SICS Technical Report 2018:01

DOIT WP4 Final Report on Planning and Optimization

Björn Bjurling and Martin Aronsson
RISE SICS AB

January 2018

1 Introduction

This report gives an overview of a selection of state-of-the-art optimization techniques for applying the Assignment Planning Use Case on a specific business case. The Assignment Planning Use Case was described in a previous DOIT WP4 report (SICS TR 2017:07). Recall that, in the Assignment Planning Use Case, the task is to minimize the total cost for a carrier to carry out a set of transport assignments by finding an as good as possible matching between transport assignments, routes, vehicles, and drivers.

The business case is taken from the area of transportation of timber from stores to factories. The characteristics of the business case is such that finding the optimal assignment plan becomes very hard and time consuming. Thus, the data provides a setting for illustrating the potential in using optimization techniques in timber transportation and similar cases. The data used for building the models in this report come from a timber transporting carrier and its customers. In order to protect potentially sensitive data, we shall not reveal any specific details about the carrier or its customers. We shall also use generic terms (such 'factories' above) and we have altered some aspects of the data in the modelling process.

The selection of optimization techniques represent commonly used techniques for problems similar to the present timber transportation case. In particular, we shall focus on three models: *Minimal Cost Flow*, *Constraint Programming*, and *Set Cover* models. The report will discuss the merits of each one of the modelling approaches.

The goal with this overview is to give insights into the considerations and the trade-offs one may encounter when choosing an optimization modelling paradigm for a problem typical for the transport sector. We shall for example see below that including aspects such as restrictions on driving time, which is both natural and important from an application point of view, excludes minimium cost flow models.

The work reported here is part of the DoIT project funded by FFI/Vinnova and lead by Scania. The DoIT project has focused on investigating methods for building and using data-driven cost models for increasing efficiency in planning for road-bound transportation. This report illustrates the use of optimization techniques and in particular what kind of input data that is useful for planning and mathematical modelling in the area of transportation. We hope that the report can provide a glimpse into the possibilities with operations research in the area.

The rest of the report is structured as follows. Section 2 first states the timber transportation problem. The problem is then put into the context of DOIT by giving an overview of the optimization work in WP4 in terms of the datasets and the modelling approcahes we have studied. In Section 3 we analyse the problem in terms of the Vehicle Routing Problem and present two models of two different kinds (the Minimum Cost Flow model and the Constraint programming model) for solving the our variant o fthe VRP. The merits of two approaches are then compared. In Section 4 we present a deeper look into the the use of the Set Cover model for the timber transportation problem. In particular, as the computational complexity of this approach is as very high, we pay special attention, in this report, to the heuristics devised in the project and used for solving a mixed integer program based on the set cover model. We illustrate how the heuristics has been used and show positive results where our plans improves on the performance of the carrier based on the real data we received.

2 The TIMBER Problem

In this section we formulate the problem and relate it to the previous work in DOIT/WP4.

2.1 The TIMBER Problem Formulation

In the TIMBER problem we have a carrier called TIMBER with 16 vehicles in its fleet. TIMBER is carrying out timber transport tasks that have been agreed between forrest owners and factories. The timber is loaded from stores in the forrest and offloaded at the factories. Two drivers man the vehicles each day with a total driving time of 16 hours per day. We assume that each driver drives 8 hours a day and that they take breaks according to regulations (here simplified to that no driver may drive for more than 4.5 hour without a rest). In the beginning of the working day the vehicle drives empty to the first store to pick up timber. In the end of the day, the vehicle returns to base empty after having offloaded its last transport task at one of the factories. At the end of the day, the timber that not have been transported remain at the stores. Such timber can be picked up on any day after.

Timber that remain overnight at stores deteriorate every day. The quality of the timber is classified as *green*, *yellow*, or *red* depending on how much it has deteriorated. We assume that the carrier gets paid according to the quality of the timber it delivers. We assume that the carrier is penalized when delivering the lowest quality timber, which is labelled red.

The problem is to find sequences of transport task (which shall be called *routes*) such that each vehicle can be assigned one route every day and such that the cost for the carrier is minimized.

Note that the models we describe below may make simplifying assumptions. For example, while the Set Cover approach can model the problem as it is formulated here, the Minimal Cost Flow (MCF) formulation cannot take into account the working hour regulations, at least not in the exact way. On the other hand the MCF is very fast in finding a solution.

2.2 TIMBER problem in DOIT/WP4

In WP4, we have studied several datasets and we have aimed for applying a range of different modelling techniques. Table 1 gives a summary of the datasets

	Task generation	Task detection	Transport-lab	TIMBER Partial 1	TIMBER Partial 2	TIMBER full
Minimum Cost Flow, MCF	X	(x)	(x)		X	
Constraint Programming, CP	X				X	
Set Cover, SC		(x)	(x)	X		
SC+Heuristics				X		X

Figure 1: Dataset and Modelling approaches. Cells marked with X signify that the approach (rows) was applied to the data (columns) in successful or interesting way. Those marked with (X) signify that modelling was performed but the resulting model was uninteresting.

and the modelling approaches used in WP4. Here follows some remarks on the datasets and the appraoches.

Task Generation The first dataset was based on random generation of tasks to be performed between randomly generated sources and destinations in a grid layout. The random distribution was based on the concept of preferential entailment in order to make the synthetic dataset more realistic (thus for example making some locations more of the character of sources and others more of destinations). KPI:s such as fuel consumption and travelling time was based on the Manhattan distance defined on the grid. This dataset was used to for illustrating the use the MCF and CP models. The dataset was particularly useful while searching for a suitable real case and dataset that could be shared with us.

Task Detection One approach in DOIT for obtaining transport assignment data was to devise algorithms for automatic detection of transport assignments based on vehicle operation data. This approach did not succeed in finding suffiently many transport assignments and the resulting dataset became very sparse. The dataset was nevertheless used for illustrating the interplay between the optimization implementation and the data-driven fuel and cost models as part of the overall goal of DOIT. In particular, an implementation of an early version of the Set Cover model was used on this dataset. However, the optimization task was trivial as the optimal solutions easily could be found by generating all possible routes and trivially choose the optimal assignments based on that.

Transport-lab A rich set of task data was made available to the project from Scania's own carrier Transport-lab. The dataset contained all attributes needed for WP4 to make a contribution to DOIT. However, it turned out that the tasks in the data were both regular and predictable. After initial modelling, it was clear that the regularity of the tasks made the scheduling task easy enough for humans to find sufficiently good solutions. In other words, optimization could not improve signficantly to the plans that the humans at Transport-lab already made by hand in Excel.

TIMBER This is the dataset that this report deals with. It is a large set and problem is severly complex with this data. The set contains a month worth of timber transports which approximately corresponds to 3000 transport assignments. We started with two smaller derived sets.

1. *TIMBER Partial 1* contains the transport assignments for the first day only (about 600 tasks). The first Set Cover model (the one mentioned above used for the Task Generation Case) was tested on the data. However, there was no realistic chance that the the model would produce a solution in a reasonable time frame (not even less than a day) due to the high computational complexity (the number of decision variables was in the order of 1 billion).
2. *TIMBER Partial 2* contains only the yellow tasks in the dataset. The yellow tasks are the ones where the timber is close to be consider as too low-quality and should therefore be prioritized so that the carrier can avoid being penalized for delivering timber marked as “red” (the lowest quality). The results of the modelling is presented below.
3. *TIMBER Full* contains all the entries in the TIMBER set. Being able to model this set was deemed to be a proof of success of WP4 in DOIT. Modelling this set required finding suitable and efficient heuristics for tackling the computational complexity. By using heuristics together with the Set Cover formulation, we managed within 40 minutes to find a plan for the whole set with a 7.2 percent improvement over the ground thruth with respect to volume transported per kilometer. This is described in Section 4.

3 Solution Techniques in DOIT

In this section we shall consider the two techniques Minimum Cost Flow and Constraint Programming. But first let us quickly recall the Vehicle Routing Problem.

The vehicle routing problem (VRP) formalizes the question "What is the optimal set of routes for a fleet of vehicles to traverse in order to deliver goods to a given set of customers?". Goods are delivered from one or more depots to one or more customers with requirements on the delivery. In the basic formulation of the VRP, each depot is home to one or more vehicles. In our formulation, the vehicles are based at places different from the depots. Given a road network connecting bases, depots (*stores* in our case), and customers (*factories* in our case), a solution is a set of routes over the road network (one route for each vehicle, beginning and ending at the vehicle’s base) such that all goods are delivered according to the customer requirement and such that the global transportation cost is minimized. The cost can be in terms of, for example, monetary cost or time. The VRP generalises the Travelling Salesman Problem and finding a solution is NP-hard[1]

3.1 Mixed Integer and Linear Programming

There are several ways to model the Vehicle Routing Problem (VRP). One classic way is to use a set cover model where tours consisting of atomic transportation tasks are formed. The solution is then found by choosing a subset of all tours that covers all the transportation tasks. This way of modelling the VRP, as a set cover, is described in Section 4 in this report. We will now turn our attention to a special case where a more efficient method could be used to model the VRP, to use a Minimum Cost Flow model (MCF).

