metric c_1 represents the optimum **iff** $c_1(x, y) > c_1(x, y^*) \quad \forall y \in Y(x) \setminus \{y^*\}$.*

Proof. If c_1 represents the optimum, it must hold that $y^* = F_1(x, c_1) = \arg\min_{y \in Y(x)} c_1(x, y)$, so it must hold that $c_1(x, y) > c_1(x, y^*) \quad \forall y \in Y(x) \setminus \{y^*\}$. On the other hand, if it holds that $c_1(x, y) > c_1(x, y^*) \quad \forall y \in Y(x) \setminus \{y^*\}$ than y^* is the optimum according to c_1, so c_1 represents the optimum. □

Using Proposition 1, it is trivial to show that the function $c_1(x, y) = \mathbb{1}_{\{y \neq y^*\}}$ represents the optimum. Unfortunately, the problem of finding this metric is equivalent to the problem itself.

Proposition 2. *A sufficient condition for c_1 to represent the optimum y^*, z^* is that $c_1(x, y) > c_1(x, y')$ if $c(x, F_2(x, y)) > c(x, F_2(x, y')) \quad \forall y, y' \in Y(x)$.*

Proof. Let $y' = y^*$ and $y \in Y(x) \setminus \{y^*\}$. Since $z^* = F_2(x, y^*)$ is the unique optimum, $c(x, F_2(x, y)) > c(x, F_2(x, y^*)) \quad \forall y \in Y(x) \setminus \{y^*\}$. If we ensure that $c_1(x, y) > c_1(x, y')$ if $c(x, F_2(x, y)) > c(x, F_2(x, y')) \quad \forall y, y' \in Y(x)$, then from this it follows that $c_1(x, y) > c_1(x, y^*)$ for $y \in Y(x) \setminus \{y^*\}$. Since this holds $\forall y \in Y(x) \setminus \{y^*\}$, it follows from Proposition 1 that c_1 represents the optimum. □

Proposition 3. *A stronger sufficient condition for c_1 to represent the optimum y^*, z^* is that* $c_1(x, y) = c(x, F_2(x, y)) \quad \forall y \in Y(x)$.

Proof. If $c_1(x, y) = c(x, F_2(x, y)) \quad \forall y \in Y(x)$ then $c(x, F_2(x, y)) > c(x, F_2(x, y')) \Leftrightarrow c_1(x, y) > c_1(x, y') \quad \forall y, y' \in Y(x)$, such that c_1 represents the optimum according to Proposition 2. □

From Proposition 3 it follows that c_1 represents the optimum if $c_1(x, y) = c(x, F_2(x, y))$. This means that c_1 gives for each y the actual cost for the optimal z that follows from this y. This leads to the aforementioned similarity with fixed point iteration.

2.3 The feedback mechanism

The feedback mechanism that we introduce is based on the observation that using the metric $c_1^*(x, y) = c(x, F_2(x, y))$ for the first phase will result in the optimal solution being found (Proposition 3). We start with an a priori belief $\tilde{c}_1$ of c_1^* and we iteratively update this belief based on observations, with as goal to let $\tilde{c}_1$ converge to c_1^*. More precisely, if an intermediate solution $\hat{y}$ results in a solution $\hat{z}$, we update $\tilde{c}_1$ such that $\tilde{c}_1(x, \hat{y}) = c(x, \hat{z})$. Important is that we repeat this procedure until it is no longer needed to update $\tilde{c}_1$, so when $c(x, \hat{z}) = \tilde{c}_1(x, \hat{y})$. In that case, we do not update $\tilde{c}_1$ and it is easy to see that the next iterations will result in the same solution, in which case we say that $\tilde{c}_1$ has *converged*. This means that we can immediately stop and we converge in the same sense as in fixed point methods.

Definition 2. *$\tilde{c}_1$ has converged if as a result of the feedback mechanism $\tilde{c}_1$ has been updated such that it holds that $\tilde{c}_1(x, \hat{y}) = c(x, F_2(x, \hat{y}))$ for $\hat{y} = F_1(x, \tilde{c}_1)$.*

Note that the update in each next iteration may invalidate the equality $\tilde{c}_1(x, \hat{y}) = c(x, \hat{z})$ for the previous iteration. However, depending on the structural properties of c_1, there may be some freedom in the way c_1 is updated, such that additional conditions can be set as constraints for the update. We have observed that in this sense it helps to require that $\tilde{c}_1$ not only represents the true costs for the last observation, but also for the best observation so far (the *incumbent*). This will become clear in the example in Section 4.

2.4 Optimistic a priori belief

Letting $\tilde{c}_1$ represent the incumbent solution at any time helps the converge of $\tilde{c}_1$. However, in order to avoid premature convergence to a local optimum, there is another condition that should be met: the a priori belief $\tilde{c}_1$ should be optimistic towards 'better' intermediate solutions y.

Definition 3. *A cost metric c_1 is optimistic towards an intermediate solution y if $c_1(x, y) \leq c(x, F_2(x, y))$.*

This is because if it is too conservative (pessimistic), the method becomes a 'self fulfilling prophecy'. To see this, observe that in the first iteration, the intermediate solution $\hat{y}$ that optimizes $\tilde{c}_1$ is found. If the metric is pessimistic, then it will hold that actual value $c(x, F_2(x, \hat{y}))$ is lower than $\tilde{c}_1(x, \hat{y})$. As a result, $\tilde{c}_1$ gets updated (decreased) such that $\hat{y}$ is positively reinforced, despite it already being optimal. For other solutions y the value of $\tilde{c}_1$ may also decrease (if they are similar to $\hat{y}$), but probably not as much as for $y = \hat{y}$, so typically in the next iteration the same y is found. This means that the method immediately terminates with a solution $\hat{z} = F_2(x, \hat{y})$, of which the quality depends on the quality of the a priori belief $\tilde{c}_1$. If, on the contrary, $\tilde{c}_1$ is taken to be optimistic, then for any y the reinforcement is negative, such that alternate solutions will be found in the next iteration. This continues until the solution no longer changes. In this case the solution is considered optimal, even after the negative reinforcement and when it is compared to the optimistic belief for (unseen) alternative solutions. The more optimistic the a priori belief is, the more different solutions are explored. This means that the level of optimism can be used as a parameter to steer the algorithm towards global search, usually at the expense of additional iterations.

2.5 The algorithm

Algorithm 1 shows the global steps of the method. It is worked out in detail for the three problems in Sections 3, 4 and 5.

Algorithm 1 Simple feedback mechanism.

```
1: procedure SimpleFeedback(x, F_1, F_2, c̃_1, c)
2:     while c̃_1 has not converged do
3:         ŷ ← F_1(x, c̃_1)
4:         ẑ ← F_2(x, ŷ)
5:         c̃_1 ← Update(c̃_1, ŷ, ẑ, c)        ▷ Update c̃_1(x, ŷ) := c(x, ẑ)
6:     end while
7: end procedure
```

3 The pallet matching problem

3.1 The problem

To introduce the pallet matching problem, we consider the operational planning of an automated warehouse that contains pallets. At the warehouse, *orders* arrive that consist of a number of orderlines, which we call *requests*. Each request needs to be matched with a pallet in the warehouse, where some requests can be fulfilled by multiple pallets in the warehouse. The requests need to be matched in pairs of two to be stacked for transportation. Since the requests already contain information about the physical properties of the pallets that can be used for each request, information about feasible stacking possibilities is available at the request level. If two requests are stacked, the corresponding pallets need to be retrieved from the warehouse in the proper sequence. First the top pallet is retrieved, and is lifted while the bottom pallet is retrieved and arrives below the top pallet. Depending on the positions of the pallets in the warehouse, parts of the automated retrieval may be done in parallel. This means that the total time it takes to retrieve two pallets depends on the combination of the pallets. The problem is how to stack the requests and match the pallets to requests such that the total retrieval time for the resulting combinations of pallets is minimized.

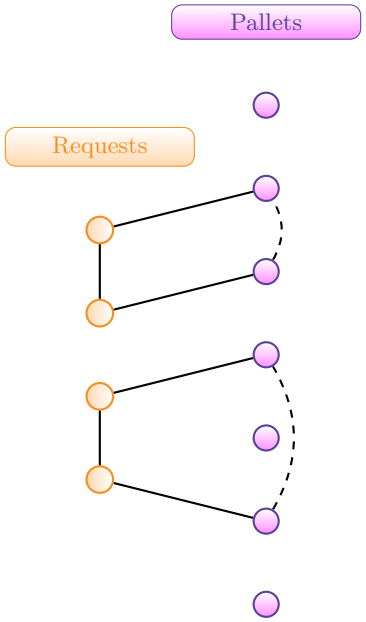


Figure 2: Visualization of the pallet matching problem

Figure 2 is a schematic representation of the problem. The orange nodes indicate requests, while the purple nodes correspond to pallets. The lines indicate the matches of pallets to requests and the requests that are stacked, but the retrieval time is defined by the resulting combinations of pallets, as indicated by the dashed lines. Since the retrieval times should be minimized, we sometimes refer to them as the 'costs' for the combinations of pallets. Formally the order of the pallets within the stack does matter, but since the optimal order can easily be determined, we define the costs for each combination as the minimum retrieval time for the optimal order of the pallets within the stack, respecting feasible stacking constraints.

3.2 Mathematical problem formulation

The pallet matching problem can be expressed as a mathematical program using the following notation:

- Set of pallets P
- Set of requests R
- Symmetric 'cost matrix' T with $t_{ij} = t_{ji}$ the retrieval time for a combination of pallets $i \in P$ and $j \in P$
- Matrix A with $a_{ik} = 1$ if pallet $i \in P$ can be used to fulfill request $k \in S$
- Symmetric matrix B with $b_{k\ell} = 1$ if *requests* $k \in S$ and $\ell \in S$ can be stacked

Using matrix algebra, it can be seen that the matrix ABA' contains a nonzero entry for every combination of pallets that can potentially be stacked.

We introduce the following decision variables:

$$x_{ij} = \begin{cases} 1 & \text{if pallet } i \text{ and } j \text{ form a combination} \\ 0 & \text{otherwise} \end{cases} \qquad i, j \in P, (ABA')_{ij} > 0$$

$$y_{ik} = \begin{cases} 1 & \text{if pallet } i \text{ is matched to request } k \\ 0 & \text{otherwise} \end{cases} \qquad i \in P, k \in R, a_{ik} = 1$$

Now the pallet matching problem can be expressed as a Mixed Integer Program (MIP) as follows:

$$\min \quad \sum_{(i,j):(ABA')_{ij}>0} t_{ij}x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{j:(ABA')_{ij}>0} x_{ij} + x_{ji} = \sum_{k:a_{ik}=1} y_{ik} \qquad i \in \mathcal{P} \tag{2}$$

$$\sum_{i:a_{ik}=1} y_{ik} = 1 \qquad k \in R \tag{3}$$

$$\sum_{k:a_{ik}=1} y_{ik} \leq 1 \qquad i \in P \tag{4}$$

$$y_{ik} + x_{ij} + y_{j\ell} \leq 2 \qquad i,j \in P, k,\ell \in R, a_{ik} = 1, a_{j\ell} = 1, b_{k\ell} = 0 \tag{5}$$

$$x_{ij} \in \{0,1\} \qquad i,j \in P, (ABA')_{ij} > 0 \tag{6}$$

$$y_{ik} \in \{0,1\} \qquad i \in P, k \in R, a_{ik} = 1 \tag{7}$$

The formulation is explained below:

- (1) The objective is to minimize the total retrieval time for all combinations of pallets that are selected.
- (2) Pallets should be combined with another pallet if and only if they are matched to a request.
- (3) Each request should be fulfilled by exactly one pallet.
- (4) Each pallet can fulfill at most one request.
- (5) Prevent that a combination of pallets together with the matchings of these pallets to requests corresponds to a non-stackable combination of requests.
- (6), (7) Define decision variables.

The number of constraints (5) can theoretically be as large as $(|P| \cdot |R|)^2$, so that it is impractical to include them explicitly. Since only very few of the constraints are required for finding the optimal solution, they should be added to the problem lazily. In our implementation, we do this by solving the problem without the constraint after which we check if any of the resulting matches is infeasible. If so, we add the corresponding constraint (5) and resolve the problem, until no more infeasible matches occur. Using Gurobi (Gurobi Optimization, 2014) as a MIP solver, this procedure enables us to find the optimal solution which we use to benchmark the results of our framework.

3.3 Decomposition of the problem

In order to apply the framework we described in Section 2 we need to decompose the problem into two subproblems that can subsequently be solved. Although there are multiple ways in which this may be done, we choose the following method. In the first phase, we make stacks of requests by solving a (non-bipartite) weighted matching problem. In the second phase the pallets are matched to requests. Although the second phase looks like a bipartite matching problem, this is not the case since costs depend on the resulting combinations of pallets rather than directly on the matches of pallets to requests. The problem may be formulated as a weighted set packing problem, but this problem is NP-hard (Crescenzi and Kann, 1995). Therefore we solve it using the MIP formulation from Section 3.2, where the matrix B is updated such that only the fixed stacks of requests are allowed. This severely limits the complexity of the problem since the number of variables is reduced as $(ABA')_{ij}$ now only has nonzero entries for combinations of pallets that correspond to the fixed stacks of requests. This leads to instances which can be solved in reasonable time using the Gurobi solver.

3.4 Filling in the ingredients

By decomposing the problem into two subproblems we have the ingredients needed to apply the simple feedback mechanism.

3.4.1 The intermediate solution y

The intermediate solution y is a matching of requests. We write $y_{k\ell} = 1$ if request k is matched to request ℓ and $y_{k\ell} = 0$ otherwise.

3.4.2 The cost metric c_1

$c_1(x, y)$ gives for the matching y the costs and can therefore be represented by a cost *matrix* D. The matrix should either be symmetric or only the upper triangular part should be considered. If we let the entry $d_{k\ell}$ from the matrix represent the cost for matching requests k and ℓ, then $c_1(x, y) = \sum_{k,\ell \in R:k<\ell} y_{k\ell} d_{k\ell}$.

3.4.3 The function F_1

$F_1(x, c_1)$ solves the general weighted matching problem using the cost metric c_1, thus using the cost matrix D. A minimum cost perfect matching is found using the LEMON (2014) matching

library. This library solves the general (non-perfect) weighted *maximum* matching problem, but it can be shown that by using the weight matrix $W = K - D$, for a sufficiently large constant K, the solution corresponds to the minimum cost maximum cardinality matching for D. This matching will be perfect if and only if a perfect matching exists. If no perfect matching exists, the problem is infeasible since not all requests can be stacked, regardless of D.

3.4.4 The function F_2

$F_2(x, y)$ solves the problem of optimally matching the pallets to the requests given the request matchings in y. As was described in Section 3.3, the problem is solved using the MIP formulation with the matrix B updated according to the matchings of pallets. More precisely, B is updated such that $b_{k\ell} = y_{k\ell}$.

3.4.5 The a priori belief $\tilde{c}_1$

The chosen decomposition of the pallet matching problem is a perfect example for which an optimistic a priori belief for $\tilde{c}_1$ can be defined. This is because c_1 decomposes into independent components corresponding to $d_{k\ell}$ for which we can calculate a lower bound. In words, for any combination of requests k and ℓ we can define $\tilde{d}_{k\ell}$ (corresponding to $\tilde{c}_1$) as the minimum costs over all possible combinations of pallets that can result from this combination of requests:

$$\tilde{d}_{k\ell} = \min_{i,j \in P: a_{ik}=1, a_{j\ell}=1, i \neq j} t_{ij} \quad \forall k, \ell \in R. \tag{8}$$

Figure 3 uses an example to illustrate how the entry $\tilde{d}_{12}$ is calculated. Top left is the stackability matrix B, top right and bottom left are the pallet request matrix A and bottom right is the cost matrix T, so the full matrix is $\left(\begin{smallmatrix} B & A' \\ A & T \end{smallmatrix}\right)$.

