Constraint Models for the Container Pre-Marshaling Problem

Andrea Rendl and Matthias Prandtstetter

AIT Austrian Institute of Technology GmbH
Mobility Department, Dynamic Transportation Systems
Giefinggasse 2, 1210 Vienna, Austria
{matthias.prandtstetter|andrea.rendl}@ait.ac.at

Abstract. The container pre-marshaling problem (PMP) is an important optimization problem that arises in terminals where containers are stacked in bays before they are collected by vessels. However, if a due container is stacked underneath other containers, it is blocked, and additional relocations are necessary to retrieve the container from the bay, which can cause significantly delays. To avoid this effect, the PMP is concerned with finding a minimal number of container movements that result in a container bay without blocked containers.
In this paper, we introduce a Constraint Programming (CP) formulation of the PMP and propose a specialized search heuristic that attempts to try out the most promising moves first. Furthermore, we present a robust variant of the problem that considers the uncertainty of the arrival time of collecting vessels. We show how the robust PMP can be very naturally formulated using CP and give some preliminary results in an experimental evaluation.

1 Introduction

Containers are an essential means for transporting large quantities of goods. The main reason for this is the standardized format of containers through which containers are transportable by all major means of transportation. This is important since transporting goods often requires a chain of different transport modes, such as ship, train or truck. For example, a container that transports goods from a Chinese factory to a European seller, will most likely first travel by train from the factory to a harbor, then by ship to Europe, and then by truck to its final destination.

An important aspect in the transportation chain of containers is the hub between different modes of transport: the *container terminal*. Container terminals handle the exchange of containers between different vessels, as well as the storage of containers until their respective vessel arrives. For illustration, Fig. 1 shows a container terminal for trains and trucks. Since container terminals have to handle an increasingly large amount of incoming and outgoing vessels, it is increasingly difficult to manage all the traffic as well as the container storage within the terminal.

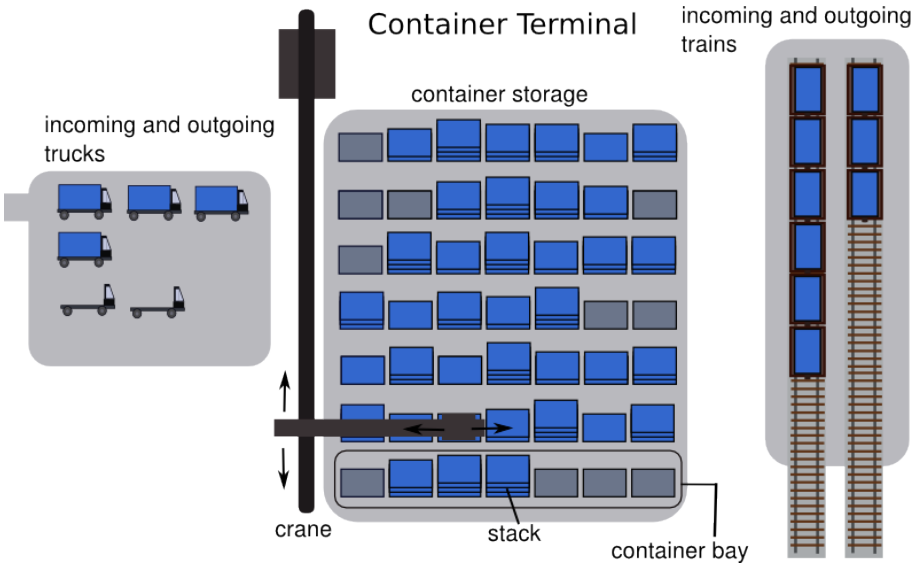

Fig. 1: Container terminal that operates as a hub for trucks (left) and trains (right). Containers are stored in several container bays in the center. A gantry crane moves the containers.