3.1.1 Minimum cost flow

A MCF model can be used when there are no restrictions on the tours themselves, e.g. no restricion on the length of a tour. We may have simple restrictions on individual turns between tasks (e.g. some task may not follow after some other task) or legs in the tour but in MCF we cannot have restrictions that spans several turns.

The MCF model is powerful in terms of execution efficiency, but the expressive power is limited. One important application where the model is really useful is when all vehicles are of the same kind and there are no restriction on the length of a tour. Such an example is the construction of tours of vehicles (but not personnel) in repeated traffic according to a timetable, e.g. commuter traffic, bus services etc.

MCF returns a cyclic schema, i.e the schema and the tours are all cyclic. If the problem is not to create a cyclic schema, then we can still use the MCF model by introducing a dummy transport k which should start and end each tour. Thus, k has an end time that is less than all real transports start time (i.e. can turn into all other tasks), and a start time that is larger than all real transports. The dummy task is of course impossible in reality, since it starts long after it ends. It "takes the resources back in time" in order to make the tours cyclic. By minimizing the difference between start and end time of the dummy task the schedule becomes the most efficient one with respect to makespan.

3.1.2 Terminology

The following terminology is used.

k	Dummy transport, only in the problem to get the start and of every tour. There is a possible turn from k to every other real transport in the problem, and there is a possible turn to k from every real transport.

[1]The description of the VRP is based on the entry on Wikipedia `https://en.wikipedia.org/wiki/Vehicle_routing_problem`.

i	An index that vary over all the transports, including the dummy transport k at the beginning.
j	An index that vary over all outbound transport from a terminal, including the end transport k.
n	The number of transports in the problem
x_{ij}	Decision variable (binary): If $x_{ij} = 1$ then the arriving vehicle with transport i is reused in (turned into) transport j. Turns can only be made if there is enough time between i and j.
xp_{ij}	Binary variable to measure whether a turn can be performed or not. xp_{ij} is used to "count" the number of possible turns based on the respective transport's (task's) departure and arrival times, which can move within their respective earliest and lastest time windows. Used if the plan should maximize the number of replanning options.
T_{ij}	The minimum time it takes to carry out a turn from an inbound transport to an outbound transport. The time includes lead time on arrival and departure as well as a possible transit time from one terminal to another (without payload)
S_k	The number of vehicles used for transport k, in this case $S_k = 1$ for all $k \geq 1$. The number of vehicles allocated to the dummy transport S_0 determines the number of vehicles needed to solve the allocation problem.
p_i	The transport i's destination terminal
q_i	The transport i's origin terminal
a_i	The arrvial time of i. Every arrival has an upper bound, the latest arrival $\overleftarrow{a_i}$
d_i	The departure time of i, Every transport's departure time has a lower bound, the earliest departure $\overrightarrow{d_i}$.
tt_i	The transport time for i
D	The set of terminals, i.e. depos for vehicles, marking the start and end of a tour

3.1.3 The basic model

The model below describes a model based on the minimal cost flow model, MCF. The model can be thought of as a graph, where each node in the network constitutes a transport and each arc constitutes a turn of a vehicle from an inbound transport i to an outbound transport j through binding the variable

$x_{ij} = 1$. A solution consists of an assignment of all x_{ij} such that all transports S_i are supplied with a vehicle and all pick up and delivery times are obeyed.

If there are pick up or delivery restrictions then we need to impose equations on the turn variables x_{ij} if i's arrival time overlaps with j's departure time, since if the turn is to be made j must depart after i arrive. In such a case x_{ij} must be declared integer (binary) explicitly. If the model contains many such explicit declarations then this affects the performance of the execution of the model negatively. In a sense, when we have large overlapping domains we have also introduced the task of ordering the transports into the problem, which slows down execution.

Figure 2 is the example data called 'Task Generation Data', i.e. data generated synthetically, which were used in the beginning of the development. This represents a number of transports having pick up times and delivery times at different places. An exerpt from the data needed to compute the schedule and allocation for the vehicles is given in Figure 3. the 'Task Generation Data' as described in Section 2.2.

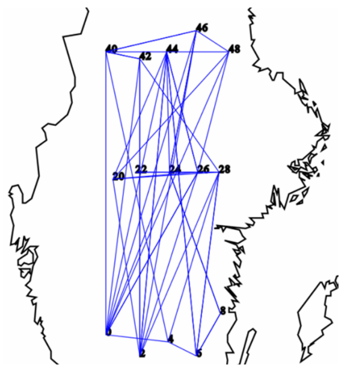
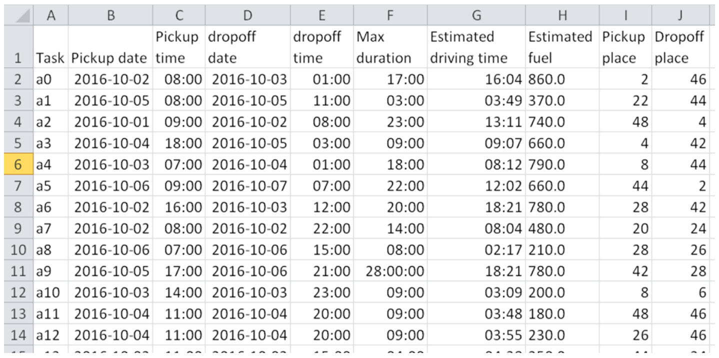


Figure 2: Graphing the data for MCF



	A	B	C	D	E	F	G	H	I	J
1	Task	Pickup date	Pickup time	dropoff date	dropoff time	Max duration	Estimated driving time	Estimated fuel	Pickup place	Dropoff place
2	a0	2016-10-02	08:00	2016-10-03	01:00	17:00	16:04	860.0	2	46
3	a1	2016-10-05	08:00	2016-10-05	11:00	03:00	03:49	370.0	22	44
4	a2	2016-10-01	09:00	2016-10-02	08:00	23:00	13:11	740.0	48	4
5	a3	2016-10-04	18:00	2016-10-05	03:00	09:00	09:07	660.0	4	42
6	a4	2016-10-03	07:00	2016-10-04	01:00	18:00	08:12	790.0	8	44
7	a5	2016-10-06	09:00	2016-10-07	07:00	22:00	12:02	660.0	44	2
8	a6	2016-10-02	16:00	2016-10-03	12:00	20:00	18:21	780.0	28	42
9	a7	2016-10-02	08:00	2016-10-02	22:00	14:00	08:04	480.0	20	24
10	a8	2016-10-06	07:00	2016-10-06	15:00	08:00	02:17	210.0	28	26
11	a9	2016-10-05	17:00	2016-10-06	21:00	28:00:00	18:21	780.0	42	28
12	a10	2016-10-03	14:00	2016-10-03	23:00	09:00	03:09	200.0	8	6
13	a11	2016-10-04	11:00	2016-10-04	20:00	09:00	03:48	180.0	48	46
14	a12	2016-10-04	11:00	2016-10-04	20:00	09:00	03:55	230.0	26	46

Figure 3: Example data for the MCF

In order to get an optimized solution we need some objective function to optimize on. A common objective is to minimize the resources needed to perform all the tasks, especially if the tasks are already scheduled in time. Another common objective is to optimize on makespan, i.e. the shortest period of time that the whole set of tasks can be performed, respecting the resource limitations. We will argue that in many cases, specifically when planning with uncertain information, an important objective is to produce a plan with many re-planning opportunities. The rationale behind this is that uncertainty means that assumptions (that the plan rests on) will change over time. The more opportunities that the plan has to be replanned, the more 'safe' it is. It is often better to trade a bit of makespan efficiency in favour of having more replanning opportunities. The plan then becomes 'self-healing' and more robust: This in turn build confidence in the plans and that they are useful to work with.

In Section 3.1.4 below, we have formulated these different aspects of optimality as different and alternative objective functions.

3.1.4 Equations

The following equations are used to build up the optimization model. Observe that not all formulas are used in all problem variants.

0 Objective functions

a) Minimize travelling without payloads.
$\sum_{ij} T_{ij} x_{ij} = DHT$ where T_{ij} is the time between transport i's arrival time and j's departure time.
DHT is then the total transport time without payload. $T^d_{ij}$is 0 if $p_i = q_j$

b) Minimize the number of vehicles used in the problem
$\sum_j x_{kj} = N$ where N is the number of vehicles used in the solution. If minimized on, we get the least amount of resources to solve the problem.

c) **Minimize makespan**
Makespan is the shortest time all the transports can be made in,
The simplest way to do that is to introduce a special variable MS which is larger than all real transports end time, and then minimize that variable
$\forall k : MS - a_k > 0$

d) Maximize replanning options
$\forall ij : \sum_{ij} xp_{ij} = PT$ where PT is the number of replanning options. In order to maximize the number of possible turns for each transport, we introduce a "shadow variable" xp_{ij} to the "real" turn variable x_{ij}. This variable is used in the object function to count all possible alternate turns that the solution will have.

$$\begin{cases} \forall ij: d_j - a_i - M\,xp_{ij} \geq -M + Setup \\ \forall ij: d_j - a_i - M\,xp_{ij} \leq Setup \end{cases} \quad xp_{ij} \text{ is 1 if the vehicle}$$
can turn from i's arrival added with *Setup* to departuring transport j
The two equations realizes an equivalence relation using two implications: $d_j \geq a_i + Setup \rightarrow xp_{ij} = 1$ and $d_j \leq a_i + Setup \rightarrow xp_{ij} = 0$

1 Flow conservation equations.
$\forall i: \sum_j x_{ij} = S_i$
$\forall j: \sum_i x_{ij} = S_j$
Note that all $S_i = 1$ and $S_j = 1$ except for the dummy transport S_0which measures the number of vehicles used in the problem.