The a priori belief $\tilde{c}_1$ as defined in Equation 8 is based on the assumption that we can match to each pair of requests the best pair of pallets. This is optimistic because this assumption implies that we assume we do not need one of the pallets of the best pair to fulfill other requests. However, it is never pessimistic since the costs of the final matching cannot be lower than the lower bound.

3.4.6 The updating mechanism of $\tilde{c}_1$

If after the first iteration for the solution z it holds that $c(x, z) = \tilde{c}_1(x, y)$, the algorithm has immediately found the optimal solution, since $\tilde{c}_1$ represents the lower bound. Typically, we will find a value $c(x, z) > \tilde{c}_1(x, y)$, which means that we need to adjust the values of $\tilde{d}_{k\ell}$. Note that

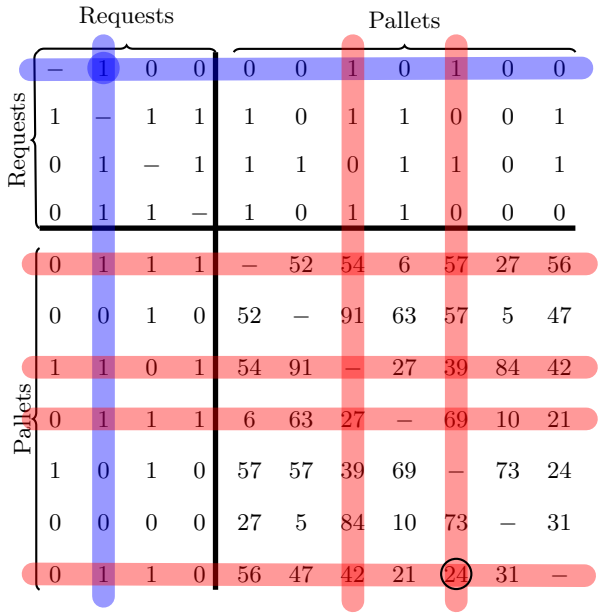


Figure 3: Calculating entry $\tilde{d}_{12}$ from the lower bound matrix $\tilde{D}$ for $\tilde{c}_1$

$c(x, z)$, the final cost of the solution, is the sum of the costs for the resulting pairs of pallets. The costs for each pair of pallets can directly be linked to the pairs of requests, so the trivial way of updating $\tilde{c}_1(x, y)$ to represent the actual cost is by updating the values of $\tilde{d}_{k\ell}$ according to the observed costs for the pairs of pallets corresponding to the pairs of requests. In more formal notation, write the solution z as z_k representing the pallet that is matched to request k. Then we update $\tilde{d}$ using the following formula:

$$\tilde{d}_{k\ell} = t_{z_k, z_\ell} \quad \forall k, \ell \in R : y_{k\ell} = 1. \tag{9}$$

3.5 Simple feedback mechanism for the pallet matching problem

Filling in the steps from Section 3.4 in the algorithm in Section 2.5 results in Algorithm 2 for the pallet matching problem.

Algorithm 2 Simple feedback mechanism.

```
1: procedure SimpleFeedbackPalletMatching(x)
2:     D ← calculateLowerBoundMatrix(x)
3:     while true do
4:         ŷ ← matchRequests(x, D)
5:         ẑ ← matchPallets(x, ŷ)
6:         if c̃1(x, ŷ, D) == c(x, ẑ) then return
7:         end if
8:         D ← Update(D, ŷ, ẑ, c)                    ▷ Update c̃1(x, ŷ) := c(x, ẑ)
9:     end while
10: end procedure
```

3.6 Variant: a heuristic approach for the second subproblem

For the pallet matching problem, solving the NP-complete second subproblem to optimality is done using the MIP formulation. This may be impractical if the size of the instance grows, especially if it has to be done repeatedly using the feedback mechanism. For most practical problems, we would run into the same problem: it is very rarely possible to solve the subproblems to optimality. This raises the question whether the simple feedback mechanism still remains applicable if heuristics are used to solve the subproblems. To get some insight in how this works, we propose to replace the optimal procedure $F_2(x, y)$ with a heuristic method, which we will denote with $\tilde{F}_2(x, y)$.

3.6.1 The function $\tilde{F}_2$

As a heuristic method $\tilde{F}_2(x, y)$ for the second subproblem, we also choose to repeatedly solve simple matching problems. This time, the matching problem is a bipartite matching problem because each match consists of one request and one pallet. Let $m(k)$ be the request to which request k is matched in the first step, so $\ell = m(k)$ if request k is matched to ℓ. In other words, $\ell = m(k)$ if and only if $y_{k\ell} = 1$, and this also implies that $k = m(\ell)$. We define the entry e_{ik} of the matrix E for matching pallet i to request k as the minimum costs that result, if, given the matching of i to k, the optimal pallet j is matched to request $\ell = m(k)$:

$$e_{ik} = \min_{j:a_{j,m(k)}=1} t_{ij}. \tag{10}$$

Note that this 'best response' is mutually exclusive, thus if $(i, j) = \arg\min_{i,j:a_{ik}=1,a_{j,m(k)}=1} t_{ij}$, then $e_{ik} = e_{j,m(k)} = t_{ij}$. We apply a similar transformation as with the request matching problem to transform the problem into a maximum weighted matching problem.

3.6.2 Motivation for a greedy heuristic

After the first iteration, if in the solution each request is matched to the optimal pallet, the solution is optimal. If not, some of the requests may still be matched to the optimal pallets, but to other requests suboptimal pallets have been assigned if the optimal pallets have been used for other requests. Moreover, as a result of a suboptimal matching to a request, the matching of the pallet to the complementary request (the request that was matched to the first request) may be suboptimal as well, since the best response is defined by the ideal pallet for the first request. In other words, knowing that the first request is matched to a suboptimal (second choice) pallet, we may want to assign a different (conditionally optimal) pallet to the complementary request.

In order to make this possible, we suggest a simple greedy procedure in which we fix the assignment of pallets to requests if for a pair of requests the optimal pallets have been matched to both requests. Then we remove these pallets and requests from the problem, such that the best responses in the cost matrix E are updated. We solve the smaller problem and repeat this procedure. Typically, this converges to a situation where no unmatched requests remain, but it may occasionally happen that no pair of requests is matched to their ideal pair of pallets. This happens when two pairs of requests have the same ideal pair of pallets: in that case it may happen that one of the pallets is matched to a request of the first pair, while the other is matched to a request of the second pair. Although ideally some branching scheme would be used to determine which of the pairs gets assigned the pair of pallets, we continue greedily by applying a simple mechanism comparable to the first step: for each matched pallet i to request k we update the costs e_{ik} to the actual costs t_{ij} resulting from this match. Here j is the pallet that is matched to request $\ell = m(k)$.

We experimented with applying the simple feedback mechanism in general for the second subproblem. However, for this particular problem, this turned out not to be very effective. Using the simple feedback mechanism, the costs for the pallets that are mutually matched to their best responses will not be updated and therefore these matchings are implicitly fixed just as with the greedy strategy. However, for the other pallets, the costs get updated according to the cost of assignments that have been the (accidental) result of decisions based on false beliefs for the best responses. Therefore the updating mechanism is ineffective, while we can much more accurately update the best response matrix E by disregarding the matches that we know are fixed, which is why we implemented the greedy strategy.

3.7 Experiments

We solved a test set of instances using the following three methods:

- Solving the optimal MIP using lazy constraints
- Applying the two-step approach until convergence using the conditionally optimal MIP to solve the second subproblem
- Applying the two-step approach until convergence using the greedy repeated matching heuristic for the second subproblem

3.7.1 Dataset

The dataset consists of 10 randomly generated instances:

- $P = \{1, ..., 52\}$
- $R = \{1, ..., 500\}$
- C is a 500×500 matrix with $c_{ij} \begin{cases} \sim \text{Unif}(0,1) & \text{if } i < j \\ = c_{ji} & \text{if } i > j \end{cases}$
- A is a 500×52 matrix with $a_{ij} \sim \text{Bern}(0.02)$
- B is a 52×52 matrix with $b_{k\ell} \begin{cases} \sim \text{Bern}(0.7) & \text{if } k < \ell \\ = b_{\ell k} & \text{if } k > \ell \end{cases}$

In a typical trailer there are 2 columns with 13 stacks of 2 pallets, which is why we consider instances with $2 \cdot 13 \cdot 2 = 52$ pallets. Furthermore, 500 is a reasonable estimate for the size of the warehouse. We assume that the (normalized) 'costs' for retrieving any combination of two pallets is uniformly distributed between 0 and 1, that two arbitrary requests can stack with probability 0.7 and that a pallet can be used for each request with probability 0.02. As a result, on average $0.02 \cdot 500 = 10$ pallets can be used for each request. Furthermore we assure that A and B are symmetric, which means that with stacking (both requests and pallets), we do not need to keep track of the order of the pallets or requests in the stack.

3.7.2 Results

The results for the method using the optimal approach for the second phase with the 10 instances are in Table 1, and the results when using the heuristic for the second phase are in Table

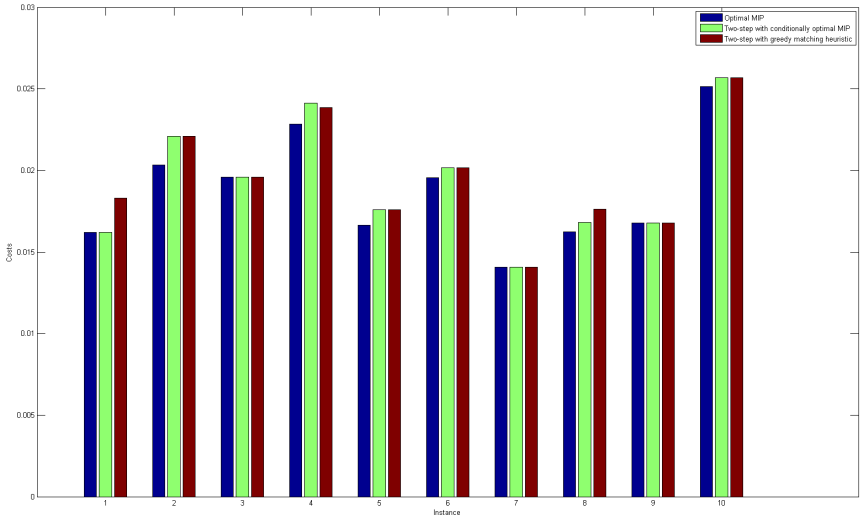


Figure 4: Results for 10 randomly generated instances

2. In Figure 4 the results using both approaches are visualized. Since lower costs are better, the two step approach obviously results in lesser quality solutions than the optimal approach. However, for the two step approach with optimal second step, the average 'gap' is only 2.88%. If additionally we use a heuristic for the second step, the gap is 4.55%, so this is only a minor additional loss. This is average behaviour because we observe that for our instances, in seven out of ten cases, using the heuristic for the second steps results in the same final solution. In one case using the heuristic actually improves the result while only for two cases the result is strictly worse.

To give some insight in how the mechanism works, Figure 5 plots $\tilde{c}_1(x, y)$ and $c(x, z)$ against the iterations using the optimal approach for the second phase for the 10 instances. From this we observe that typically convergence is achieved within 5 iterations. In most cases, the quality of the solution increases in subsequent iterations. In some cases, first a worse solution is found but after that the solution significantly improves and the mechanism terminates with a solution better than the initial solution. In one case $\tilde{c}_1(x, y) = c(x, z)$ from the start, which means that we could optimally match all pallets and the solution does not change at all and in one case the solution quality decreases first but finally the initial solution is found as the best.

Instance	Optimal	Initial solution		Final solution		Convergence	
	solution	Val	Gap	Val	Gap	It	Time
1	0.0162	0.0282	73.98 %	0.0162	0.14 %	3	0.36 s
2	0.0203	0.0448	120.52 %	0.0221	8.73 %	5	0.24 s
3	0.0196	0.0196	0.00 %	0.0196	0.00 %	1	0.22 s
4	0.0228	0.0544	138.01 %	0.0241	5.62 %	5	0.31 s
5	0.0167	0.0431	158.70 %	0.0176	5.67 %	4	0.41 s
6	0.0196	0.0405	107.16 %	0.0202	2.98 %	5	0.43 s
7	0.0141	0.0192	36.30 %	0.0141	0.00 %	2	0.38 s
8	0.0162	0.0355	118.93 %	0.0168	3.60 %	4	0.40 s
9	0.0168	0.0445	164.92 %	0.0168	0.00 %	5	0.22 s
10	0.0252	0.0257	2.11 %	0.0257	2.11 %	7	0.16 s
Average	0.0187	0.0355	92.06 %	0.0193	2.88 %	4.1	0.16 s

Table 1: Results for the 10 instances using the conditionally optimal MIP approach for the second phase

Instance	Optimal	Initial solution		Final solution		Convergence	
	solution	Val	Gap	Val	Gap	It	Time
1	0.0162	0.0459	183.24 %	0.0183	13.01 %	4	0.11 s
2	0.0203	0.0448	120.52 %	0.0221	8.73 %	5	0.13 s
3	0.0196	0.0196	0.00 %	0.0196	0.00 %	1	0.03 s
4	0.0228	0.0544	138.01 %	0.0239	4.46 %	5	0.09 s
5	0.0167	0.0431	158.70 %	0.0176	5.67 %	4	0.11 s
6	0.0196	0.0405	107.16 %	0.0202	2.98 %	5	0.08 s
7	0.0141	0.0192	36.30 %	0.0141	0.00 %	2	0.05 s
8	0.0162	0.0782	382.00 %	0.0176	8.58 %	5	0.12 s
9	0.0168	0.0445	164.92 %	0.0168	0.00 %	5	0.07 s
10	0.0252	0.0257	2.11 %	0.0257	2.11 %	7	0.06 s
Average	0.0187	0.0416	129.30 %	0.0196	4.55 %	4.3	0.06 s

Table 2: Results for the 10 instances using the heuristic approach for the second phase

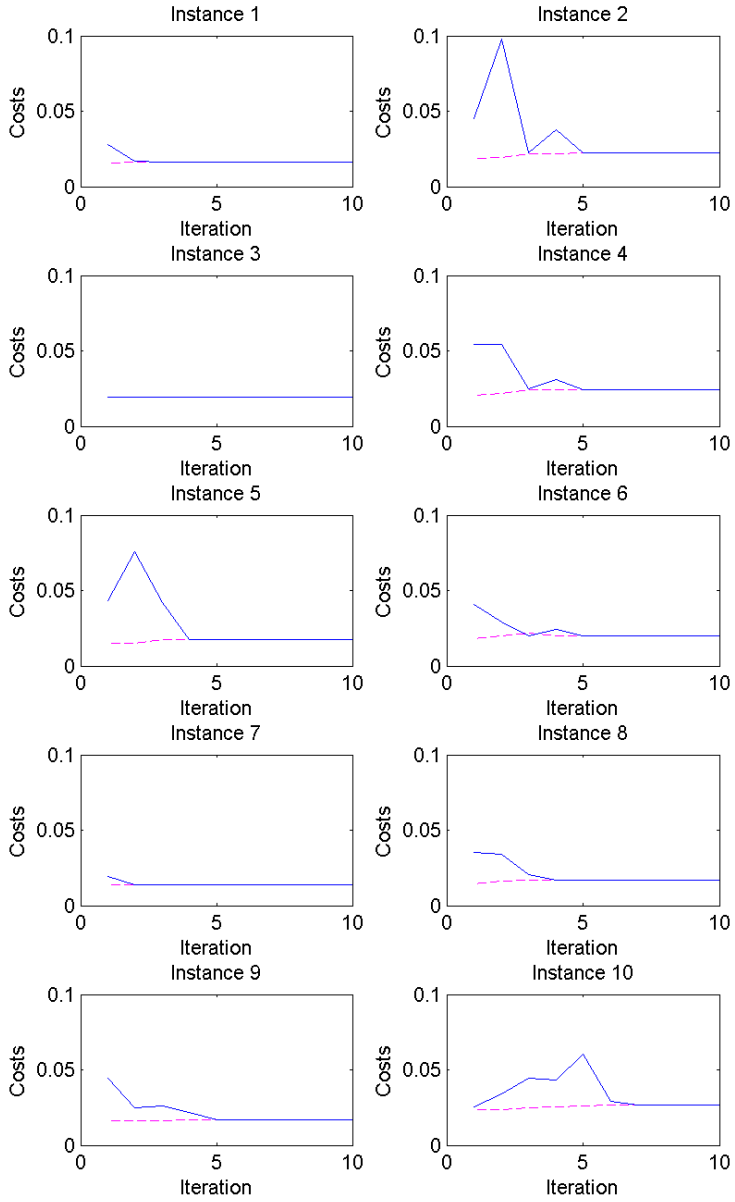


Figure 5: $\tilde{c}_1(x, y)$ (dashed purple) and $c(x, z)$ (blue) plotted against the iterations

4 The capacitated vehicle routing problem

4.1 The problem

The vehicle routing problem (Toth and Vigo, 2001) is one of the most intensively studied combinatorial optimization problems. It knows many variations that include special restrictions such as time windows and driving time limitations but we consider it only in its most basic form: the *capacitated vehicle routing problem* (CVRP) with a single depot. This problem is as follows: given a set of customers with a certain demand for each customer and a single depot, determine a set of routes starting and ending at the depot such that the total demand for the customers in each route does not exceed the capacity of the vehicle and such that the total route length, measured in driving time or distance, is minimized. We assume that the driving time or distance does not depend on the driving direction, so formally we consider the symmetric CVRP (SCVRP), although we will write it as CVRP.

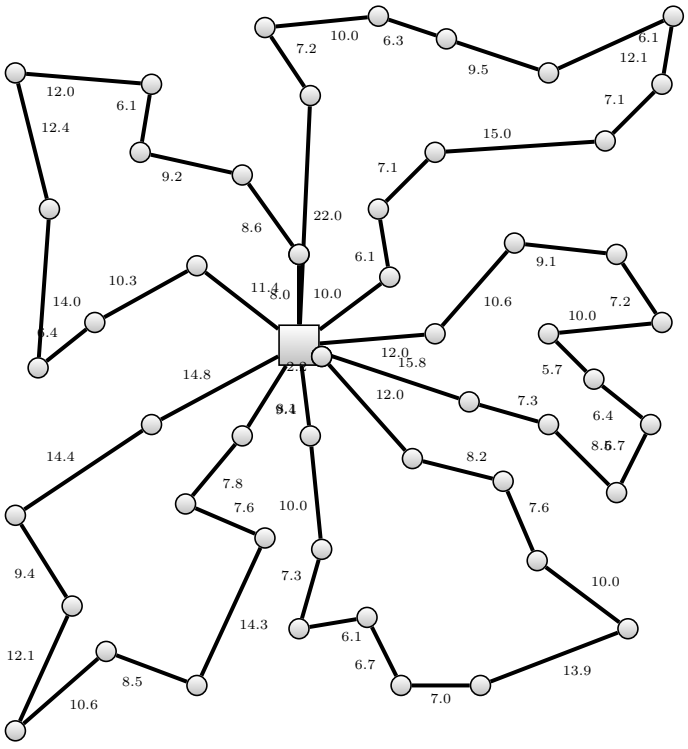


Figure 6: Visualization of the (capacitated) vehicle routing problem

4.2 Mathematical problem formulation

The CVRP can be expressed as a mathematical program using the following notation:

- Set of locations $V = \{0, ..., n\}$, with $0 \in V$ the depot and $V \setminus \{0\}$ the set of customers
- Symmetric 'cost matrix' T with $t_{ij} = t_{ji}$ the driving time or distance between locations i and j, $i, j \in V$
- Demand a_i for all customers $i \in V \setminus \{0\}$
- Vehicle capacity B

We will use terminology from graph theory with this problem: the set V is the set of nodes or vertices, while every route section between nodes i and j, $i \neq j$ is called an (undirected) edge.

Let the following decision variables be defined:

$$x_{ij} = \begin{cases} 1 & \text{if a vehicle traverses the edge from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases} \qquad i, j \in V, i < j$$

Now the CVRP can be expressed as a Mixed Integer Program (MIP) as follows:

$$\min \quad \sum_{i,j \in V : i<j} t_{ij} x_{ij} \tag{11}$$

$$\text{s.t.} \quad \sum_{h<i} x_{hi} + \sum_{j>i} x_{ij} = 2 \qquad i \in V \setminus \{0\} \tag{12}$$

$$\sum_{i \in S} \sum_{h<i, h \notin S} x_{hi} + \sum_{i \in S} \sum_{j>i, j \notin S} x_{ij} \geq 2r(S) \qquad S \subseteq V \setminus \{0\}, S \neq \emptyset \tag{13}$$

$$x_{ij} \in \{0, 1\} \qquad i, j \in V \setminus \{0\}, i < j \tag{14}$$

$$x_{0j} \in \{0, 1, 2\} \qquad j \in V \setminus \{0\} \tag{15}$$

The objective and constraints are explained below:

(11) The objective is to minimize the total distance or time that is driven by all of the vehicles.

(12) Each customer should have exactly two adjacent edges that are driven. As a result, the number of edges adjacent to the depot will be even so a Eulerian tour exists for the set of edges that is selected, with total distance corresponding to the objective.

(13) For each nonempty subset of the customers S, the number of vehicles that go from or to a location in S from or to a location in $V \setminus S$ should at least be two times the number of vehicles required to serve the customers S, in terms of capacity. This number $r(S)$ is the solution of the one-dimensional *bin packing problem* BPP (Martello and Toth, 1990) for items with size $a_i, i \in S$ and bin size B. The factor two is because each vehicle should drive to and from the set of customers. This set of constraints not only ensures that the capacity restrictions are met, but also ensures that the graph representing the solution is connected, since for every *cut* (partition of the graph in two sets of nodes), there are at least two edges selected such that one end of the edge is in the first set and the other end in the second set. Together with constraints (12) this ensures the existence of a single Eulerian tour that visits al nodes, although optionally the depot is visited multiple times.

(14) Define decision variables for edges between two customers. Each edge can be traversed at most one time.

(15) Define decision variables for edges between a customer and the depot. Each edge can be traversed at most two times, in order to allow for routes consisting of a single customer.

Apart from being the capacity constraints, the constraints (13) can be seen as generalizations of the subtour elimination constraints that occur with the *travelling salesman problem* (TSP) and the number of them is exponential in the size of the problem. Therefore, they should be added to the problem dynamically, just as we did for the matching problem and is done by Pataki (2003) for the TSP. However, exactly solving the problem using this MIP formulation is only possible for very small sized instances. Many other exact formulations have been developed, for instance using column generation (Barnhart et al., 1998), but we will not consider those here.

4.3 Decomposition of the problem

The CVRP is a generalization of the *travelling salesman problem* (TSP) (for an introduction see, for example, Lawler et al. (1985)), that has been mentioned already in the context the MIP formulation. The TSP consists of finding the minimum length route to visit a set of

customers using a single vehicle. Two classes of often used solution methods for the CVRP use the fact that the problem generalizes the TSP: the route-first-cluster-second and cluster-first-route-second methods. These methods decompose the CVRP problem in two subproblems: a clustering problem and a TSP, where the difference between the two classes is determined by the order in which the problems are solved. If first the clusters are made, then the second subproblem is a TSP which can be solved to create a route for each cluster separately. If first the routing is done, then a single TSP is solved with all the locations. In the second step, the solution is broken down into sections for single vehicles, which need to be transformed into feasible routes by connecting the beginning and end to the depot, respecting the capacity restrictions.

The decomposition that we will use as the basis for our simple feedback mechanism is the cluster-first-route-second method developed by Bramel and Simchi-Levi (1995). This method is inspired by the famous paper of Fisher and Jaikumar (1981), who propose to create clusters by solving a *generalized assignment problem* (GAP) using seed locations. Bramel and Simchi-Levi build on this approach but use the *the capacitated concentrator location problem* (CCLP) rather than the GAP for the clustering. Using this method, seed *sets* of locations can be used rather than single locations and the best seed sets are selected when the clusters are created. The reason that we use the cluster-first-route-second method developed by Bramel and Simchi-Levi is that it is a very clear decomposition of the problem in two phases. Although the original paper uses a subgradient method to solve the CCLP, we take advantage of increased computation power and the availability of solvers as Gurobi (Gurobi Optimization, 2014) to solve the problem directly using the MIP formulation. For reasonable sized clusters, the TSP in the second subproblem can be solved to optimality using the MIP formulation with dynamically added subtour elimination constraints (Pataki, 2003).

4.4 Filling in the ingredients

4.4.1 The intermediate solution y

The intermediate solution y is the set of clusters that has been formed by the CCLP. Although there are multiple ways to represent clusters, for instance using an adjacency matrix or simply by enumerating the clusters, we will use the structure resulting from the CCLP by listing the connections that are made. More formally, let $y_{ij} = 1$ if location i is connected to seed j, for $i, j \in V \setminus \{0\}$. $y_{ij} = 1$ implies that location j is a seed, so we do not need to explicitly keep track of the seeds that are selected. For consistency, we write $y_{jj} = 1$ if j is a seed and is therefore

'connected' to itself.

4.4.2 The cost metric c_1

The cost metric $c_1(x, y)$ gives for the solution y of the CCLP for instance x the cost. Although the CCLP distinguishes setups costs and connection costs, we will add the setup costs for a seed i to the connection costs of location i to seed i, which are always zero. Therefore, we let c_1 be represented by a matrix D with entries d_{ij} for the connection costs of location i to seed j for $i \neq j$ and d_{jj} the setup costs for seed j. Using this matrix, c_1 is defined as $c_1(x, y) = \sum_{i \in V \setminus \{0\}} \sum_{j \in V \setminus \{0\}} y_{ij} d_{ij}$.

4.4.3 The function F_1

$F_1(x, c_1)$ solves the CCLP using the cost matrix D. Although Bramel and Simchi-Levi present a formulation in which the seed sets can consist of multiple locations, we restrict ourselves to the case where each seed set only contains a single location, or in other words, each location is a seed. Therefore the locations and seeds are in the same domain, and we can use a modified version of the problem formulation that only requires one set of decision variables, which for convenience we will denote with y, corresponding to the solution that results.

$$y_{ij} = \begin{cases} 1 & \text{if location } i \text{ is connected to seed location } j \\ 0 & \text{otherwise} \end{cases} \qquad i, j \in V \setminus \{0\}, i \neq j$$

$$y_{jj} = \begin{cases} 1 & \text{if location } j \text{ is a seed location} \\ 0 & \text{otherwise} \end{cases} \qquad j \in V \setminus \{0\}$$

Now the CCLP can be formulated as follows:

$$\min \quad c_1(x, y) = \sum_{i \in V \setminus \{0\}} \sum_{j \in V \setminus \{0\}} y_{ij} d_{ij} \tag{16}$$

$$\text{s.t.} \quad \sum_{j \in V \setminus \{0\}} y_{ij} = 1 \qquad i \in V \setminus \{0\} \tag{17}$$

$$\sum_{i \in V \setminus \{0\}} a_i y_{ij} \leq B \qquad j \in V \setminus \{0\} \tag{18}$$

$$y_{ij} \leq y_{jj} \qquad i, j \in V \setminus \{0\}, i \neq j \tag{19}$$

$$y_{ij} \in \{0, 1\} \qquad i, j \in V \setminus \{0\} \tag{20}$$

The formulation is explained below:

(16) The objective is to minimize the total setup costs for the seed locations (the terms for which $i = j$) and the connection costs to the seed locations.

(17) Ensure that each location is connected to exactly one seed location (or is a seed location itself).

(18) Ensure that the demand connected to each seed location is less than the maximum capacity B.

(19) Ensure that location j is a seed location if location i is connected to location j.

(20) Define decision variables y.

Ideally we would solve the CCLP to optimality using Gurobi, within an accepted tolerance for Mixed Integer Programming, but we experienced that this is computationally far too expensive. We could implement the subgradient method that was proposed by Bramel and Simchi-Levi in the original paper, but this is not the purpose of this thesis. Since the result would be a heuristic, we choose instead to directly solve the MIP with a lower optimality tolerance, i.e. 5%, which means that in practice the Gurobi solver serves as a heuristic for the CCLP.

The result of $F_1(x, c_1)$ is y, which indicates a number of seeds and connections to those seeds, and therefore represents a number of clusters. Each cluster corresponds to a vehicle in the solution, and therefore for each cluster F_2 should create a route that visits the locations in the cluster and starts and ends at the depot.

4.4.4 The function F_2

$F_2(x, y)$ solves the symmetric TSP using the subtour elimination method. For each cluster in y, the TSP should be solved for the set of locations in the cluster and the depot. For the TSP, it is irrelevant that the depot is the actual start and end location. Let (for a certain clustering y) $K = \{j \in V \setminus \{0\} : y_{jj} = 1\}$ be the set of seeds corresponding to the clusters, and let $V_j, j \in K$ be the sets of locations for each cluster, including the depot, so $V_i \cap V_j = \{0\} \quad \forall i, j \in K, i \neq j$ and $\bigcup_{j \in K} V_j = V$. For clarity, we will now refer to each cluster, or route, with index $k \in K$.

For the symmetric TSP formulation for each cluster $k \in K$, we need a single set of decision variables, which we denote with z such that this corresponds to the overall solution:

$$z_{ij} = \begin{cases} 1 & \text{if route } k \text{ uses the edge from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases} \qquad i, j \in V_k, i < j$$

Now the TSP can be formulated as follows:

$$\min \quad \sum_{i,j \in V_k, i<j} t_{ij} z_{ij} \tag{21}$$

$$\text{s.t.} \quad \sum_{h<i} z_{hi} + \sum_{j>i} z_{ij} = 2 \qquad i \in V_k \tag{22}$$

$$\sum_{i,j \in S, i<j} z_{ij} \leq |S| - 1 \qquad S \subset V, S \neq \emptyset \tag{23}$$

$$z_{ij} \in \{0, 1\} \qquad i, j \in V, i < j \tag{24}$$

The formulation is explained below:

(21) The objective is to minimize the total distance or time of the route.

(22) Each location, so each customer and the depot, should have exactly two adjacent edges that are driven.

(23) For each subset of the locations, the number of edges connecting two locations in that subset should be lower than the number of locations. This ensures that no subtours can exist since a subtour contains the same number of edges as locations. Together with constraints (22) this ensures that a single Eulerian tour containing all locations is found.

(24) Define decision variables z.

Note that in the formulation clearly parts of the CVRP formulation in Section 4.2 can be recognized. However, there are a few differences: constraint (22) equals constraint (12) of the CVRP but now includes the depot since this is no longer implied by the capacity constraints (13) of the CVRP. These capacity constraints are no longer required, but the implied subtour elimination constraints are explicitly added. All variables are binary, since no edge needs to be traversed twice if the assumption is made that $|V_k| > 2$. This can be done because for $|V_k| = 2$ (and even for $|V_k| = 3$) the solution is trivial.