3 Transport time for all tasks
$\forall k: a_k - d_k = tt_k$
The transportation time for transport k is the arrival time subtracted by the departure time

4 Overlapping turn times
If there is a turn from i to j then de departure of j must be greater than the arrival of i plus the necessary setup time between the two transports.
Logically this is expressed as $\forall ij: x_{ij} = 1 \rightarrow d_j \geq a_i + Setup$ provided that $\overrightarrow{a_i} > \overleftarrow{d_j} \wedge \overleftarrow{a_i} \leq \overrightarrow{d_j}$.
This is translated into the (linear) equation
$d_j - a_i - M\,x_{ij} \geq -M + Setup$
with the use of the so called big M method, i.e. M is a large constant that dominates the equation. It is necessary to introduce a binary declaration on x_{ij} here, since x_i occurs in other equations than in the column and row sums in equation 1, and thus destroys the totally unimodular property (see section 3.1.5).
Note that if $\overrightarrow{a_i} \leq \overleftarrow{d_j}$ the turn is always possible (with respect to timing) and we do not need the conditional equation, and if $\overleftarrow{a_i} > \overrightarrow{d_j}$ then the turn is always impossible and can thus be removed.
Note also that the setup time may include moving from i's arrival terminal to j's departure terminal without payload.

5 Work shifts
In order to implement work shifts, we introduce forbidden times to arrive (analogously for departing). In the model we introduce a new variable y which is interpreted as the day the transport k arrives in. a_k is restricted to be within 6:00 and 18:00.
$$\begin{cases} a_k - 24 * 60\,y_k \leq 18 * 60 \\ a_k - 24 * 60\,y_k \geq 6 * 60 \end{cases} \quad y_k \text{ integer}$$
These two equations force a_k to start within working time limits, i.e. after 6:00 and before 18:00. Analogous equations can be formed

for the delivery times.

3.1.5 Complexity

The great advantage of the MCF model is that as long as there are no restrictions on x_{ij} there is no need to declare these as integer (binary). A MCF of this type guarantees integer solutions for all x_{ij} as long as the sums S_i and $S_j$are integers (in this case 1) and no further equations references x_{ij}[2]. Under these conditions, the variable matrix is totally unimodular. Since a driving factor for complexity in MIP problems is the number of integer declarations, this is a real advantage.

Restrictions on x_{ij} arise for example if there are overlapping time windows between transport i and j, as formulated in equation 4 above. We need to do this for all x_{ij} when the choice of how the different transports are placed in time in their respective time windows will determine if the turn is possible or not. These x_{ij} have to be declared explicitly as binary in the model. The number of declarations depends on the time window sizes for the transports, potentially $\frac{n \times n}{2}$ [3]. This means that the larger the time windows are, more x_{ij} has to be declared binary and hence the complexity an execution time grows to find an optimal solution. The TIMBER case is such an example, where most of the tasks can be performed in large time windows, leading to possible overlaps between the task's execution times.

3.1.6 About the execution time to prove an optimal solution

There are cases where the MCF model does not reach a proven optimal result. In many of those the system has reached an objective value close to optimum, commonly just a few percent. It is worth noting that in many of those cases the system is already better than e.g. the CP model, and from a practical point of view it is sufficient to stop the execution. The uncertainty in input data is often larger than optimizing the last percent. A typical curve relating the objective value and the execution time is given below.

Note that the method used (simplex) knows how far from a proven optimal value it is, since it works with two limits, one is the so far found best solution and the other limit is the dual value that it cannot be better than. Thus we always know how far away the current solution is from the proven lower bound, but it can take long time to prove the the current found solution actually *is* the optimal value.

[2]This is so because this restriction that x_{ij} is only used in the summation formulas for S_i and S_j give the problem the *totally unimodular* property, i.e. that all sub matrixes' have determinant $+1$ or -1.

[3]If we do not have an upper limit on empty cargo between cities, assumed for convenience, this happens immediately, then the first transport that is done may potentially turn into all others, while the last one in the order can not turn anywhere else than in in our added dummy transport. Thus, the number of possible turns decreases by 1 for each transport in a vector arranged for arrival time.

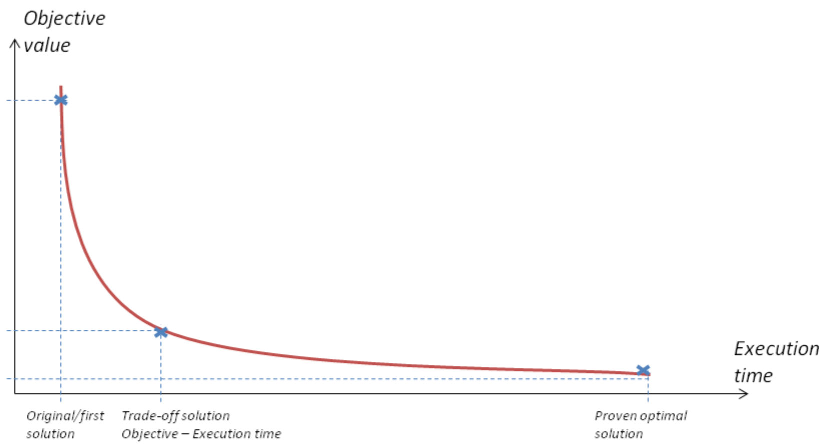


Figure 4: To improve the first or original solution is often made in short execution time, but the improvement gets lesser with time. From a practical point of view we can often stop the execution when we have reached within a few percent from proven optimal solution.

3.1.7 Applying MCF on the TIMBER Case

For the purpose of comparison, we have used the same datasets for MCF and for the CP-model in Section 3.2. Actually the two models are completely interchangeable, using the same data sets as input. Thus they are fully comparable. In all cases reported here the MCF model outperforms the CP model regarding the KPIs we want to measure. We have introduced a time limit on the execution, since as discussed above the time to actually prove an optimal value can be long, while the first solution delivered by the MCF model almost imediately reach a better objective value than the CP model. We have limited the execution time to 10 minutes.

As in the CP model we have concentrated on the stores which have timber that have begun to detoriate and which are important to get to the mills, i.e. the logs that are marked as "yellow" in the input data. This is the same input set as the CP model uses.

3.1.8 Test runs with the MCF model

If we maximize only on replanning possibilities (Fig 5), we get a schedule and allocation with all the tasks spread out as much as possible, as shown in the gantt schema below. There are 4934 options to change a turn into another one. The makespan for this solution is 6.75 days.

This schedule is not a good one regarding the efficiency, i.e. usage of the vehicles. We should probably trade the number of replanning options for better usage of the vehicles. We can do that in two ways, all in one run (i.e. have both efficiency and replanning options in the same objective function) or make

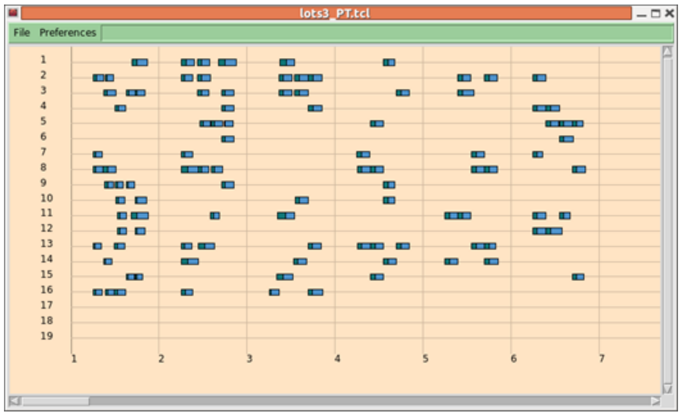


Figure 5: Maximizing replanning possibilities

separate runs. When doing it as tow runs, we have again two opportunities, either first optimize on replanning options and then minimize the makespan, or the other way around, first minimize makespan and then maximize replanning options. We have done the latter one since the number of replanning options is very large.

Fig. 6 shows the balanced solution in one combined run. We get just slightly less number of possible changed turns, 4763, while the makespan is improved significiantly, 30 %, to 4.74 days. Note that the solution obeys the fact that no transports are performed during night.