4.4.5 The a priori belief $\tilde{c}_1$

The a priori belief $\tilde{c}_1$ is represented by the matrix $\tilde{D}$. Unlike the pallet matching problem, we now cannot define an exact optimistic a priori belief by calculating an exact lower bound. This is because the connection costs for a location to a seed set represent the increase in length of the route if the location is inserted into the route. This clearly depends on the order in which the locations are inserted and on the other locations in the route. Therefore with this problem we

define a parameter $\gamma \in (0, 1]$ that describes the level of optimism we use. This parameter scales the insertion costs, such that a lower γ results in lower costs and thus a more optimistic $\tilde{c}_1$.

We do not scale the setup costs $\tilde{d}_{jj}$ for locations j, because we consider them as truth: if no other locations are connected then the length of the route only serving customer j clearly is $2t_{0j}$. We could use a similar argument to state that, given the fixed setup costs, the insertion costs $\tilde{d}_{ij}$ should always be equal to $t_{0i}+t_{ij}-t_{0j}$ since it should hold that $\tilde{d}_{jj}+\tilde{d}_{ij} = t_{0j}+t_{0i}+t_{ij}$, according to routes with two locations and the depot. However, this uniquely defines all insertion costs such that there is no possibility to update the belief matrix $\tilde{D}$. This shows that, because we restrict $\tilde{c}_1$ to being represented by the matrix $\tilde{D}$, it is not possible to let $\tilde{c}_1$ represent the true c_1 for all solutions. Fortunately, for the feedback mechanism it is only required that $\tilde{c}_1$ represents the actual c_1 for good quality solutions, and that the metric is worse for worse solutions (following from Proposition 1). Because of this, we do not restrict $\tilde{D}$, apart from $\tilde{d}_{jj} = 2t_{0j}$ and $\tilde{d}_{ij} \geq 0$. The last inequality simply states that a route can never be shorter by adding an additional location.

Although we do not restrict the matrix $\tilde{D}$ according to routes of two locations and the depot, we use the equalities for those routes to construct our a priori belief. If we also apply the 'optimism parameter' γ, the a priori belief matrix $\tilde{D}$ is defined as follows:

$$\tilde{d}_{ij} = \gamma(t_{0i} + t_{ij} - t_{0j}) \qquad \forall i, j \in V \setminus \{0\}, i \neq j, \tag{25}$$

$$\tilde{d}_{jj} = 2t_{0j} \qquad \forall j \in V \setminus \{0\}. \tag{26}$$

4.4.6 The updating mechanism of $\tilde{c}_1$

It is very unlikely that after the first iteration $c(x, z) = \tilde{c}_1(x, y)$, because $\tilde{c}_1$ is not an exact bound. If γ is chosen suitably optimistic, then it will hold that $c(x, z) > \tilde{c}_1(x, y)$, and we need to adjust the connection costs for the clusters such that $c(x, z) = \tilde{c}_1(x, y)$. Note that the costs $c(x, z)$ decompose into a cost for each route k: if $TSP(V_k)$ is the TSP solution for route $k \in K$, then $c(x, z) = \sum_{k \in K} TSP(V_k)$. Refer that the index k of each route indicates the seed location, (so $y_{kk} = 1$ for $k \in K$), so we can write the believed costs for the route as $2\tilde{d}_{kk} + \sum_{i \in V_k \setminus \{0,k\}} \tilde{d}_{ik}$. Again, we consider $\tilde{d}_{kk}$ as truth and therefore only update $\tilde{d}_{ik}$ for $i \neq k$. This implies that after the update the following should hold:

$$\sum_{i \in V_k \setminus \{0,k\}} \tilde{d}_{ik} = TSP(V_k) - 2\tilde{d}_{kk} \quad \forall k \in K. \tag{27}$$

Although there are multiple ways in which this can be done, we choose to scale the entries

from the belief matrix $\tilde{D}$ (in the previous iteration) such that the equation holds. In other words, we update $\tilde{D}$ using the following formula:

$$\tilde{d}_{ik} = \frac{\tilde{d}_{ik}}{\sum_{i \in V_k \setminus \{0,k\}} \tilde{d}_{ik}} (TSP(V_k) - 2\tilde{d}_{kk}) \quad \forall k \in K. \tag{28}$$

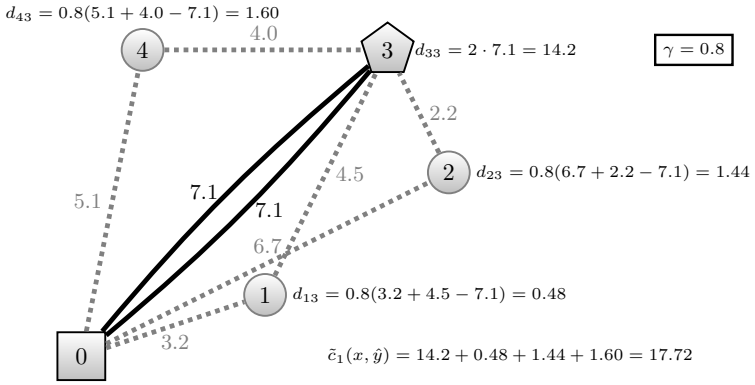
Figure 7 shows an example how the updating mechanism works. The first subfigure calculates $\tilde{c}_1(x, \hat{y})$ before the update, the second subfigure illustrates the optimal TSP tour $\hat{z}$ and the length $c(x, \hat{z})$, and the third figure illustrates how the insertion costs are scaled such that $\tilde{c}_1(x, \hat{y}) = c(x, \hat{z})$.

We have experimented with alternatives, such as updating $\tilde{D}$ by scaling the initial $\tilde{D}$ rather than the last $\tilde{D}$, or scaling the observed leave-one-out insertion costs for an iteration, and updating by using convex combinations of the updated matrices $\tilde{D}^{(1)}, \tilde{D}^{(2)}$ and $\tilde{D}^{(3)}$ according to the three options.

Definition 4. *A convex combination of matrixes $\tilde{D}^{(1)}, ..., \tilde{D}^{(n)}$ is a matrix $\tilde{D}$, such that for each entry $\tilde{d}_{ik}$ it holds that $\tilde{d}_{ik} = \sum_{j=1}^{n} \alpha_j \tilde{d}_{ik}^{(j)}$, where $\alpha_j \geq 0$ and $\sum_{j=1}^{n} \alpha_j = 1$.*

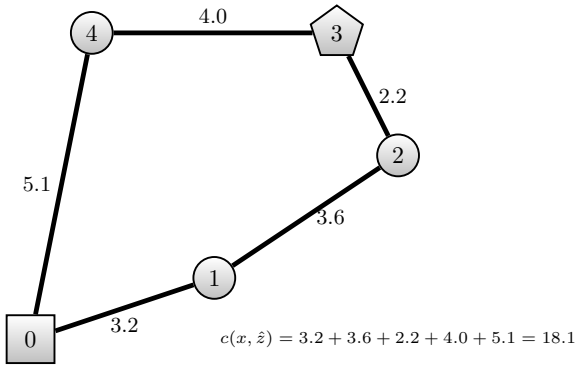
Unfortunately, using the convex combination ruins the natural convergence properties of the method since $\tilde{c}_1$ could be updated even if $c(x, z) = \tilde{c}_1(x, y)$. Of the individual options, scaling the last $\tilde{D}$ performed best. For any of the methods, with the advanced update mechanism described next, it may happen that a connection cost becomes zero (if a location lies exactly in between two other locations), which should be appropriately accounted for.

1.



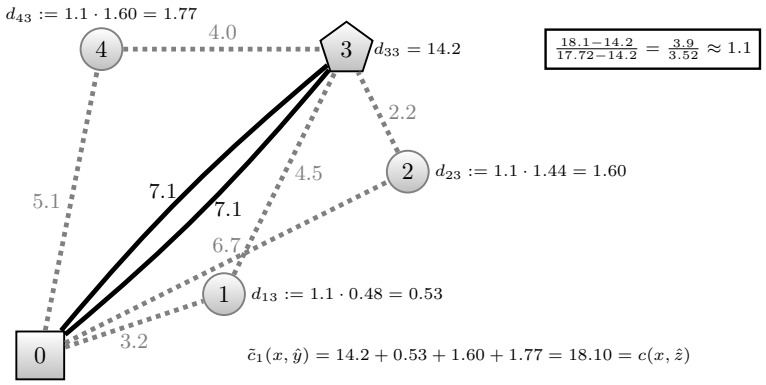
Calculation of $\tilde{c}_1(x, \hat{y})$

2.



Calculation of $c(x, \hat{z})$

3.



Updating of $\tilde{c}_1(x, \hat{y})$ such that $\tilde{c}_1(x, \hat{y}) = c(x, \hat{z})$

Figure 7: Illustration of the updating mechanism for the CVRP

Let $\tilde{c}_1$ always represent the incumbent Experiments showed that updating $\tilde{c}_1$ such that $c(x,z) = \tilde{c}_1(x,y)$ typically results in a diverging algorithm. This is because each update of $\tilde{c}_1$ may ruin what has been learned in previous iterations. To some extent, this is unavoidable, since it is not possible to let $\tilde{c}_1$ represent al previous iterations. For bad quality solutions, this is not much of a problem, but ideally we would want $\tilde{c}_1$ to remain accurate for good solutions. In order to achieve this, we use the fact that we have some freedom in updating the weights to include an additional restriction: we restrict the update of $\tilde{D}$ to be such that not only $c(x,z) = \tilde{c}_1(x,y)$ for the last solution observed, but also for the best solution so far, which we call the *incumbent*.

More formally, if we let z^* and y^* represent the incumbent and $\hat{z}$ and $\hat{y}$ the last solution, then we require that both $c(x,\hat{z}) = \tilde{c}_1(x,\hat{y})$ and $c(x,z^*) = \tilde{c}_1(x,y^*)$. This means that Equation (27) should hold for both the incumbent and the last solution. We will show using an example how we can update $\tilde{D}$ such that this holds. If the new solution is better than the incumbent, we simply update the matrix and make the new solution the incumbent. Since the incumbent is the last solution, the equations hold trivially. Otherwise, we assume that $\tilde{D}$ before the update has the property that it satisfies Equation (27) for the incumbent. If we update $\tilde{D}$ such that the equation still holds, then by induction it follows that $\tilde{D}$ has this property for all iterations. Now if we let K^* be the set of seed locations for the incumbent, and K for the last solution, then only for $k \in K^* \cap K$ the update affects Equation (27) for the incumbent. In other words, for $k \in K \setminus K^*$ we can normally apply the update, for $k \in K^* \setminus K$ nothing is updated but for $k \in K^* \cap K$ we need to update $\tilde{D}$ such that Equation (27) holds for the two solutions.

Let $k \in K^* \cap K$ be a seed location for both the incumbent and the last solution. Let V_k^* and V_k correspond to the clusters for the incumbent and the last solution respectively. Then if $V_k^* = V_k$, Equation (27) trivially holds, regardless of how we update $\tilde{D}$. Therefore, assume $V_k^* \neq V_k$ and define $S_0 = V_k^* \cap V_k$, $S_1 = V_k^* \setminus V_k$ and $S_2 = V_k \setminus V_k^*$. For an illustration, see Figure 8. Now we update the connection costs $\tilde{d}_{ik}$ for $i \in S_0 \setminus \{0,k\}$ such that Equation (27) holds with V_k replaced by S_0, the intersection of the location sets of both routes. Next we update the connection costs $\tilde{d}_{ik}$ for $i \in S_1$ such that they sum up to the increase in length if the locations in S_1 are added to the TSP solution for S_0, and we do similarly for S_2. More formally, we require the following for $\tilde{D}$ based on S_0:

$$\sum_{i \in S_0 \setminus \{0,k\}} \tilde{d}_{ik} = TSP(S_0) - 2\tilde{d}_{kk}. \tag{29}$$

We can fill this in in Equation (27) for V_k^* and V_k by observing that $\{S_0, S_1\}$ and $\{S_0, S_2\}$ are

partitions of V_k^* and V_k respectively.

$$\sum_{i \in V_k \setminus \{0,k\}} \tilde{d}_{ik} = TSP(V_k) - 2\tilde{d}_{kk} \tag{30}$$

$$\sum_{i \in S_2} \tilde{d}_{ik} + \sum_{i \in S_0 \setminus \{0,k\}} \tilde{d}_{ik} = TSP(V_k) - 2\tilde{d}_{kk} \tag{31}$$

$$\sum_{i \in S_2} \tilde{d}_{ik} + TSP(S_0) - 2\tilde{d}_{kk} = TSP(V_k) - 2\tilde{d}_{kk} \tag{32}$$

$$\sum_{i \in S_2} \tilde{d}_{ik} = TSP(V_k) - TSP(S_0) \tag{33}$$

$$\tag{34}$$

For V_k^* it holds that

$$\sum_{i \in S_1} \tilde{d}_{ik} = TSP(V_k^*) - TSP(S_0). \tag{35}$$

Again, we apply the scaling according to $\tilde{D}$, so the updating formula is (for $k \in K^* \cap K$):

$$\tilde{d}_{ik} = \frac{\tilde{d}_{ik}}{\sum_{i \in S_0 \setminus \{0,k\}} \tilde{d}_{ik}} (TSP(S_0) - 2\tilde{d}_{kk}) \qquad i \in S_0 \setminus \{0,k\}, \tag{36}$$

$$\tilde{d}_{ik} = \frac{\tilde{d}_{ik}}{\sum_{i \in S_1} \tilde{d}_{ik}} (TSP(V_k) - TSP(S_0)) \qquad i \in S_1, \tag{37}$$

$$\tilde{d}_{ik} = \frac{\tilde{d}_{ik}}{\sum_{i \in S_2} \tilde{d}_{ik}} (TSP(V_k^*) - TSP(S_0)) \qquad i \in S_2. \tag{38}$$

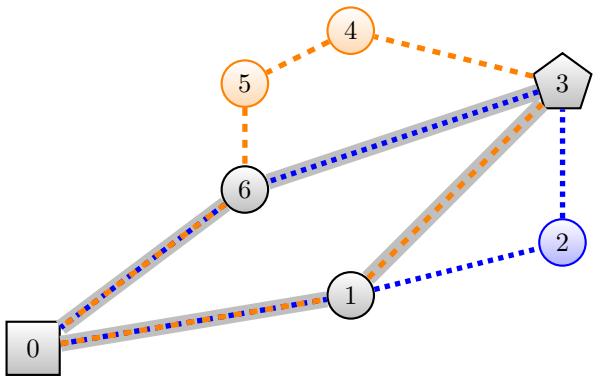


Figure 8: Illustration of the intersection between the route in the incumbent solution (orange) and the route in the last solution (blue). The seed $k = 3$, 0 is the depot, $S_0 = \{0, 1, 3, 6\}$, $S_1 = \{4, 5\}$ and $S_2 = \{2\}$. The thick gray line is the TSP solution for S_0.