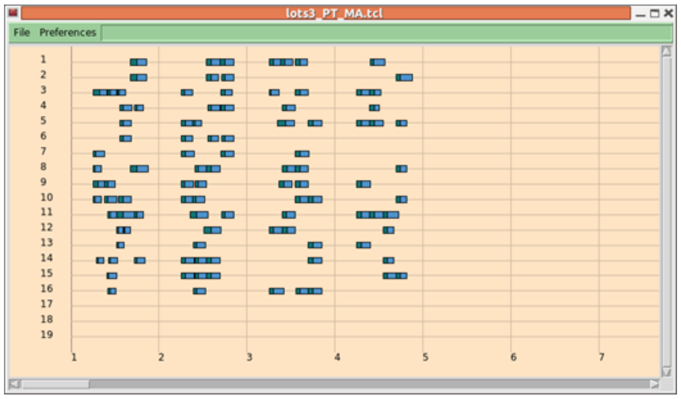


Figure 6:

By first optimize on makespan, which gives us an end limit for the whole plan, and then do a second run to maximize the replanning options within the limitted makespan, we get the solution in Fig 7. This is the preferred one, since makespan is further improved to 3.61 days (gaining 24 % compared to the balanced solution) while the number of replaning options is only decreased to 4658 (i.e. only 2 %). This last schema shows quite good efficiency, and thus the two-pass model is what is recommended in this case. It gives good efficiency while at the same time have reasonable execution times and good replanning opportunities.

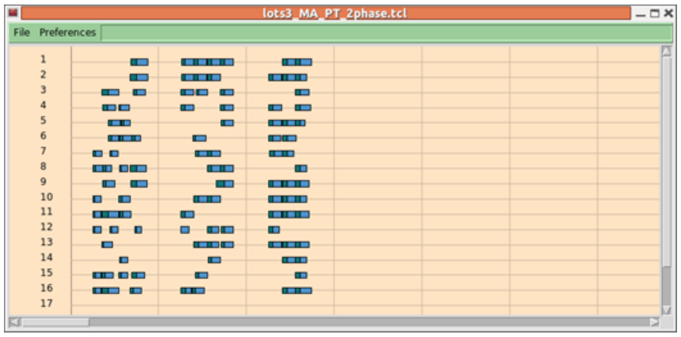


Figure 7:

3.2 Constraint Programming

This section gives a short introduction to the constraint programming model developed in the DOIT project. The model is not meant to be a complete application but rather an example implementation to be compared to the Minimum Cost Flow model, both regarding model structure and efficiency. We have used the constraint system embedded within SICStus Prolog[4].

3.2.1 What is a Constraint Program

A constraint programming module is often embedded in an existing programming language, which could be C++, Prolog, Java or any other programming language. The most common type of constraint programs are over finite domains (FD), but there are other types also, e.g. continous variables. We will in this presentation restrict ourselves to finite domains.

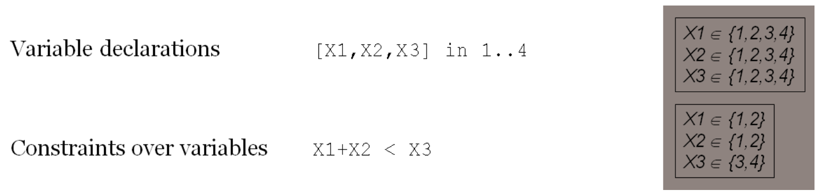
The constraint programming language consists of a number of modelling constructs over a special type of variables. These variables are special in the sense that they have a domain of possible values associated with them. During search, that domain is shrunk by deleting values until it only constist of a single element, which is then assigned to the variable.

The modelling constructs are of two types: simple arithmetic relations and more complex ones called global constraints. Global constraints have special algorithms that are encapsulated in the constraints, and typically states some modelling property to hold between all the variables mentioned in the global

[4]Mats Carlssn et. al. Swedish Institute of Computer Science. Release 4.3.5 December 2016, https://sicstus.sics.se/sicstus/docs/latest4/pdf/sicstus.pdf

constraint. For example, `all_different([X,Y,Z])` states that the variables `X`,`Y` and `Z` all should be different, whatever value the search procedure will find for them.

A small example of a constraint program is shown below in Figure 8. In the grey area to the right the resulting domain restrictions are shown after variable declaration and after posting the constraints to the constraint store.



```
[X1,X2,X3] in 1..4
X1+X2 < X3
```

Figure 8: CP: Simple example of variables and constraints

The constraint store is a graph consisting of the domain variables as nodes and the constraints as (hyper)arcs between the nodes. Whenever a variable's domain is shrunk, called *pruning*, all constraints connected to the variable are pushed onto a stack for validity checking, a process called *propagation*. During this check other variables' domains can be shrunk, and the corresponding constraints are also pushed onto the stack. This process is continued until either the stack is empty (no more constraints to check) or a constraint is found to be invalid, in which case the the prunings are unwound. This process is referred to as a fix point algorithm: whenever a variable's domain is shrunk, the pruning and propagation algorithm starts and continues until no more prunings can be made.

In order for the constraint system to be able to find a value for the variables, it needs a search procedure. The search procedure is an iterated procedure that uses the propagation and pruning algorithm in each step. The search consists of three phases, where all phases are indeterministic (i.e. the choice can be undone and another path in the search can be taken):

1. Choose a variable not yet bound to a value
2. Make a choice to restrict the domain of that variable, thus kicking the pruning and propagation algorithm to execute. One of two things can happen:
   (a) If that choice later turns out to lead to failure by the pruning and propagation algorithm, the search procedure backtracks to the latest choice point and make a new choice
   (b) If the pruning and propoagation algorithm succeed (i.e. a fixpoint is reached), the search procedure is iterated at step 1 again, but keeping the choicepoint if further search step should show that there are no consistent bindings to the variables.

3. The search terminates with success if all variables can get a value which satisfies all constraints, in which case the variables are said to be ground (have ground values). The search terminates with failure if there are no choices left and no consistent variable bindings exist.

Continuing the small example above we get a solution during search, which is dependent on how we search the search tree. Figure 9 illustrates in the gray area to the right.

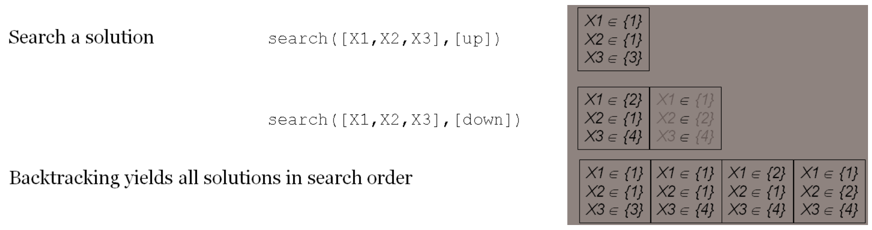
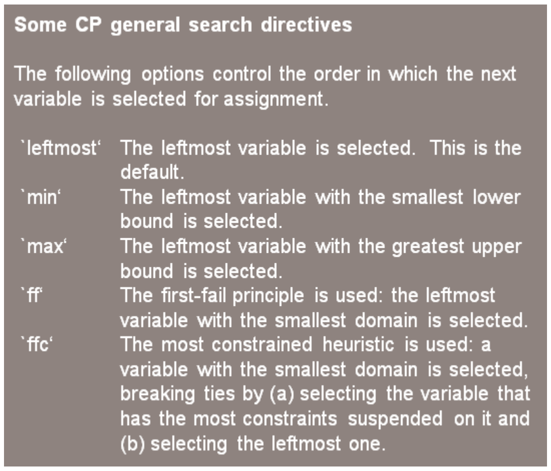

Search a solution
search([X1,X2,X3],[up])
X1 ∈ {1} X2 ∈ {1} X3 ∈ {3}
search([X1,X2,X3],[down])
X1 ∈ {2} X2 ∈ {1} X3 ∈ {4}
X1 ∈ {1} X2 ∈ {2} X3 ∈ {4}
Backtracking yields all solutions in search order
X1 ∈ {1} X2 ∈ {1} X3 ∈ {3}
X1 ∈ {1} X2 ∈ {1} X3 ∈ {4}
X1 ∈ {2} X2 ∈ {1} X3 ∈ {4}
X1 ∈ {1} X2 ∈ {2} X3 ∈ {4}

Figure 9: Example of search results

The basic predefined variable selection strategies and domain restriction policies commonly found in many constraint systems are described in Figure 10 below. These are combined to configure the basic procedure implemented in the constraint system. Most constraint system also offer the possibility to implement own variable selection algorithms and pruning strategies through a special programming interface.

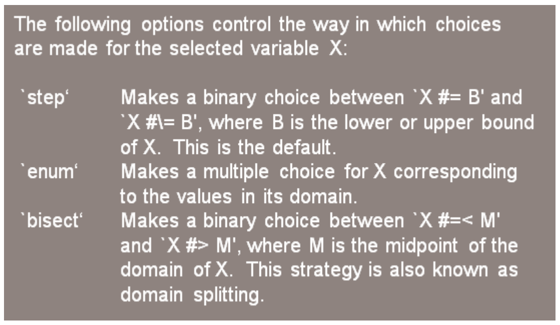

Some CP general search directives

The following options control the order in which the next variable is selected for assignment.

\`leftmost'	The leftmost variable is selected. This is the default.
\`min'	The leftmost variable with the smallest lower bound is selected.
\`max'	The leftmost variable with the greatest upper bound is selected.
\`ff'	The first-fail principle is used: the leftmost variable with the smallest domain is selected.
\`ffc'	The most constrained heuristic is used: a variable with the smallest domain is selected, breaking ties by (a) selecting the variable that has the most constraints suspended on it and (b) selecting the leftmost one.

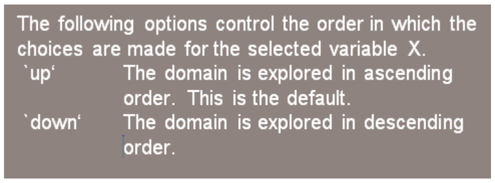
Figure 10: Strategies for variable selection

The above described algorithm finds feasible solution but does not perform any optimization. To get an optimal value another search algorithm is wrapped around the solution search algorithm. This is done quite simple in the following


The following options control the way in which choices are made for the selected variable X:

\`step'	Makes a binary choice between \`X #= B' and \`X #\= B', where B is the lower or upper bound of X. This is the default.
\`enum'	Makes a multiple choice for X corresponding to the values in its domain.
\`bisect'	Makes a binary choice between \`X #=< M' and \`X #> M', where M is the midpoint of the domain of X. This strategy is also known as domain splitting.

Figure 11: Policies of for restricting variables


The following options control the order in which the choices are made for the selected variable X.

\`up'	The domain is explored in ascending order. This is the default.
\`down'	The domain is explored in descending order.

Figure 12: Search domain direction

way. A variable is chosen which should be minimized or maximized. If a solution is reached with the previuosly mentioned search algorithm, the value of the designated minimzation (maximization) variable is stored as the currently best bound. All other variables' values are also store. Then the optimization procedure forces the execution to backtrack thus exploiting the nearest choicepoint which will make the search take another branch in the search tree. If the search should come upp with a better value for the minimzation (maximization) variable then the best bound is updated as well as the stored best solution. This process is repeated until there are no more choicepoints left to explore.

This search for an optimal value performed in this way is quite weak. There is no built-in guidance that could guide the search to optimality, as there is in traditional OR algorithms used in linear programming systems. Optimality search in Constraint Programming is more of a test algorithm, although built around a quite efficient search for satisfiability. For example, where the linear programming algorithm in each iteration "knows" which path that leads most to optimum and thus performs that step, constraint propagation is "blind" to which path that can improve the optimal value.

3.2.2 The basic model for the TIMBER example

The basic model is built upon a tutorial made by Philip Kilby [5]. The key component is a global constraint for forming an Hamiltonian circuit. An Hamil-

[5]Constraint Programming for Vehicle Routing Problems, Philip Kilby, Tutorial held at CP2013. Slides available at link: http://cp2013.a4cp.org/slides/t3.pdf

tonian circuit is a graph where all nodes are visited exactly once, and the graph is closed forming a circuit. The global constraint is stated as a vector, where the positions in the vector are the node's identification number, and the value assigned to the variable in the vector's position is the arc to the next node in the circuit. The interpretation in our domain is that a vehicle goes from one task i to the next task j, which is often referred to as “turning the vehicle from task i to task j”.

3.2.3 Constraints used in the model

In the figure 13 the hamiltonian circuit is the first (upper) vector. In this case we have 4 vehicles and 10 tasks. Note the two coloured rectangles to the right, they represent the depo(s) where the vehicles start and end. To model the depos we introduce 8 dummy tasks, 4 “leaving-depo-tasks” and 4 “end-at-depo-tasks” which are at the depos. The arcs going from the rightmost rectangles (end-at-depo-task) to the left one (leaving-depo-task) are more of a technical nature and does not represent an actual movement of the vehicle. THese technical arcs make the cycle hamiltonian.

The next vector in figure 13 states that if there is a turn from one task to the next in the upper vector, then they must use the same vehicle. There is no global constraint currently implemented for this so we use the `element(I,Vec,Val)` instead, where `I` is a variable over the positions in the vector `Vec`, and the value `Val` is the value of the `I`:th position in the vector `Vec`. We thus have as many `element/3` constraints as there are positions in the vector, since each position need its own constraint. Mathematically we write $Vec_I = Val$.

The third vector represents the starting times for each task. If there is a turn from task i to task j, then task j must start after task i's start time plus task i's duration plus reposition time for the vehicle to get to task j's starting position. We use the `element/3` constraint here too, to build up all necessary constraints between the differernt tasks.

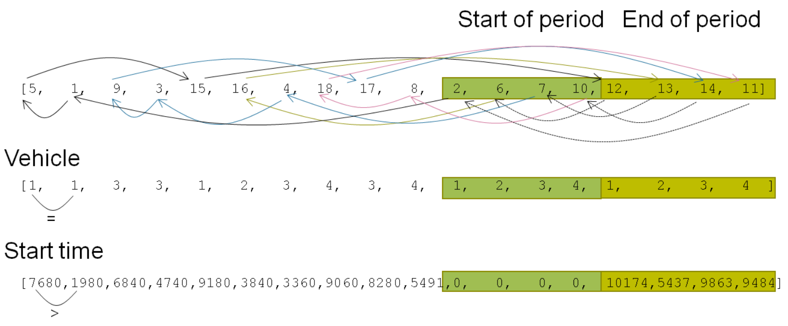


Figure 13: Key global constraints

For the vehicles this suffices to create valid schedules, but since humans work in shifts we have to introduce restrictions when the vehicles are allowed to run. TIMBER uses two shifts, and in order to model that we introduce another global constraint `disjoint/2`. This constraint is a geometric one, where a set of rectangles are certain to not overlap inside a large rectangle. The idea is that the X-axis is the time and the Y-axis represents the vehicles. Thus each row in the `disjoint/2` area is a vehicle number (the same as in the second row in figure13). Figure 14 illustrates this idea.

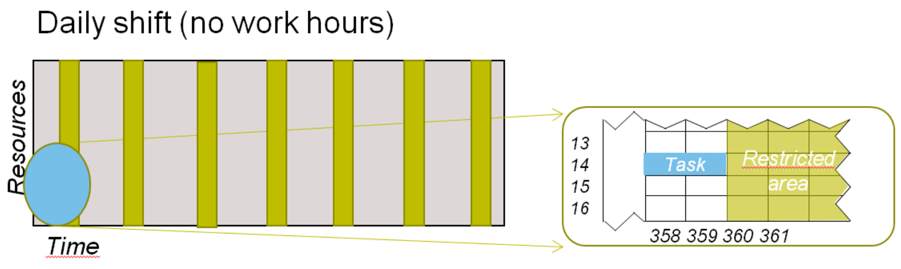


Figure 14: Constraints for work shifts

In the `disjoint/2` constraint we add static rectangles for the time periods where there is no work to take place. We then add all tasks to the disjoint/2 constraint as well. These can move along the X-axis as long as they are inside their time restrictions and along the Y-axis according to the vehicle they gets allocated to. The coloured rectangles in the figure are the no-work hours. Each task must now start and end within the grey areas and not overlap with the coloured areas, which is exemplified with the blue task in the blown-up part to the right.

3.2.4 Example set: the TIMBER restricted case

The example test case used in here, as well as in the other tests in this report, is a dataset collected from TIMBER . The tasks are transportation of timber from stores in the forrest to paper and saw mills. The timber may be at the stores for a quite short time before they begin to deteriorate. This means that the objective is to fetch all the logs before they deteriorate and the quality gets so low that the mills cannot use the timber any longer.

We have for this example set taken all the timber which are regarded as being still good for delivery but soon to become too low quality and thus lost, at a certain point in time (previuosly termed “yellow”). These volumes are then splitted into vehicle loads. Each such load is a task to be performed, and each task has an earliest pickup time and a latest pickup time inherited from the original data and the known detoriation. There are 102 tasks present in our dataset.

3.2.5 Runtime behaviour

The constraint program is heavily dependent on the variable selection algorithm and how the domain is restricted, described earlier. The search space is huge, if we consider that time is in fact discretized (compare that with the Minimum cost flow presented earlier in this report).

There is also the possibility to program a variable selection algorithm from scratch, as well as a domain restriction policy. We have not done that but combined a number of the predefined ones. These give different search behaviuor and quite different results, both regarding the actual plans and regarding search time. We present the tested ones below.

As can be seen, different search strategies gives rise to quite different layouts in the gantt schemas. The gantt schemas also reveal that it is just not enough with the standard search schemas. One would like to allocate the tasks in such a way that the vehicles are equally used and minimizing the overall ready time. None of the combinations gives a satisfactory layout of the schema.

Also note that the search space is large already with this small task set. As a result, for some of the combinations the execution does not end within reasonable time (in this case, 5 minutes). All of the other ones end within 9 seconds. Note that we only search for the first, satisfactory solution here. Optimization is another and harder task for the system to solve, and we have not reached a satisfactory result for the optimization case regarding execution time.

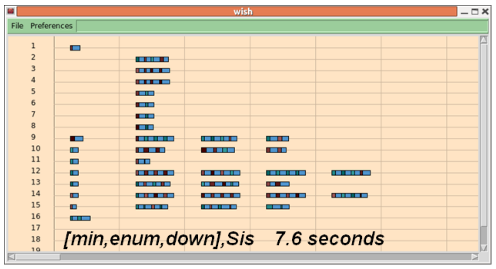


Figure 15: MinEnumDown

The search strategy presented in figuer 15 is based on first getting values for all the turn variables. When this is completed, all other variables are bound to values in a phase 2. As can bee seen it does not balance the load equally between the trucks.