Connection costs of 0 As was mentioned, we should appropriately account for the possibility that a connection cost becomes 0. If $\tilde{D}$ is updated by scaling the last $\tilde{D}$, then once the connection cost has become 0 it will remain 0. However, if all connection costs to update are 0, then the denominator becomes zero and as a result the update is not well defined. Therefore, if that is the case, we divide the insertion costs of the locations evenly over the locations. Typically this means that a single location with insertion costs 0 gets updated to the true insertion costs according to the last route. More formally, the updating formula becomes the following (for $k \in K^* \cap K$):

$$\tilde{d}_{ik} = \begin{cases} \frac{\tilde{d}_{ik}}{\sum_{i \in S_0 \setminus \{0,k\}} \tilde{d}_{ik}} (TSP(S_0) - 2\tilde{d}_{kk}) & \text{if } \sum_{i \in S_0 \setminus \{0,k\}} \tilde{d}_{ik} > 0 \\ \frac{1}{|S_0|-2} (TSP(S_0) - 2\tilde{d}_{kk}) & \text{otherwise} \end{cases} \qquad i \in S_0 \setminus \{0,k\}, \tag{39}$$

$$\tilde{d}_{ik} = \begin{cases} \frac{\tilde{d}_{ik}}{\sum_{i \in S_1} \tilde{d}_{ik}} (TSP(V_k) - TSP(S_0)) & \text{if } \sum_{i \in S_1} \tilde{d}_{ik} > 0 \\ \frac{1}{|S_1|} (TSP(V_k) - TSP(S_0)) & \text{otherwise} \end{cases} \qquad i \in S_1, \tag{40}$$

$$\tilde{d}_{ik} = \begin{cases} \frac{\tilde{d}_{ik}}{\sum_{i \in S_2} \tilde{d}_{ik}} (TSP(V_k^*) - TSP(S_0)) & \text{if } \sum_{i \in S_2} \tilde{d}_{ik} > 0 \\ \frac{1}{|S_2|} (TSP(V_k^*) - TSP(S_0)) & \text{otherwise} \end{cases} \qquad i \in S_2. \tag{41}$$

$$\tag{42}$$

Prevent cycling Although typically the updating mechanism adjusts the 'optimistic' a priori belief by increasing it, after a few iterations it may happen for some individual values that they decrease because according to the updating formulae the 'distribution' of the costs over the sets S_0, S_1 and S_2 is different. As a result, the algorithm may cycle rather than converge to a single $\tilde{c}_1$. This may be prevented using a simple cycle detection mechanism which terminates the algorithm if this happens. There are other options, such as updating the belief matrix $\tilde{D}$ such that not only the incumbent and the last solution are represented, but also the solution that starts the cycle. Using Venn diagrams (Venn, 1880) it can be shown that the updating mechanism can be extended such that the matrix $\tilde{D}$ represents more than two solutions, but this may cause problems if there are empty intersection sets. We consider the implementation of the different options without the scope of this thesis, and solve the cycling problem by limiting the number of iterations to 100.

4.5 Simple feedback mechanism for the CVRP

Filling in the steps from Section 4.4 in the algorithm in Section 2.5 results in Algorithm 3 for the pallet matching problem.

Algorithm 3 Simple feedback mechanism.

procedure SIMPLEFEEDBACKCVRP(x, γ)
  $D \leftarrow$ calculateSetupAndConnectionCostsMatrix(x, γ)
  while true **do**
    $\hat{y} \leftarrow$ createClustersUsingCCLP(x, D)
    $\hat{z} \leftarrow$ solveTSPs($x, \hat{y}$)
    if $\tilde{c}_1(x, \hat{z}, D) == c(x, \hat{z})$ **then return**
    end if
    $D \leftarrow$ Update($D, \hat{y}, \hat{z}, c$) ▷ Update $\tilde{c}_1(x, \hat{y}) := c(x, \hat{z})$
  end while
end procedure

4.6 Variant: a heuristic approach for the second subproblem

Just as with the pallet matching problem we can use a heuristic approach to solve the second subproblem. Although there are many heuristic approaches for the TSP, we use the algorithm of Christofides (1976) because it is a very simple and elegant method that is mostly deterministic. Christofides algorithm consists of five steps:

1. Create the *minimum spanning tree* MST of the graph.
2. Find a minimum cost perfect matching between the odd degree nodes of the graph.
3. Add edges corresponding to the matching to the minimum spanning tree to create a multigraph[1] with no odd degree nodes.
4. Find a Eulerian tour in the resulting multigraph.
5. Create shortcuts in the graph to skip nodes that are visited twice.

The length of the tour that is found this way is guaranteed to be within $\frac{3}{2}$ times the optimal tour length. This is a theoretical worst case scenario, and we found that the tour is on average 6% longer than optimal. For creating the MST we use the famous algorithm of Prim (1957) since

[1] A graph in which two nodes can be connected by more than one edge.

this is more efficient than Kruskal (1956) according to Dijkstra (1959), who also presented Prim's algorithm in the paper in which he develops his famous shortest path algorithm. To solve the minimum cost perfect matching problem, we again use the LEMON (2014) library. Finding the Eulerian tour is trivial since any subtour that is not included can be inserted at any node. The shortcutting simply means that we remove duplicate nodes in the tour after the first occurrence. Most steps in the algorithm are deterministic, but minor differences in the solution may be found depending on how the Eulerian tour is constructed and which node is taken as the starting node for shortcutting. We always create the tour in the same way and start the shortcutting at the depot, so the algorithm is deterministic and we accept that it might be slightly suboptimal as a consequence.

Using a heuristic for the TSP, it may occur that a tour found for an intersection set (of the incumbent and last solution) is actually longer than the tour for the larger set of locations in the incumbent or last solution. If in that case we apply the updating formulas, we get a negative insertion costs, which is incorrect since the true optimal tour of a smaller set of locations is always smaller. Therefore, for the intersection set corresponding to a seed location, we do not only calculate a tour using the heuristic, but also by removing the locations not in the intersection from the tour corresponding to that seed in the incumbent and the last tour. There is one important thing to notice. If the set of locations in the tour in the incumbent is a *subset* of the set of locations in the last solution, the intersection set is equal to the set of locations for the incumbent. If in that case we find a better solution for the intersection set, we need to also update the corresponding route for the incumbent, in order to assert that all equations remain valid. The same holds if the set of locations in the tour in the last solution is a subset of the set of locations in the incumbent.

4.7 Experiments

In order to test the feedback mechanism, we use the benchmark instances from Christofides and Eilon (1969). We perform a full factorial design for a number of parameter settings, which are described in Table 4.

Setting	Values
TSP method	'Optimal', 'Christofides'
Updating mechanism according to	'Last', 'Initial'
Optimality tolerance CCLP	5%, 20%
Optimism parameter γ	0.6, 0.8

Table 4: Full factorial parameter design for the simple feedback mechanism for the CVRP

4.7.1 Results

Table 5 gives the results for the 100 customer instance from Christofides and Eilon (1969). The full results tables for the other instances are given in Appendix A. It can be seen that for one configuration, the algorithm cycles and terminates after the maximum of 100 iterations. In general we observed that the following configuration was the best performing: using the optimal approach for the second phase, updating the cost matrix from the previous iteration, solving the first phase up to a MIPgap of 5 % and using the optimism parameter $\gamma = 0.80$. This configuration gives the best tradeoff between execution time and quality of results. The results for this configuration for all six instances are given in Table 6. We see that there is no improvement for the smallest two instances, which indicates that the mechanism requires some scale in order to have enough 'flexibility' to correctly update $\tilde{c}_1$. For the other instances, the improvement may seem small in absolute terms. However, since one can do no better than the optimum, the costs for the optimal routes are unavoidable and we should only consider the gap with the optimum. This is what can be decreased using a better solution, and indeed we see that if we compare the gaps with the optimal (or best known) values between the first and last iterations, we see that we are able to reduce the 'inefficiency' by a factor 2 to 5.

Figure 9 plots $\tilde{c}_1(x, y)$ and $c(x, z)$ against the iterations for the 6 instances. Just as with the pallet matching problem, we observe that in most cases the solution improves within the first few iterations. In the plots we can see that $\tilde{c}_1(x, y)$ and $c(x, z)$ indeed approach each other.

Instance vrp3-100

Parameters				Initial solution		Incumbent		Convergence	
Method	Update	Tol.	γ	Val	Gap	Val	Gap	It	Time
'Optimal'	'Initial'	20.0 %	0.60	868.91	5.18 %	849.95	2.88 %	9	27.4 s
			0.80	887.28	7.40 %	887.28	7.40 %	3	3.5 s
		5.0 %	0.60	864.44	4.64 %	839.61	1.63 %	44	1243.4 s
			0.80	858.54	3.92 %	845.31	2.32 %	100	1984.1 s
	'Last'	20.0 %	0.60	868.91	5.18 %	850.82	2.99 %	5	19.4 s
			0.80	887.28	7.40 %	887.28	7.40 %	3	3.5 s
		5.0 %	0.60	864.44	4.64 %	848.92	2.76 %	26	753.9 s
			0.80	858.54	3.92 %	832.43	**0.76** %	8	169.7 s
'Christof.'	'Initial'	20.0 %	0.60	933.55	13.00 %	911.99	10.39 %	9	18.5 s
			0.80	967.06	17.06 %	937.57	13.49 %	5	12.9 s
		5.0 %	0.60	947.39	14.68 %	876.77	6.13 %	29	895.7 s
			0.80	930.86	12.68 %	876.51	6.10 %	24	433.2 s
	'Last'	20.0 %	0.60	933.55	13.00 %	911.99	10.39 %	9	17.1 s
			0.80	967.06	17.06 %	961.44	16.38 %	5	6.7 s
		5.0 %	0.60	947.39	14.68 %	883.02	6.89 %	18	558.7 s
			0.80	930.86	12.68 %	891.45	7.91 %	11	234.1 s

Table 5: Results for instance with 100 customers

Instance	Initial solution		Incumbent		Convergence	
	Val	Gap	Val	Gap	It	Time
vrp1-50	539.22	2.78 %	539.22	2.78 %	2	0.4 s
vrp2-75	841.93	0.80 %	841.93	0.80 %	7	137.5 s
vrp3-100	858.54	3.92 %	832.43	0.76 %	8	169.7 s
vrp4-100	832.83	1.62 %	826.90	0.90 %	4	69.2 s
vrp5-120	1052.54	1.00 %	1045.42	0.32 %	7	428.5 s
vrp6-150	1076.66	4.69 %	1053.24	2.41 %	10	878.5 s

Table 6: Results for different instances with parameter settings ('Optimal', 'Last', 5.0 %, 0.80)

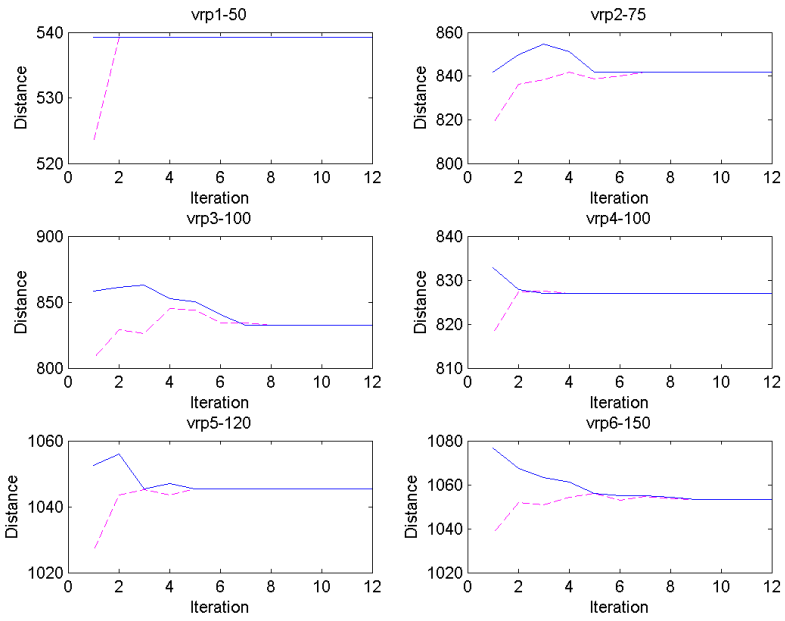


Figure 9: $\tilde{c}_1(x, y)$ (dashed purple) and $c(x, z)$ (blue) plotted against the iterations

5 The resource assignment problem

5.1 The problem

The resource assignment problem (RAP) (Visser, 2014) is a variant of the multiple trip vehicle routing problem (MTVRP) with a single depot, time window constraints and driving time constraints. Referring to the vehicles as trailers, the problem is to create routes for trailers for a fixed time horizon, and assign drivers (resources) to the *sections* (trips) in the trailer routes. Sections occur if the trailer needs to visit the depot multiple times, for instance because the total demand of the customers in the route exceeds the trailer capacity. It may be optimal, or even required, to use different drivers for a single route, for example when the total allowed driving time is limited or more efficiency is achieved when a driver skips loading time by continuing with another trailer. Apart from the total driving time, we consider regulations concerning breaks after a certain driving time. This means that in a route section a break should be planned if the cumulative driving time for the driver would otherwise exceed a certain limit. The primary goal is to minimize the number of drivers. Secondly, the number of trailers or other measures, such as the total driving distance, may be considered.

The RAP is a complex problem with many practical constraints. It is significantly more complicated than the CVRP, which only takes into account capacity constraints. The RAP has additional constraints that limit the possibilities for total route lengths for drivers, and enforce breaks to be planned, while at the same time each customer can only be visited within a certain time window. With the CVRP only routes for vehicles need to be created, but the RAP requires routes to be created for both trailers and drivers, with different restrictions.

5.2 Decomposition of the problem

We use the method developed by Visser (2014) to solve the RAP in two phases. In the first phase, we use a parallel cheapest insertion heuristic to create routes for the trailer. In order to be able to implement the feedback mechanism, we introduce an additional parameter which can be used to influence the insertion costs depending on the time. In the second phase, we assign drivers to the trailer sections and plan breaks in these sections such that a feasible driver assignment is found that minimizes the number of required drivers. This is done using an exact column generation method. Figure 10 is an illustration of a solution, although the routes are not shown on a map but on a Gantt chart. Each row corresponds to a driver and consists of

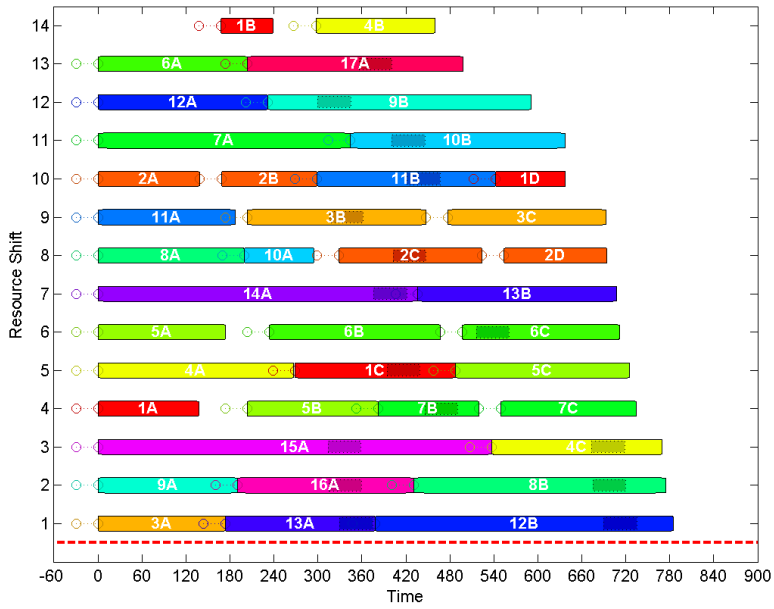


Figure 10: Resource assignments

multiple sections, which are colored to represent the trailers. The dark areas outlined by dashes represent the breaks for the drivers.