As in figure 15, we have in figure 16 a two-phase solution procedure where we first fixate the turn variables and then all other variables. We use the forst-fail-principle for selecting a variable, and change the search direction for the variable restriction policy to "up". As can be seen, the schulede is quite different compared with the previous one.

These two examples shows that it matters quite a lot which direction we restrict the variable, i.e. if we search it from lowest value and up, or the other

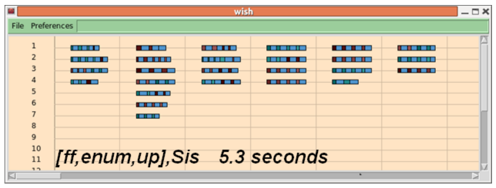


Figure 16: ffEnumUp

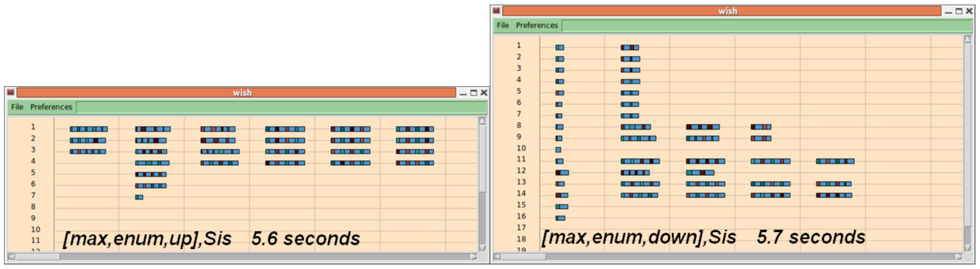

wish
File Preferences
[max,enum,up],Sis 5.6 seconds
wish
File Preferences
[max,enum,down],Sis 5.7 seconds

Figure 17: MaxEnumUp vs MaxEnumDown

way around. This schedule is quite different compared to the previous one, and the only difference in search configuration is the direction the domains are searched.

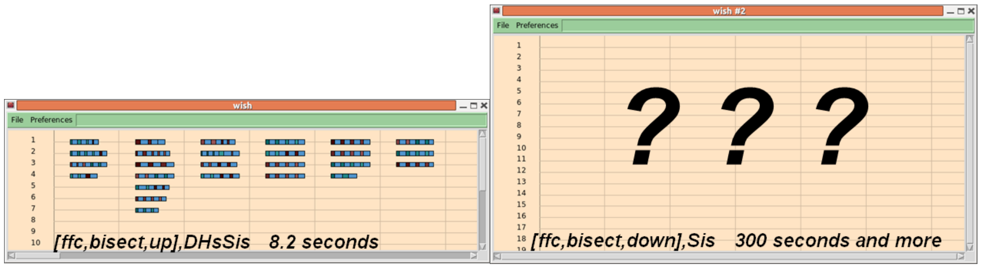

wish
File Preferences
[ffc,bisect,up],DHsSis 8.2 seconds
wish #2
File Preferences
? ? ?
[ffc,bisect,down],Sis 300 seconds and more

Figure 18: ffcBisectUp vs ffcBisectDown

The only difference in search configuration between the two examples in figure 18 is the search direction. This shows that the search configuration could be quite delicate, and by taking the wrong one we end up with execution inefficiency.

All the examples showed here are just for finding the first solution. When trying to optimize on e.g. maximum of rerouting possibilities (i.e. shortest overall comletion time), the execution times increases substantially, and none of the search configurations showed here are able to finish with an optimal value. For the best search strategy of the above we get 4256 number of rerouting possibilities with 10 minutes of execution, which is about 86 % of what the

MCF model gets in 10 minutes, and the first solution found by the MCF model is already better than the final solution delivered by the CP model after 10 minutes execution. Even with additional 20 minutes we only get an increase with 0.02 % (i.e. 1 additional rerouting possibility). The reason for this is almost certainly that the time is discrete as it is modelled as an FD variabel and thus part of the search (i.e. every time period is a domain in minutes, and this domain can be quite large). This means that the search space gets large. The time variables are actually not decision variables, since it is the allocation order of the tasks to vehicles that is important. But since the pruning of the time domains are to weak, we have to incorporate the time variables in the search in order to get a satisfactory binding to the real decision variables, which results in a too large search space to be solvable. To the contrary, the MCF model has the advantage that time can still be modelled as a continous variable and thus reduce the search space.

3.2.6 Pros and cons with the model

Constraint programming is easy to start with, especially for programmers. Further the models are close to the problems, meaning that it in general should be easy to assess the relevans and fit of constraint programs. Also, constraint programming is open for defining and integrating custom search prcedures. However, CP is poor at finding optimal solution, or more generally, in optimization. CP is mainly geared towards discrete variables and finite domains, which in some applications can be too restrictive.

If, however, one could use a propagation and pruning algorithm that are complete (i.e. the algorithm guarantees that there always exists at least one ground solution left in the store in each iteration step) then there is a large advantage in that the constraint programming approach can be used incrementally in time, adding new tasks as the arrive in time. This is however not a common property of the regular constraint programming systems, most pruning and propagation algorithms are incomplete with respect to this, and thus we always have to run the search algorithm until we reach a ground solution.

4 A Set Cover Formluation

In this section, we formulate the TIMBER case as a Set Cover (SC) and focus on finding a plan for the full TIMBER dataset. The problem is much more complex than the problem reported on in the first WP4 report. Some of the sources of the complexity in the present case with TIMBER are:

1. From any place where a vehicle is empty (at a base, or at a factory after having delivered timber), there is a large set of possible stores from where to begin the next task.
2. tasks can be made in any order
3. there are time windows spanning over several days

4. multiple bases and multiple vehicles at every base

5. vehicles must return to base every day

In the rest of this section, we shall first briefly review the solution strategy and characterize the data. Then we go in Section 4.3 and form a Basic model. In Sections 4.4 through 4.6 we define heuristics and solve for a solution of the whole TIMBER dataset. The results and the quality of the solution is discussed in Section 4.7

4.1 Solution Strategy

In the SC approach, there is a number of tasks (transport assignments) that have to be covered by a set of tours. We aim at finding a set of tours S covering the task such that the cost of performing the tours in S is minimal (with respect to the cost of all possible covering sets of tours). As is common in SC approaches, the problem is divided into a tour-generation phase and a solution phase.

The solution phase consists in solving a MIP problem. Among the critical aspect the solution phase is to keep the number of decision variables low. For a comparison, with the implementation of the SC model in the DOIT project, problems as large as 100.000 decision variables can be seen as feasible (ca 10 minutes of computation on a reasonable priced laptop).

In our case, a more critical point is the huge number of possible tours. We will focus the efforts on finding heuristics for restricting the number of feasible tours. The number of tours directly affects both the time it takes for the generation phase to terminate and the number of decision variables for the MIP model in the solution phase.

The cost of using heuristics is that we often trade a few percent points off the true optimium for better execution efficiency and solvability. However, heuristics can be tuned so that we get a solution that is close enough to the optimal one. This is often acceptable in industrial applications. Tuning and selection of heuristics is part of the craftmanship in optimization. Another approach to tackle large sets of tours in the SC formulation, is to use the so-called column generatation tchnique. We will not cover that in this report.

4.2 Data and concepts

The dataset identifies places which we categorize as *bases, factories*, and *stores*. The data also identifies a number of *bases* and the vehicles that belong to each of the bases. From the data, we have constructed 3071 tasks where a task is to transport timber from a store to a factory. The 3071 tasks are unevenly distributed over 31 days. Next follows basic definitions of the concepts found in the data and used in the modelling.

vehicle	A named vehicle in the data
base	A pair of coordinates where at least one of the vehicles initially is located (The pair of coordinates are called *location*)

store	A pair of coordinates marked in the data as a store
factory	A pair of coordinates marked in the data as a factory where there is a demand for timber
task	A task is defined by store location, factory location, a date, timber type, an the age of the store (which is an indicator of the quality of the timber).

4.3 Basic Model and Solution

In this section we generate tours and build the MIP model without considering heuristics. Further below, we will use heuristics to restrict the basic model.

Definition 1. (Concepts)

1. A *place* is either a base, a store, or a factory
2. A *leg* is pair of places (a, b) such that either of a or b, but not both, is a store. If (a, b) is a leg where a is a store and b is a factory, then (a, b) is said to be *loaded*. If a leg is not loaded, it is said to be *empty*.
3. A *pre-route* is a sequence $r = [(a_0, b_0), (a_1, b_1), \ldots, (a_k, b_k)]$ of legs such that $b_i = a_{i+1}$ for every $i < k$, and such that for no $j \leq k$, neither of a_j and b_j is a base
4. A *route* is a sequence $r = [(a_0, b_0), (a_1, b_1), \ldots, (a_k, b_k)]$ of legs such that $b_i = a_{i+1}$ for every $i < k$ and such that $a_0 = b_k$, a_0 is a base, for $i > 0$, a_i is not a base, and for $j < k$, b_j is not a base. We say that a_0 is the *base of* r.
5. A *tour* is a pair (v, r) where v is a vehicle and r is a route. (Henceforth, we assume that if (v, r) is a tour, then the vehicle v belongs to the base of r.) ■

Definition 2. (Further concepts and notation)

1. *Duration of a leg.* The duration of a leg $\lambda = (a, b)$ is denoted δ_λ and defined by an estimate of the drivning time between the end points of the leg. If b is a factory we increase the duration by 30 minutes to model unloading of timber, and if b is a store, we add 30 minutes to model loading of timber.
2. *Duration of a tour.* The duration of a tour $t = (v, r)$ is the sum of the durations of the legs in the route r. ■

4.3.1 Feasible tours

In Section 4.4 below we give a formal definition of the generation of the feasible tours taking also heuristics into account. In this section, we give an informal characteriztion of the set of feasible tours.

Informally, then, if F is the largest set of tours that can formed from the TIMBER data in accordance with Definition 1. Then the set T of *feasible tours* is the largest subset of F such that whenever $t \in T$:

1. the duration of t does not exceed 16 hours
2. no two consequetive legs in t is of longer duration than 4.5 hours
3. if (a, b) is a leg in t and b is a store, then the store is not empty (vehicles only drive to stores for loading timber)

The size of the set of feasible tours determines the number of decision variables in the MIP model defined in the next subsection. Note that the set of feasible tours is huge, as it is more or less all tours possible to generate. Solving for the optimal solution to the MIP model based on this set would be practically infeasible. After stating the MIP model, we shall in Section 4.4 construct a subset of the feasible tours that is small enough to enable finding a solution in reasonable time.

4.3.2 MIP Model

Let $T = \{0, 1, \ldots, S-1\}$ be the set of feasible tours. (Assuming that S is the size of the set of feasible tours.) Let A be the set of tasks, and assume also that A is enumerated.

We will use the following notation:

Definition 3. (Notation)

$A = \{a_0, \ldots, a_n\}$	the set of tasks
$V = \{v_0, \ldots, v_m\}$	the set of vehicles
$T = \{0, 1, \ldots, S-1\}$	the set of feasible tours.
x_i	decision variable for tour $i \in T$ for all $i < S$, such that $x_i = 1$ if tour i is included in the solution and $x_i = 0$ otherwise
c_i	cost associated with tour $i \in T$, for all $i < S$
a_j	the j-th task.
a_{ji}	boolean constant which is 0 unless the task a_j is covered by tour i (in which case it takes the value 1)
v_{ji}	boolean constant which is 0 unless the vehicle v_j is associated with tour i (in which case it takes the value 1) ■

Definition 4. (MIP model)
Objective function

$$minimize \sum_{i<S} c_i x_i$$

subject to Constraints

$$\sum_{i<S} a_{ji} x_i = 1, \qquad a_j \in A \tag{1}$$

$$\sum_{i<S} v_{ji} x_i \leq 1, \qquad v_j \in V \tag{2}$$

$$x_i \in \{0,1\}, \qquad i < S \tag{3}$$

■

4.4 Heuristics

The VRP is known to be NP-complete. As a consequence, the time for being guaranteed to find an optimal solution is exponential in the number of decision variables. Thus, reducing the number of decision variables exponentially reduces the worst case scenario for the time it tkes to find the optimal solution. The heuristics given below aims at reducing the number of decision varaibles in the MIP problem.

The most important heuristic used in our solution is to ignore equivalent routes and symmetric tours:

Definition 5. (Heuristics (1))

1. Let K be a set of (pre-)routes and define a an equivalene relation on K such that ρ_0 and $\rho_1$are equivalent if, and only if, they contain the same tasks. Then for any set R we write

   $$[\rho] \in R$$

   whenever R contains a (pre-)route that is equivalent to ρ

2. If b is a base and $V(b)$ is a set of vehicles belonging to b, then we use one (1) imaginary vehicle v_b to represent any vehicle in $V(b)$. Thus, we replace (v, ρ) with (v_b, ρ) in the generation of the tours and in the solution of the MIP, whenever $v \in V(b)$ and for any (pre-)route ρ ■

The following heuristics aim at reducing the computation time for generating tours.

Definition 6. (Heuristics (2))

1. *fan-out f* For every factory or base b among the places in the data, define the set $H_f(b)$ of the f closest stores. The complexity of the problem can be reduced by lowering the number f of alternative stores to go to after having unloaded timber at a factory or when leaving the base in the morning.
2. *Duration of a dutyperiod h_d* The number of alternative ways to compose a dutyperiod increases exponentially with the length of it. Consequently, the computation time increases expontentially with the duration of a dutyperiod. After findning a solution with shorter dutyperiods, these can be combined into full working day durations, as it done here. A more sofisticated variant is that of *rolling planning* where the computation for one short dutyperiod is initialized according a solution for an immediately preceding dutyperiod.
3. *Cap on number of task per dutyperiod h_a*. In the TIMBER case some of the tasks were considerably shorter than others. This may lead to fitting to many tasks into one dutyperiod, resulting in a combinatorial explosion. Keeping h_a low (max 5) reduced computation time considerably, while however also reducing the quality of the solution.

We shall assume the following are given from the data:

1. the set A of tasks
2. the set V of vehicles
3. the set P of places with the sorts *base, store, factory*
4. the set L of legs as defined in Def. 1

For the following definitions, we need some further notation:

Definition 7. (Notation)

$A(\rho)$	the number of loaded legs in the (pre-)route ρ
$\rho(a,b)$	the (pre-)route obtained from adding the leg (a,b) to the end of the sequence of legs in ρ. (The notation $\rho(a,b)(c,d)$ means that (c,d) is added to the (pre)-route $\rho(a,b)$)
$(a,b)\rho$	the (pre-)route obtained from adding the leg (a,b) to the beginning of the sequence of legs in ρ.
ρ^{-1}	if ρ is a (pre)-route and (a,b) is the last leg in ρ, then ρ^{-1} is the same as b (in other words, it is the latest place visited by ρ)

$\delta(\rho)$	the duration of the (pre)-route ρ, which is the sum of the durations of the legs in ρ (see Def. 2)

Definition 8. (Generating pre-routes) Let H_f, h_a be heuristics as defined in Def. 6. The set Π of *pre-routes* is defined as the smallest set such that, assuming $\rho \in \Pi$ is a pre-route:

1. any loaded leg belongs to Π (recall that a loaded leg is a leg (a,b) s.t. a is *a* store and b is a factory)
2. if (a,b) is a loaded leg, $(a,b) \notin \rho$, then $\rho(\rho^{-1},a)(a,b) \in \Pi$

   Unless

   (a) $a \notin H_f(\rho^{-1})$
   (b) $[\rho(\rho^{-1},a)(a,b)] \in \Pi$
   (c) $A(\rho) > h_a$
   (d) $\delta(\rho(\rho^{-1},a)(a,b)) > h_d$ unless $\delta(\rho) < h_d$
3. (this is like the previous item, just that the new legs are added to the front of the (pre)-route)
   if (a,b) is a loaded leg, $(a,b) \notin \rho$, then $(a,b)(b,\rho^0)\rho \in Pi$

   Unless

   (a) $\rho^0 \notin H_f(b)$
   (b) $[(a,b)(b,\rho^0)\rho] \in \Pi$
   (c) $A(\rho) > h_a$
   (d) $\delta((a,b)(b,\rho^0)\rho) > h_d$ unless $\delta(\rho) < h_d$

Potentially, the set Π of pre-routes contains a large share of routes that never will end up in a solution. Before defining the MIP model, we should get rid of those. For that, we define the following heuristics, which are common in the literature.

Note first that the coverage matrix is defined as a (m,n)-matrix where m is the number of task and n is the number of covers (that is, here, pre-routes). Cell (i,j) in the coverage matrix contains a 1 if the task i is covered by pre-route j, and 0 otherwise.

Definition 9. (Heuristics (2))

1. Remove all pre-routes ρ in Π for which there is a $\rho' \in \Pi$ such that ρ is either a prefix or a suffix of ρ' (In an implementation, this can and should be done while generating the pre-routes)

2. Form the coverage matrix
   (a) Use the coverage matrix to detect whether there are tasks that are not covered by any pre-route (equivelently: find an all-zero row). If so, there is no solution to the problem. End here.
   (b) Use the coverage matrix to identify tasks that are covered by a unique pre-route (rows with a single 1). Remove the task from the problem; save the corresponding pre-route ρ and remove it from the matrix. Also remove all pre-routes that cover any task covered by ρ (they cannot be a part of the solution since ρ is)
   (c) Use the coverage matrix to identify dominated routes (columns in the coverage matrix) and remove them from Π (a pre-route ρ is dominated if there is another pre-route which cover the same tasks as ρ) ■

For exemplification, in the TIMBER case, let Π be the set of pre-routes obtained from a set of 120 tasks from one day in the data. The heuristics defined in 6 and 9 and applied to Π reduce the number of decision variables from 2.69 millions to merely 14280. That is a factor of 190.[6]

Definition 10. (Feasible set of tours) Let Π be a set of pre-routes that has been reduced by applying the heuristics in 5, 6, and 9. Let V be a set of vehicles. Then we form the set T of *feasible tours* by

1. Let B be the smallest set of imaginary vehicles repesenting the bases in the data
2. for each $\rho \in \Pi$, define, for each $v \in B$, a route $(v, \rho') \in T$ such that ρ' is of the form $(\beta, \rho^0)\rho(\rho^{-1}, \beta)$ where β is the base represented by v ■

Note that the tours by Def. 10 can be shorter in duration than the working day duration W as they are limited in duration by the heuristics h_d (Def. 6). The idea is that we shall formulate the MIP problem based on T in Def 10 and then combine the resulting solution into full workdays. We shall also assign vehicles to the combined full tours. From a complexity point of view this is welcome. However, we loose a bit in quality of the solution.