5.3 Filling in the ingredients

5.3.1 The relation between the two phases

In contrast with the pallet matching problem and the CVRP, with the RAP there is no direct relation in terms of costs between the first phase and the second phase because the unit of the measure that is optimized is different. To see this, note that both phases for the pallet matching problem optimize the costs/retrieval times and both phases for the CVRP optimize the total distance. With the RAP, phase one optimizes the trailer routes with respect to the number of trailers and the distance. This does not take the number of needed drivers into account, which is in fact the global objective. The problem is that there is no clear relation between the trailer routes and the number of drivers needed. We will try to introduce this relation using the notion of a 'topline', which is directly related to the number of drivers and can be observed from the solution of the second phase. Next we introduce a way to estimate the topline from the solution

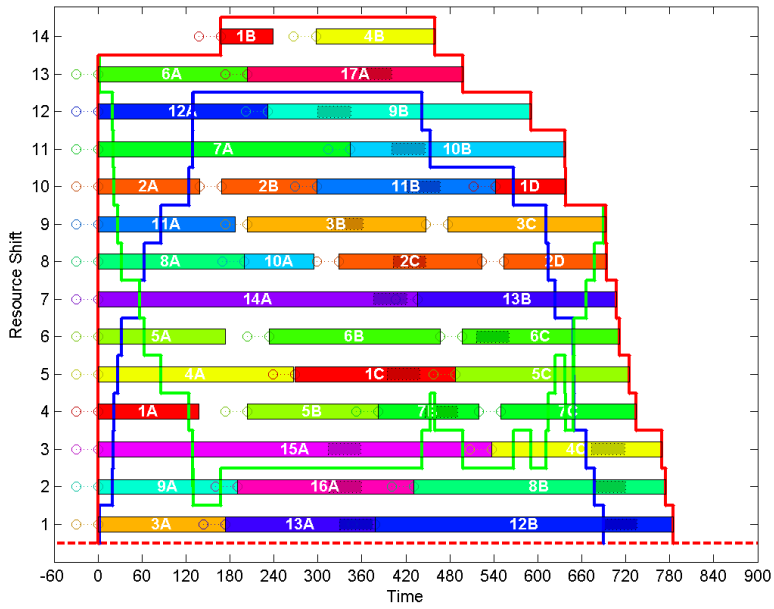


Figure 11: Illustration of the topline (red), the lower bound (blue) and the bias (green)

of the first phase, and to calculate an estimate for the number of drivers needed from it.

The topline The topline is the line that encloses the intersection (per row) of the two diagrams representing the solution if all driver shifts are sorted by either their start time or their completion time. Typically, the difference between the two sortings will be minor, such that the topline almost encloses both diagrams. For the example in Figure 10, there is no difference between the sortings and the topline is as illustrated by a red line in Figure 11. If the start time of the latest starting driver shift is before the completion time of the earliest completing driver shift, then the shifts will intersect for all rows and the height of the topline will equal the number of drivers. This is illustrated in Figure 12. Very rarely this is not the case, such that there are two disjunct driver shifts, which apparently cannot be combined because of constraints. Therefore, we consider the time between the completion time of the first shift and the start time of the last shift as blocked and add it to the topline. From this definition it follows that the maximum of the topline is equal to the number of drivers needed.

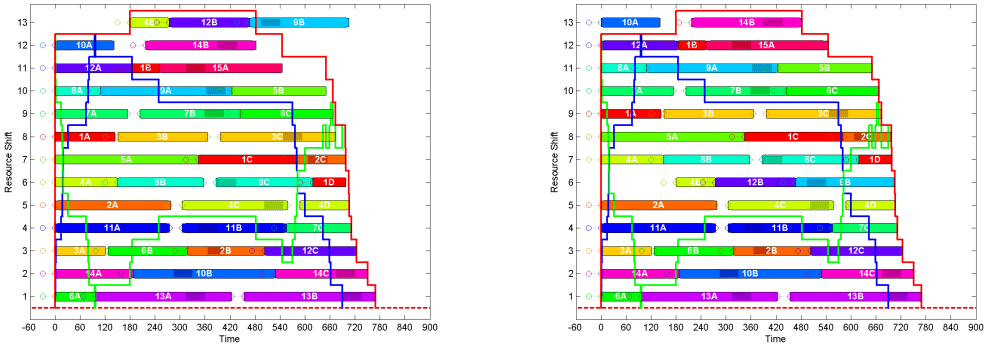


Figure 12: Illustration of the topline (red) with different sortings for the start and completion time

Estimating the number of drivers The height of the topline during a time interval is a measure for how busy the schedule is during that time interval, where gaps between shifts that cannot be used are also counted as busy time. To estimate the number of drivers from the trailer routes using the topline, we need to define two things: how to estimate the topline from the trailer routes and how to estimate the number of drivers from the *estimated* topline. Although we cannot observe the topline from the trailer routes, we are able to construct a *lower bound* for the number of drivers by accumulating the time intervals that cannot be avoided by shifting the trailer route sections in time. The difference between the topline and the lower bound line is a result that depends on how the sections are shifted and where there are gaps, and we denote this difference as the *bias*. Note that the lowerbound line can be calculated both for the second phase and the first phase. This means that we can observe the bias in iteration k from the solution of the second phase using

$$\text{bias}_k = \text{topline}_k - \text{lowerBound}_k. \tag{43}$$

To estimate the topline for the next iteration from the first phase we can update the indices for the topline and the lower bound and rewrite the formula as

$$\text{topline}_{k+1} = \text{lowerBound}_{k+1} + \text{bias}_k. \tag{44}$$

These equations are visualized in Figures 11 and 12: the red line is the topline, the blue line is the lower bound and the green line is the bias.

To estimate the number of drivers from the topline, it is very intuitive to take the maximum value. However, typically the estimated topline from Equation (44) is not very smooth because

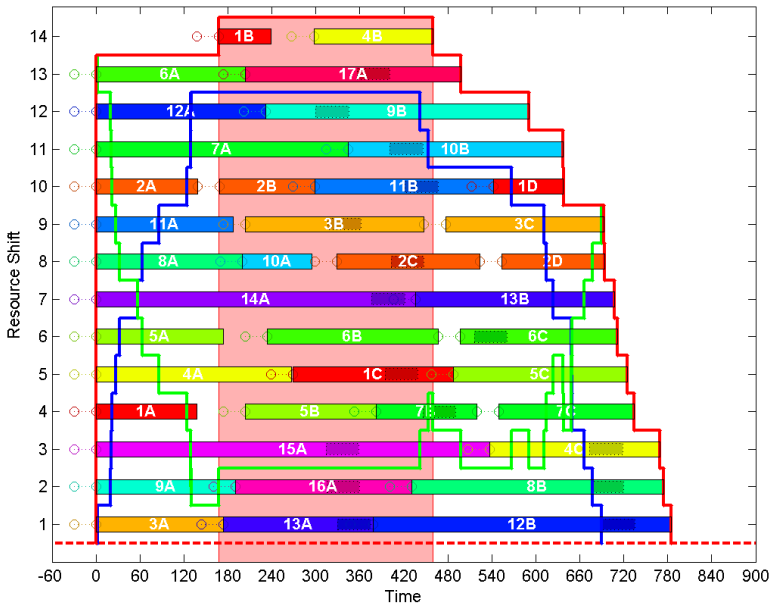


Figure 13: Illustration of the topline with the critical interval

the intervals in the lower bound line and the bias line do not correspond. Therefore, we introduce a more robust estimator: the average height of the topline in the critical interval. Just as the bias, we observe the critical interval from the solution of the second phase: we define it as the interval in which the topline is the highest. By definition, the average height of the topline in this interval is equal to the maximum and therefore to the number of drivers. This is illustrated in Figure 13.

In Section 5.3.7 we will present an adaption to the Equations (43) and (44) such that it also takes into account the bias and the critical interval of the incumbent solution. In Section 5.4 we will visualize the updating mechanism.

5.3.2 The intermediate solution y

The intermediate solution y is the set of trailer routes. We will not go into details about the mathematical representation of y. Instead, we explain and visualize the intermediate solution (corresponding to the solution in Figure 10) in Figure 14. The dark colored bars represent the trailer route sections as they are planned, while the corresponding lighter bars represent the earliest start time and latest completion time of these sections. These are the intervals in which

the sections can be shifted without violating the time window constraints. As with the final solution, the small dark areas outlined by dashes indicate the breaks. These are the breaks that would be required if each trailer is driven by a single driver, but these may be changed in the second phase depending on the driver assignment.

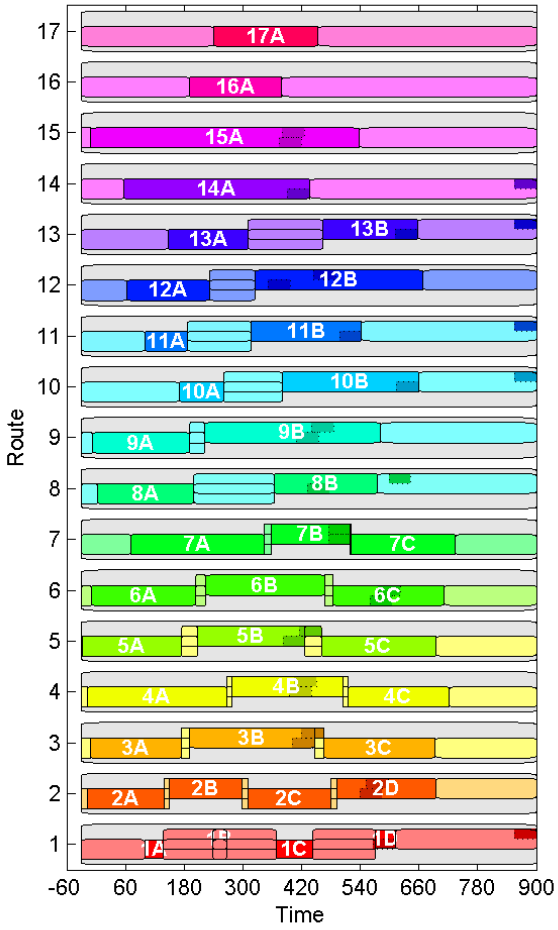


Figure 14: y: trailer routes with sections

5.3.3 The cost metric c_1

The cost metric c_1 is the estimated number of drivers, which follows from the lower bound line and the bias line as we explained.

5.3.4 The function F_1

The function F_1 creates the trailer routes by taking into account the objective c_1. This means that we modify the parallel cheapest insertion heuristic such that it minimizes the *estimated topline*. Although formally the topline is evaluated only at the critical interval, we try not to sacrifice other intervals by minimizing the topline in general. To do so we keep track of the topline at any stage during the execution of the heuristic. This means that we start with an empty lower bound, such that initially the topline is equal to the bias. Then, if we insert a stop to the route, we update the lower bound and as a result the topline. With each next insertion, we add a cost such that insertion is less attractive in intervals at which the topline has a high value. This results in a topline that is more spread over the time horizon. We repeat the procedure five times with different weights for the topline insertion costs compared to the default insertion costs such that we find five different solutions. The output of F_1 is defined as the one that minimizes $\tilde{c_1}$, the average height of the estimated topline in the critical interval.

5.3.5 The function F_2

The function F_2 applies the column generation approach to create optimal driver assignments and breaks for the trailer routes in y.

5.3.6 The a priori belief $\tilde{c}_1$

A priori, we have no belief for $\tilde{c}_1$, since we cannot say anything about the number of drivers or make assumptions about the bias. Therefore, we execute F_1 the first time normally without a bias and take as a measure the number of trailers.

5.3.7 The updating mechanism of $\tilde{c}_1$

As we described, the bias and critical interval can be observed from solution z after the second phase. Therefore the updating mechanism consists of observing the bias and the critical interval and setting these as the parameters for $\tilde{c}_1$. Since the lower bound is the same for both phases, by definition it holds that $\tilde{c}_1(x, \hat{y}) = c(x, \hat{z})$ after the update.

Let $\tilde{c}_1$ always represent the incumbent During experiments we found that the method does not converge. This is the same problem as we initially had with the CVRP and therefore we tried a similar solution: to always let $\tilde{c}_1$ represent the incumbent. However, with the GAP

this is theoretically not possible due to the structural properties of $\tilde{c}_1$. In other words, since the lower bound line does not uniquely determine the intermediate solution y, we can theoretically have a last solution $\hat{z}$ and an incumbent solution z^* such that the lower bounds are equal while $\tilde{c}_1(x, \hat{z}) \neq \tilde{c}_1(x, z^*)$. Therefore, with the structure for $\tilde{c}_1$ we have chosen, we cannot formally constraint this. However, we can informally let the incumbent influence the solution found by F_1, by considering also the bias and critical interval from the incumbent z^*.

In order to take into account the incumbent bias and critical interval, we adapt F_1 such that the five different repetitions use different combinations of weights for the toplines based on the incumbent bias and the last solution bias. Since we have two biases and two critical intervals, we are able to calculate four values that could be estimates for $\tilde{c}_1$. However, if we observe that the average height of the topline is the sum of the average height of the lower bound line and the average height of the biasline, we can subtract the average height of the bias line to remain with two different values: the average height of the lower bound line in the two critical intervals. Now we redefine $\tilde{c}_1$ to be a linear combination of the two values. Now let I_1 be the critical interval corresponding to the incumbent z^* and I_2 the interval corresponding to the last solution $\hat{z}$, and let $h^{lb}(I, z)$ be the average height of the lower bound line for solution z in the interval I. By solving the linear system of Equations (45) and (46), we can find weights α_1 and α_2 corresponding to I_1 and I_2 such that $\tilde{c}_1(x, y^*) = c(x, z^*)$ and $\tilde{c}_1(x, \hat{y}) = c(x, \hat{z})$, if these exist. Otherwise, we relax the first equation and set $\alpha_1 = 0, \alpha_2 = \frac{c(x,\hat{z})}{h^{lb}(I_2,\hat{z})}$.

$$\alpha_1 h^{lb}(I_1, z^*) + \alpha_2 h^{lb}(I_2, z^*) = c(x, z^*) \tag{45}$$

$$\alpha_1 h^{lb}(I_1, \hat{z}) + \alpha_2 h^{lb}(I_2, \hat{z}) = c(x, \hat{z}) \tag{46}$$

Using the values for α_1 and α_2, $\tilde{c}_1$ becomes

$$\tilde{c}_1(x, y) = \alpha_1 h^{lb}(I_1, z) + \alpha_2 h^{lb}(I_2, z). \tag{47}$$

Although using Equation (47) for $\tilde{c}_1(x, y)$ means that the incumbent is represented, it does not guarantee that in the first phase always a solution $\hat{y}$ is found for which $\tilde{c}_1(x, \hat{y}) < \tilde{c}_1(x, y^*)$. This is because the heuristic using five repetitions with different parameters is a very simple way to explore the solution space. Again, we compare with the CVRP, where for the CCLP we used a solver with the incumbent inserted. That basically meant that the solver tries to find a solution with a lower objective value, but returns the incumbent if none is found. We can define F_1 to do the same thing here: try to find a solution for which $\tilde{c}_1(x, \hat{y}) < \tilde{c}_1(x, y^*)$, but return

y^* if none is found. This then automatically terminates the procedure. Formally, this requires that $\tilde{c}_1(x, y^*) = c(x, z^*)$, although this cannot be guaranteed if we limit $\tilde{c}_1$ to be written as in Equation (47). However, we can model this situation to fit within the framework by redefining $\tilde{c}_1$ as

$$\tilde{c}_1(x, y) = \begin{cases} c(x, z^*) & \text{if } y \equiv y^* \\ \alpha_1 h^{lb}(I_1, z) + \alpha_2 h^{lb}(I_2, z) & \text{otherwise.} \end{cases} \tag{48}$$

This is impractical since it introduces additional discontinuity in $\tilde{c}_1$ which will jeopardize the feedback mechanism, but it shows that the framework for the feedback mechanism does not forbid these imperfections to exist.

5.4 Simple feedback mechanism for the resource assignment problem

Filling in the steps from Section 5.3 in the algorithm in Section 2.5 results in Algorithm 4 for the resource assignment problem.