Assume therefore now that the MIP resulting from using the tours defined by Def 10 has been solved in accordance with Def. 4. Note that Constraint (2) in the model must be altered to:

$$\sum_{i<S} \beta_{ji} x_i \leq h_{\beta_j}, \qquad \beta_j \in B$$

where h_{β_j} is the product of W/h_d and the number of vehicles at the base represented by β_j.

[6]Note that the factor depends on using the heuristics in Def. 5. Without those heuristics, the factor would be much larger.

The solution to the MIP thus consists of a number of routes which cover the tasks as efficiently as possible. We shall construct the full routes that correspond to the full working day (16 hours in the TIMBER case) and distribute the full routes over the set of actual vehicles.

1. combine the routes into groups of W/h_d items so that for every group g, every pre-route in g have the same base (where W is the duration of the working day, and h_d is the heuristically chosen duration of the dutyperiod. These numbers can be chosen so that W/h_d is an integer).

2. For every group g defined above, form a sequencce of legs by concatenating the routes in g

3. for all resulting sequences, form a route by replacing each subsequence $(a, b)(b, c)$ with (a, c) where a is a factory, b is a base, and c is a store. (Figure 19 below show the result of joining shorter routes into full working day routes. Here every line is the concatenation of three shorter routes. The gray bars are empty legs.)

4. Let G be the set of routes defined in the previous item. We claim without proof that, if W/h_d is an integer, then there is a 1-1 mapping γ from the set of vehicles onto G such that for every $v \in V$, if $\gamma(v) = (\beta, \rho)$, then v belongs to the base represented by β. (The proof is trivial.)

4.5 Notes on the Tasks and the Cost Function

The tasks in the data are labelled green, yellow, or red. Green indicate good quality, yellow indicates fair quality timber soon to become low quality, and red indicates poor quality. We deemed that the yellow tasks were more critical than the green ones, and that the red tasks were the least important to fulfill[7]. The transport capacity for one day (in the particular application at hand) is limited to circa 100 tasks and often, the number of tasks in the data for one day exceeded that number.

As an heuristic, we therefore ordered the tasks to ensure that the most critical tasks would be consideed in the solution. As a second heursitic, we also put a cap on how many task to consider per day. The more tasks considered, the better the chance of finding a good solution. However, the complexity of the problem grows rapidly with the number of tasks. Choosing the 100 most critcal tasks led to poor solutions, while increasing the number of tasks considered to 150 led to much better solutions (in terms of the KPI:s as discussed in the next section).

In the SC formulation, considering more tasks than we can find a solution for, lead to failure (since there are then tasks that cannot be covered). We used an imaginary extra vehicle to cover the tasks that couldn't be covered by the vehicles specified in the problem. The cost of that vehicle was set to very high

[7]This judgement can be customized in the model

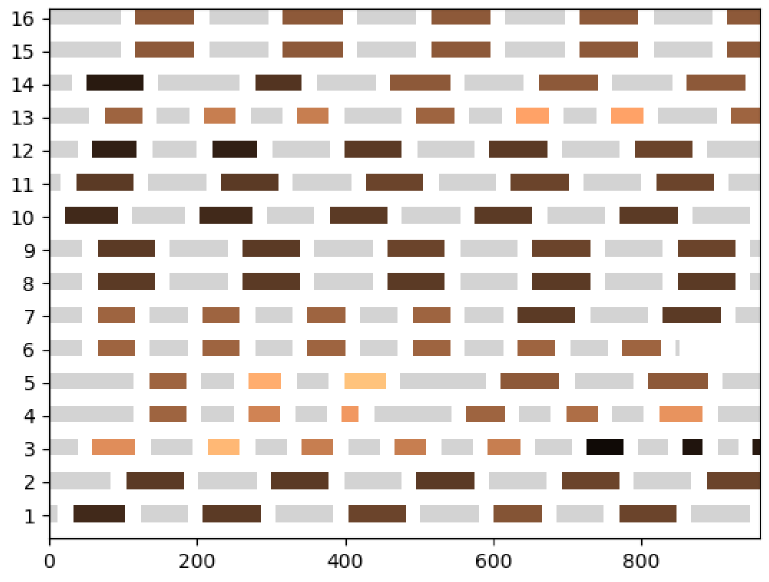


Figure 19: Result after optimizing for one day of tasks. Gray bars are empty legs. Vehicle numbers on the y-axis and minutes on the x-axis.

so that its presence wouldn't interfere with finding the best plan possible for the actual vehicles. The tasks covered by the imaginary vehicle were considered unfinished and subject for further optimization.

Recall that the MIP model defines the objective function (Definition 4) in terms of the cost of the tours. In order to ensure that the yellow tasks become covered in the solution, we penalized both green and red tours. The red tours were penalized the hardest (since the timber in that case was of very low quality and thus of low commercial value). Thus, with the chosen penalty schema, the optimizer prefers yellow tasks before the green before the red in the solution. The level of penalties can be customized in the model.

4.6 Finding a Plan for the Full TIMBER Set

The dataset consists of store status data from 31 days. The data has been transformed into a 3071 tasks (transport assignments). Every task specifies a store, a factory, a status (green, yellow, red), volume to be transported, and the date when the store was created.

It is practically impossible to plan the whole month in one go. Even so, it is also questionable to do so, since a whole month would not be known in advance by the carrier. Instead we plan one day at a time. Next follows the planning strategy:

1. Initially, no task is marked 'done'.
2. At any day D, select the 150 most urgent tasks (i.e. the non-red tasks originating from 150 oldest stores) with a creation date D or prior to D and not yet marked as 'done'. Genererate a set of feasible tours using the defined heursitics and solve the corresponding MIP. Mark the tasks covered by the solution as done. Increase by one day the age of the tasks remaining from day D and any task from days prior to D that haven't been marked 'done'.
3. Repeat (2) until the last day in the data has been processed. End.

4.7 Results

We have succeeded in showing that the SC model can improve the planning performance at TIMBER by 7.2%, also when simulating the day-by-day planning process performed. We only had one dataset consisting of one month, but by planning 31 day instances in a row, feeding the leftover tasks from previous day, the sequence is of fair length, and there is reasons to believe that if the sequence is extended the results would still hold. With further develpment, there is reason to believe that a 10% increase in efficiency is achievable.

An equally important result is that the heuristics defined for the TIMBER case helped to reduce the number of decision variables by a factor of around 200. As noted above, for day 2 in the data the number of decision variables was reduced from 2.69 millions to 14280. This translates into a reduction of computation time with a factor of k^{200} for some $k > 1$ (as a comparison, the estimated number of atoms in the Universe is bounded from above by 2^{100}.) The other days in the data showed similar reductions in complexity.

The computation for the whole dataset took 40 minutes on a reasonbly equipped laptop (16GB RAM, Dual-core i7 28W CPU (Intel Kaby Lake)) using the open-source MIP-solver CBC[8]. Generation of tours was made in Python.

For the quality of the solution, we had access to ground truth in terms of

1. number of transports performed (NTP) (circa 2000)
2. total number of kilometers on road
3. total volume transported
4. (derived) volume per kilometer (VPK)

We also counted the number of critical yellow tasks the model managed to cover (YC). However, we had no access to ground truth for that aspect to compare with. Table 1 shows the potential in improvment resulting from using heuristics developed in DOIT for the TIMBER case.

[8]https://projects.coin-or.org/Cbc

KPI | method	ground truth	SC+heuristics (DOIT)
NTP	ca 2000	ca 2000
VPK	26.5	28.4
YC	n/a	63%

Table 1: Comparing the quality of the solution with ground truth from the dataset

5 Summary

In this report we have showed the potiential of optimization techniques for reducing cost in road bound heavy duty transportation. By modelling data from a real business case, we have illustrated

1. A Constraint Programming Model which is intuitively easy to use and which quickly finds solutions
2. A Minimum Cost Flow Model with very fast performance traded for restrictions in expressivity
3. A Set Cover Model with which we were able to improve on the performance of the real data by 7.2%. Defining and using heuristics we were able to find a near optimal solution within only 40 minutes on a reasonably priced laptop with a free MIP-solver, despite the problem's extreme complexity.

The elaborated illustrations of the optimization techniques is relevent for DOIT in the sense that they clarify and illuminte the type of indata that is critical for the use of such techniques in reducing cost and improving performance in the transport area.