Algorithm 4 Simple feedback mechanism.

```
1: procedure SimpleFeedbackRAP(x)
2:     y* ← createTrailerRoutesUsingDefaultCostMetric(x)
3:     z* ← createDriverAssignmentsForTrailerRoutes(x, y*)
4:     c̃1 ← getBiasAndCriticalIntervalFromSolution(x, z*)
5:     while true do
6:         ŷ ← createTrailerRoutesUsingC1(x, c̃1)
7:         if c̃1(x, ŷ) > c(x, z*) then return
8:         end if
9:         ẑ ← createDriverAssignmentsForTrailerRoutes(x, ŷ)
10:        if c(x, ẑ) < c(x, z*) then
11:            y* ← ŷ
12:            z* ← ẑ
13:            c̃1 ← getBiasAndCriticalIntervalFromSolution(x, z*)
14:        else
15:            c̃1 ← getBiasAndCriticalIntervalFromSolutions(x, z*, ẑ)
16:        end if
17:    end while
18: end procedure
```

Figure 15 illustrates the process for the example in Figure 10. Each column corresponds to an iteration. The first row illustrates the lower bound, the bias and the resulting topline in the first phase, while the second row illustrates the lower bound with the observed topline and

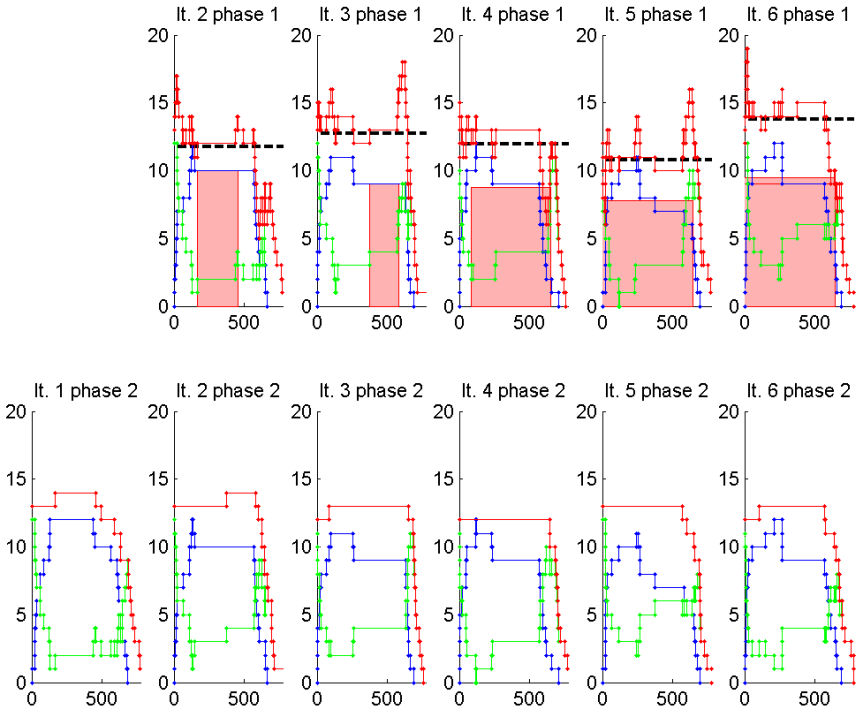


Figure 15: Illustration of the simple feedback mechanism using the topline

the resulting bias after the second phase. The black dashed line is the estimate $\tilde{c}_1$, which is a linear combination of the average height of the lower bound in the critical intervals. Only in iteration 6, there are two intervals, since iteration 2 is the first where the result is worse than the incumbent[2]. For the first phase in the first iteration, there is no visualization, since there is no bias yet.

5.5 Experiments

In order to test the simple feedback mechanism for the RAP, we use the 56 problem instances from Visser (2014). We let the solution from Visser be the initial solution, and compare it with the results of running the simple feedback mechanism until convergence, or a maximum of 10 iterations. In order to evaluate our convergence criterium, we also compare the result with the

[2]The number of trailers, although not visualized, is used as a tie-breaker, such that iteration 2 is a strict improvement over iteration 1 even though the number of drivers is equal.

result if we always run the mechanism for 10 iterations.

5.5.1 Results

In Table 7 the results are presented for the different instances. The numbers between brackets are the numbers of trailers, which are just shown to indicate that a better solution in terms of drivers, on average, also means that fewer trailers are used. In other words, it does not require more trailers to achieve a more efficient solution in terms of number of drivers.

From the table it can be observed that for a number of instances the feedback mechanism is able to improve on the initial solution, typically using one or two drivers less. Instance C104 in the table is the example in Figure 10, and the best solution found for this instance is visualized in Figure 16. The relative improvement may be small (4.96 % until convergence and 7.55 % after 10 iterations), but in absolute terms this is significant. Although the optimal solution is not known, it is very likely that using some lower bound it can be shown that gap with the optimal solution (the number of 'extra' drivers due to inefficient planning) after the improvement decreases significantly. The difference between 4.96 % and 7.55 % show that the convergence criterium can still be improved, since the mechanism in some cases stops too early. However, trading off the number of iterations against the obtained solution quality the criterium serves a clear purpose: it enables us to find (on average) 4.96 % improvement in (on average) 3.7 iterations while in order to find 7.55 % improvement we would need 10 iterations.

In Table 7 also the iterations in which the best solutions were found are presented, as well as the number of iterations until convergence. Note that the number of iterations until convergence is always more than the number of the iteration in which the best solution was found, since we always need an extra iteration to find out that, according to the belief $\tilde{c}_1$, the solution cannot be improved. We see that on average, the best solution is found within a few iterations, which means that also with limited time improvements in solution quality can be made.

Although the number of trailers is formally not an objective, we see as a side effect that this also decreases using the simple feedback mechanism, solely by using it as a tie breaker for solutions with the same number of drivers. Even for some of the instances for which we cannot find an improvement in the number of drivers, the number of trailers is decreased.

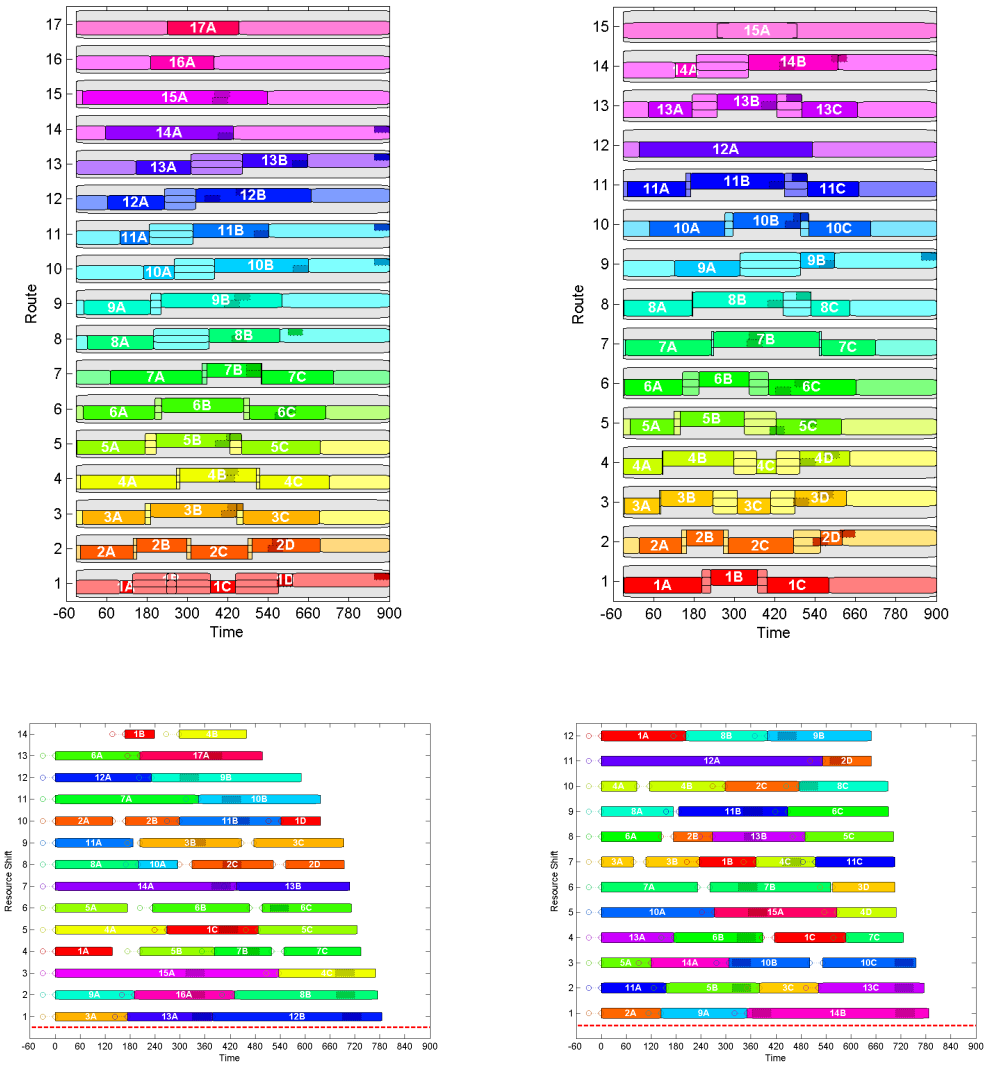


Figure 16: The initial solution (left, 17 trailers, 14 drivers) and the best solution (right, 15 trailers, 12 drivers) found for the C104 instance

Instance	Initial solution	Stopping criteria				10 iterations		
	Drv (Trl)	Drv (Trl)	Impr.	Found	Stop	Drv (Trl)	Impr.	Found
C101	21 (23)	20 (23)	4.76 %	2	8	19 (22)	9.52 %	9
C102	19 (22)	17 (19)	10.53 %	3	4	17 (19)	10.53 %	3
C103	18 (21)	15 (18)	16.67 %	3	5	15 (16)	16.67 %	6
C104	14 (17)	12 (15)	14.29 %	4	6	12 (15)	14.29 %	4
C105	19 (23)	19 (23)	0.00 %	1	2	18 (21)	5.26 %	5
C106	21 (24)	20 (23)	4.76 %	2	4	18 (22)	14.29 %	8
C107	21 (22)	19 (23)	9.52 %	2	3	19 (23)	9.52 %	2
C108	18 (22)	18 (22)	0.00 %	1	2	18 (22)	0.00 %	1
C109	16 (19)	16 (19)	0.00 %	1	4	16 (19)	0.00 %	1
C201	17 (20)	17 (20)	0.00 %	1	2	16 (20)	5.88 %	3
C202	17 (19)	15 (16)	11.76 %	2	4	15 (16)	11.76 %	2
C203	15 (19)	13 (15)	13.33 %	2	4	12 (15)	20.00 %	10
C204	13 (16)	12 (14)	7.69 %	2	3	12 (14)	7.69 %	2
C205	18 (20)	16 (19)	11.11 %	2	3	15 (19)	16.67 %	3
C206	15 (18)	15 (18)	0.00 %	1	2	15 (17)	0.00 %	2
C207	16 (17)	14 (16)	12.50 %	3	5	14 (16)	12.50 %	3
C208	15 (17)	14 (17)	6.67 %	2	5	14 (17)	6.67 %	2
R101	23 (23)	22 (22)	4.35 %	2	4	21 (22)	8.70 %	7
R102	21 (21)	19 (21)	9.52 %	5	6	19 (21)	9.52 %	5
R103	18 (19)	16 (17)	11.11 %	2	3	15 (16)	16.67 %	6
R104	13 (16)	13 (15)	0.00 %	3	4	12 (14)	7.69 %	6
R105	18 (20)	18 (20)	0.00 %	1	2	18 (19)	0.00 %	2
R106	18 (19)	16 (18)	11.11 %	3	7	16 (18)	11.11 %	3
R107	16 (18)	15 (16)	6.25 %	3	4	14 (15)	12.50 %	10
R108	13 (15)	12 (14)	7.69 %	4	5	12 (14)	7.69 %	4
R109	15 (17)	15 (17)	0.00 %	1	5	15 (17)	0.00 %	1
R110	15 (18)	15 (18)	0.00 %	1	2	15 (18)	0.00 %	1
R111	15 (17)	15 (17)	0.00 %	1	2	15 (16)	0.00 %	2
R112	14 (15)	14 (15)	0.00 %	1	2	14 (15)	0.00 %	1
R201	19 (19)	17 (18)	10.53 %	2	3	16 (18)	15.79 %	8
R202	17 (17)	14 (18)	17.65 %	2	-	14 (18)	17.65 %	10
R203	14 (15)	13 (14)	7.14 %	2	3	12 (15)	14.29 %	5
R204	13 (15)	11 (12)	15.38 %	4	5	11 (12)	15.38 %	4
R205	14 (16)	14 (16)	0.00 %	1	2	14 (16)	0.00 %	1
R206	12 (16)	12 (16)	0.00 %	1	2	12 (16)	0.00 %	1
R207	13 (14)	12 (15)	7.69 %	2	3	11 (14)	15.38 %	8
R208	11 (12)	11 (12)	0.00 %	1	3	11 (12)	0.00 %	1
R209	14 (16)	14 (16)	0.00 %	1	3	14 (15)	0.00 %	3
R210	14 (15)	14 (15)	0.00 %	1	5	13 (15)	7.14 %	7
R211	13 (14)	12 (14)	7.69 %	2	3	12 (13)	7.69 %	3
RC101	26 (28)	24 (26)	7.69 %	6	7	24 (26)	7.69 %	6
RC102	22 (24)	22 (24)	0.00 %	1	3	21 (22)	4.55 %	4
RC103	19 (22)	18 (19)	5.26 %	2	3	17 (19)	10.53 %	9
RC104	17 (20)	16 (17)	5.88 %	5	6	15 (16)	11.76 %	9
RC105	23 (24)	23 (24)	0.00 %	1	2	23 (24)	0.00 %	1
RC106	22 (24)	22 (24)	0.00 %	1	2	21 (23)	4.55 %	3
RC107	20 (22)	19 (20)	5.00 %	3	5	19 (20)	5.00 %	3
RC108	20 (22)	20 (22)	0.00 %	1	2	18 (22)	10.00 %	3
RC201	22 (23)	22 (23)	0.00 %	1	2	21 (23)	4.55 %	5
RC202	18 (21)	18 (21)	0.00 %	1	2	18 (18)	0.00 %	5
RC203	16 (19)	16 (17)	0.00 %	2	4	15 (17)	6.25 %	9
RC204	14 (16)	14 (16)	0.00 %	1	3	14 (16)	0.00 %	1
RC205	21 (21)	18 (20)	14.29 %	3	5	18 (20)	14.29 %	3
RC206	20 (21)	20 (21)	0.00 %	1	2	17 (18)	15.00 %	3
RC207	17 (19)	17 (18)	0.00 %	2	4	17 (18)	0.00 %	2
RC208	15 (16)	15 (16)	0.00 %	1	3	15 (16)	0.00 %	1
Average			4.96 %	2.0	3.7		7.55 %	4.1

Table 7: Results for the resource assignment problem

6 Discussion

The simple feedback mechanism turned out to be very effective for the three problems we have studied. Inspired by a theoretical framework for a feedback mechanism we looked into ways to implement the mechanism for three different problems, with different levels of difficulty. Some compromises had to be made which resulted in final implementations deviating from the theoretical concept. However, inspired by the theoretical concept, we succeeded in the goal of developing a mechanism that improves a solution in a few iterations using a simple feedback mechanism, without deep knowledge of the optimization techniques used in each phase.

We have seen that the efforts to achieve some sort of convergence increase with the complexity of the problem. The pallet matching problem is a very pure problem, upon which the feedback mechanism could be applied very effectively since the relation between the first phase and the second phase can be observed on a very detailed level. This is because the final costs can be decomposed into costs for each request matching. With the CVRP, the costs can be decomposed into costs for each route or cluster, but not for the individual assignment of locations to a seed. Therefore we needed to impose an extra constraint in the updating mechanism in order to ensure that we did not loose valuable information about the best solution so far. For the RAP, we needed to artificially introduce the relation between the two phases, since this could only be observed indirectly and for the solution as a whole.

Although the application of the framework varies for the three different problem, we can make one key observation. For each method, the entire two phase procedure is repeated, with an updated cost metric $\tilde{c}_1$ based on previous observations. None of the methods is based on heuristically 'fixing' suboptimalities in a solution. This is an advantage, since such heuristics usually take into account very specific details of the problem (for example specific driver regulations). This means that with different constraints the method may no longer be applicable. The simple feedback mechanism only observes the result and from a global perspective repeats the procedure with different information such that alternatives are explored. The details are indirectly represented through the information that is passed to following iterations via the belief $\tilde{c}_1$.

6.1 Conditions for the simple feedback mechanism

We have identified a number of ingredients that need to be defined in order to apply the simple feedback mechanism to an arbitrary two phase problem. Based on the results, we can formulate a number of conditions that influence the quality of the feedback mechanism.

- **The structural quality of $\tilde{c}_1$.** How well is it possible to capture the essence of the problem in the first phase?
- **The quality of F_1 and F_2.** Obviously, if F_2 is able to find good quality solutions, this is reflected in the overall solution quality since the overall solution is the direct result of F_2 for any intermediate solution. With the CVRP we have seen that also a better quality solution (a lower optimality gap) for the first subproblem results in a better overall solution.
- **The effectiveness of the updating mechanism.** Ideally the updating mechanism should only update $\tilde{c}_1$ according to what is observed and leave unchanged what is unobserved, on a level as detailed as possible. It should positively reinforce good decisions and negatively reinforce bad decisions. In other words, if $\tilde{c}_1$ is updated because a part of the solution y is considered good, then it should be likely that that part of the solution is also good in other (unobserved) solutions, and vice versa.
- **The level of optimism in the a priori belief $\tilde{c}_1$.** A higher optimism directs towards a more global search, although it may also direct the feedback mechanism in a wrong direction depending on the effectiveness of the updating mechanism (see the previous point).

6.2 Further research

First of all, further research could focus on further applications of the framework, for empirical evidence of the value of the framework. The results achieved with the RAP, which is a combination of a routing problem and a scheduling problem with additional constraints, indicate that it may be possible to apply the framework to many more multi phased problems that occur in practice. With respect to the framework itself, we ask a number of question which may serve as starting points for further research.

- How to define formal proofs and conditions for convergence of the mechanism?
- How to redefine $\tilde{c}_1$ such that it is not required to actually represent the true objective function, but only induces the precedence relation on y? In other words, rather than using Proposition 3, we use the less restrictive Proposition 2, which means that we only require that $\tilde{c}_1(x, y) < \tilde{c}_1(x, y')$ if $c(x, z) < c(x, z')$, where $z = F_2(x, y)$ and $z' = F_2(x, y')$. This gives more freedom but therefore it requires careful thought on how this should be implemented and how convergence should be achieved or stopping criteria should be applied.

- How to apply the framework for a problem where we formally want to take into account multiple objectives? The obvious solution is to use some cost function that scores a solution, but such a cost function is always means that there is a 'less natural' relation between y and $c(x, z)$. Possibly there are alternatives, such as not requiring $\tilde{c}_1$ to exactly represent the true objective function (see previous question).

- How to apply a hybrid use of optimal approaches and heuristics for the first and second subproblem? For instance, it may be possible to use heuristics for either F_1 or F_2 for the first iterations to quickly improve the solution, and use computationally more expensive approaches to find better solutions once the largest and obvious steps have been made. How does this influence the overall solution quality compared to the total computation time required?

A Results CVRP

This Appendix contains the full factorial design results for the different instances presented in Table 6.

Instance vrp1-50

Parameters				Initial solution		Incumbent		Convergence	
Method	Update	Tol.	γ	Val	Gap	Val	Gap	It	Time
'Optimal'	'Initial'	20.0 %	0.60	538.25	2.60 %	538.25	2.60 %	2	0.4 s
			0.80	593.75	13.18 %	593.75	13.18 %	2	0.4 s
		5.0 %	0.60	538.25	2.60 %	524.61	0.00 %	4	0.8 s
			0.80	539.22	2.78 %	539.22	2.78 %	2	0.4 s
	'Last'	20.0 %	0.60	538.25	2.60 %	538.25	2.60 %	2	0.4 s
			0.80	593.75	13.18 %	593.75	13.18 %	2	0.4 s
		5.0 %	0.60	538.25	2.60 %	524.61	0.00 %	4	0.8 s
			0.80	539.22	2.78 %	539.22	2.78 %	2	0.4 s
'Christof.'	'Initial'	20.0 %	0.60	594.86	13.39 %	594.86	13.39 %	2	0.3 s
			0.80	643.69	22.70 %	643.69	22.70 %	2	0.3 s
		5.0 %	0.60	594.86	13.39 %	578.50	10.27 %	7	1.5 s
			0.80	600.70	14.50 %	600.70	14.50 %	5	0.8 s
	'Last'	20.0 %	0.60	594.86	13.39 %	594.86	13.39 %	2	0.3 s
			0.80	643.69	22.70 %	643.69	22.70 %	2	0.3 s
		5.0 %	0.60	594.86	13.39 %	578.50	10.27 %	7	1.5 s
			0.80	600.70	14.50 %	600.70	14.50 %	5	0.8 s

Instance vrp2-75

Parameters				Initial solution		Incumbent		Convergence	
Method	Update	Tol.	γ	Val	Gap	Val	Gap	It	Time
'Optimal'	'Initial'	20.0 %	0.60	876.82	4.98 %	848.71	1.61 %	28	875.0 s
			0.80	853.24	2.15 %	841.93	0.80 %	9	177.9 s
		5.0 %	0.60	869.50	4.10 %	841.93	0.80 %	22	758.2 s
			0.80	841.93	0.80 %	841.93	0.80 %	13	361.2 s
	'Last'	20.0 %	0.60	876.82	4.98 %	842.85	0.91 %	21	461.6 s
			0.80	853.24	2.15 %	845.91	1.27 %	5	57.2 s
		5.0 %	0.60	869.50	4.10 %	854.86	2.35 %	26	638.6 s
			0.80	841.93	0.80 %	841.93	0.80 %	7	137.5 s
'Christof.'	'Initial'	20.0 %	0.60	931.62	11.54 %	872.70	4.48 %	26	653.4 s
			0.80	899.84	7.73 %	867.93	3.91 %	7	109.9 s
		5.0 %	0.60	930.68	11.42 %	867.78	3.89 %	20	540.5 s
			0.80	887.32	6.23 %	865.02	3.56 %	19	467.3 s
	'Last'	20.0 %	0.60	931.62	11.54 %	882.68	5.68 %	24	585.4 s
			0.80	899.84	7.73 %	863.77	3.41 %	12	148.5 s
		5.0 %	0.60	930.68	11.42 %	864.33	3.48 %	29	782.7 s
			0.80	887.32	6.23 %	874.47	4.69 %	16	319.9 s

Instance vrp3-100

Parameters				Initial solution		Incumbent		Convergence	
Method	Update	Tol.	γ	Val	Gap	Val	Gap	It	Time
'Optimal'	'Initial'	20.0 %	0.60	868.91	5.18 %	849.95	2.88 %	9	27.4 s
			0.80	887.28	7.40 %	887.28	7.40 %	3	3.5 s
		5.0 %	0.60	864.44	4.64 %	839.61	1.63 %	44	1243.4 s
			0.80	858.54	3.92 %	845.31	2.32 %	100	1984.1 s
	'Last'	20.0 %	0.60	868.91	5.18 %	850.82	2.99 %	5	19.4 s
			0.80	887.28	7.40 %	887.28	7.40 %	3	3.5 s
		5.0 %	0.60	864.44	4.64 %	848.92	2.76 %	26	753.9 s
			0.80	858.54	3.92 %	832.43	0.76 %	8	169.7 s
'Christof.'	'Initial'	20.0 %	0.60	933.55	13.00 %	911.99	10.39 %	9	18.5 s
			0.80	967.06	17.06 %	937.57	13.49 %	5	12.9 s
		5.0 %	0.60	947.39	14.68 %	876.77	6.13 %	29	895.7 s
			0.80	930.86	12.68 %	876.51	6.10 %	24	433.2 s
	'Last'	20.0 %	0.60	933.55	13.00 %	911.99	10.39 %	9	17.1 s
			0.80	967.06	17.06 %	961.44	16.38 %	5	6.7 s
		5.0 %	0.60	947.39	14.68 %	883.02	6.89 %	18	558.7 s
			0.80	930.86	12.68 %	891.45	7.91 %	11	234.1 s

Instance vrp4-100

Parameters				Initial solution		Incumbent		Convergence	
Method	Update	Tol.	γ	Val	Gap	Val	Gap	It	Time
'Optimal'	'Initial'	20.0 %	0.60	832.83	1.62 %	828.26	1.06 %	4	18.4 s
			0.80	862.33	5.22 %	862.33	5.22 %	2	5.1 s
		5.0 %	0.60	832.83	1.62 %	826.14	0.80 %	10	262.7 s
			0.80	832.83	1.62 %	826.90	0.90 %	4	68.6 s
	'Last'	20.0 %	0.60	832.83	1.62 %	828.26	1.06 %	4	18.2 s
			0.80	862.33	5.22 %	862.33	5.22 %	2	5.0 s
		5.0 %	0.60	832.83	1.62 %	826.74	0.88 %	5	136.1 s
			0.80	832.83	1.62 %	826.90	0.90 %	4	69.2 s
'Christof.'	'Initial'	20.0 %	0.60	862.68	5.26 %	857.47	4.63 %	10	71.5 s
			0.80	892.78	8.93 %	886.99	8.23 %	6	8.7 s
		5.0 %	0.60	862.68	5.26 %	855.41	4.37 %	100	3304.5 s
			0.80	862.68	5.26 %	856.89	4.56 %	5	100.3 s
	'Last'	20.0 %	0.60	862.68	5.26 %	856.28	4.48 %	7	50.0 s
			0.80	892.78	8.93 %	886.99	8.23 %	6	8.8 s
		5.0 %	0.60	862.68	5.26 %	855.64	4.40 %	13	340.7 s
			0.80	862.68	5.26 %	856.89	4.56 %	5	101.9 s

Instance vrp5-120

Parameters				Initial solution		Incumbent		Convergence	
Method	Update	Tol.	γ	Val	Gap	Val	Gap	It	Time
'Optimal'	'Initial'	20.0 %	0.60	1045.46	0.32 %	1045.40	0.32 %	12	159.6 s
			0.80	1052.54	1.00 %	1052.54	1.00 %	2	21.6 s
		5.0 %	0.60	1045.46	0.32 %	1045.42	0.32 %	17	992.1 s
			0.80	1052.54	1.00 %	1044.54	0.23 %	14	868.5 s
	'Last'	20.0 %	0.60	1045.46	0.32 %	1044.16	0.20 %	9	109.2 s
			0.80	1052.54	1.00 %	1052.54	1.00 %	2	21.6 s
		5.0 %	0.60	1045.46	0.32 %	1044.16	0.20 %	18	1007.6 s
			0.80	1052.54	1.00 %	1045.42	0.32 %	7	428.5 s
'Christof.'	'Initial'	20.0 %	0.60	1086.84	4.29 %	1080.78	3.71 %	9	163.4 s
			0.80	1091.05	4.70 %	1091.05	4.70 %	6	26.4 s
		5.0 %	0.60	1086.84	4.29 %	1080.78	3.71 %	18	1057.7 s
			0.80	1091.05	4.70 %	1084.85	4.10 %	10	565.3 s
	'Last'	20.0 %	0.60	1086.84	4.29 %	1082.27	3.85 %	8	133.5 s
			0.80	1091.05	4.70 %	1091.05	4.70 %	6	26.3 s
		5.0 %	0.60	1086.84	4.29 %	1083.09	3.93 %	17	998.9 s
			0.80	1091.05	4.70 %	1084.85	4.10 %	10	553.6 s

Instance vrp6-150

Parameters				Initial solution		Incumbent		Convergence	
Method	Update	Tol.	γ	Val	Gap	Val	Gap	It	Time
'Optimal'	'Initial'	20.0 %	0.60	1099.57	6.92 %	1052.73	2.36 %	33	2676.2 s
			0.80	1077.00	4.72 %	1060.12	3.08 %	6	48.8 s
		5.0 %	0.60	1099.57	6.92 %	1045.92	1.70 %	22	2583.0 s
			0.80	1076.66	4.69 %	1060.24	3.09 %	9	825.2 s
	'Last'	20.0 %	0.60	1099.57	6.92 %	1060.90	3.16 %	13	1199.8 s
			0.80	1077.00	4.72 %	1061.21	3.19 %	5	45.3 s
		5.0 %	0.60	1099.57	6.92 %	1045.49	1.66 %	28	3106.2 s
			0.80	1076.66	4.69 %	1053.24	2.41 %	10	897.0 s
'Christof.'	'Initial'	20.0 %	0.60	1167.38	13.51 %	1115.26	8.44 %	18	1214.9 s
			0.80	1160.93	12.88 %	1125.49	9.44 %	14	120.2 s
		5.0 %	0.60	1167.38	13.51 %	1109.44	7.88 %	46	5455.3 s
			0.80	1146.75	11.51 %	1115.43	8.46 %	15	1379.7 s
	'Last'	20.0 %	0.60	1167.38	13.51 %	1100.64	7.02 %	26	1480.7 s
			0.80	1160.93	12.88 %	1131.99	10.07 %	9	94.8 s
		5.0 %	0.60	1167.38	13.51 %	1105.68	7.51 %	38	4516.7 s
			0.80	1146.75	11.51 %	1103.84	7.33 %	13	1100.7 s

References

C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3): 316–329, 1998.

R. Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767, 1956.

J. Bramel and D. Simchi-Levi. A location based heuristic for general routing problems. *Operations research*, 43(4):649–660, 1995.

N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document, 1976.

N. Christofides and S. Eilon. An algorithm for the vehicle-dispatching problem. *OR*, pages 309–318, 1969.

P. Crescenzi and V. Kann. A compendium of np optimization problems, 1995.

L. Davis et al. *Handbook of genetic algorithms*, volume 115. Van Nostrand Reinhold New York, 1991.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1 (1):269–271, 1959.

M. L. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124, 1981.

F. Glover and M. Laguna. *Tabu search*. Springer, 1999.

I. Gurobi Optimization. Gurobi optimizer reference manual, 2014. URL `http://www.gurobi.com`.

A. Hyvarinen. Fast and robust fixed-point algorithms for independent component analysis. *Neural Networks, IEEE Transactions on*, 10(3):626–634, 1999.

J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

E. L. Lawler, J. K. Lenstra, A. R. Kan, and D. B. Shmoys. *The traveling salesman problem: a guided tour of combinatorial optimization*, volume 3. Wiley New York, 1985.

LEMON. Library of efficient models and optimization in networks, 2014. URL `http://lemon.cs.elte.hu/pub/doc/latest-svn/index.html`.

S. Martello and P. Toth. *Knapsack problems*. Wiley New York, 1990.

G. Pataki. Teaching integer programming formulations using the traveling salesman problem. *SIAM review*, 45(1):116–123, 2003.

R. C. Prim. Shortest connection networks and some generalizations. *Bell system technical journal*, 36(6):1389–1401, 1957.

P. Toth and D. Vigo. *The vehicle routing problem*. Siam, 2001.

P. J. Van Laarhoven and E. H. Aarts. *Simulated annealing*. Springer, 1987.

J. Venn. I. on the diagrammatic and mechanical representation of propositions and reasonings. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 10(59): 1–18, 1880.

T. R. Visser. Synchronization in simultaneous vehicle routing and crew scheduling problems with breaks. Master's thesis, Utrecht University, 2014. To appear in 2015.