One of the many problems in container terminal optimization is the management of *container bays*, in which incoming containers are stored until they are due for another vessel. A container bay consists of a row of adjacent *stacks* in which containers are stacked upon arrival. When a container is due, it is removed from the bay by the gantry crane. However, due containers are often blocked by other containers in the stack, which results in additional re-locations to access the container. This can significantly increase the loading time of a vessel and cause dramatic (chains of) delays, resulting in displeased terminal clients. Therefore, terminals *pre-marshal* their container bays by re-locating containers to free all blocked containers before incoming vessels arrive. Thus, all containers in the pre-marshaled bay are directly accessible at their due date.

1.1 Related Work

Pre-marshaling is known to be NP-hard [6] and different solution approaches have been proposed for tackling it. Among them are dynamic programming [7, 12], corridor method based approaches [7], neighborhood search based methods [10], greedy heuristics [9], an A*-search [9], a tree-search based approach [5], a multi-commodity flow formulation [11] as well as an integer linear programming approach [11]. However, to the best of our knowledge, no Constraint Programming (CP) approach has been proposed so far for the PMP. In this paper, we show how the PMP can be formulated and solved using Constraint Programming.

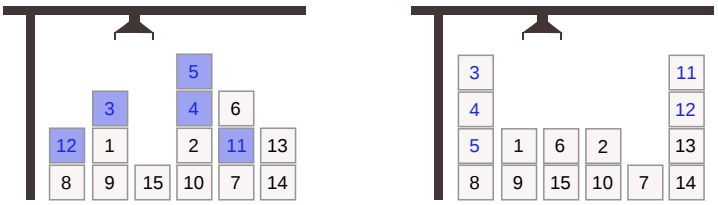

Fig. 2: Container bay before (a) and after (b) pre-marshaling. Shaded containers are blocking other containers; lower numbers indicate earlier due dates.

The PMP is only one of several optimization problems arising in container terminals; we refer to [15–17] for an overview. Furthermore, the PMP is closely related to the famous Blocks World Problem (BWP) [14] where the objective is to relocate an initial configuration of blocks into a specific goal configuration. We want to highlight, however, the differences between the PMP and the BWP [13]: unlike the BWP, in the PMP the number of stacks, as well as their height, is limited. Furthermore, in the BWP, the objective is to arrange the blocks in a pre-specified configuration, while in the PMP, the final configuration is unknown, but simply characterized by some features.

2 The Pre-marshaling Problem (PMP)

In container terminals, containers are stored in *bays* that consist of a number of stacks of maximal height. Every container has a due date at which a vessel will arrive to collect it. The due time of a container c is typically represented by *priority* p_c. The smaller the value of p_c, the sooner c is due for delivery. For instance, container with priority $p_c = 3$ is the third container that will be removed from the bay. Figure 2(a) shows a container bay with 6 stacks of maximal height 4 that holds 15 containers that are labeled with their priority.

The priority of a container is often not known at its arrival in the bay, therefore containers are often stacked randomly. As a consequence, due containers are often blocked by other containers that are due at a later time. In this case, the gantry crane has to perform additional relocations, so-called idle strokes, to 'free' the blocked container. This can result in a severe overhead in processing time which can lead to significant (chains of) delays within the terminal. Figure 2 (a) illustrates a bay with five blocking containers.

The pre-marshaling problem (PMP) is concerned with re-locating containers in a bay such that every container can be removed at its due date without any further relocations. More specifically, the PMP deals with finding a minimal number of container movements that result in a container bay without blocking containers. Figure 2(b) shows the result of pre-marshaling the container bay from Figure 2(a).

2.1 Formal problem formulation

We consider a container bay with S stacks of maximal height H where a set of containers $\mathcal{C}$ are stored. Each container $c \in \mathcal{C}$ is assigned a priority $p_c \in \mathbb{N}$ that indicates when the container will be transported from the bay: the smaller p_c, the earlier the container will be stowed into a vessel. If a container c is stored on top of a container that has a lower priority than c, then c is called a *blocking container*. The number of containers with priority c (multiplicity of c) is given by $\mu(c)$.

The initial bay setup is denoted $\mathcal{B}$ where $\mathcal{B}_{s,t}$ is the t-th container in the s-th stack, with $(s,t) \in \mathcal{S} \times \mathcal{T}$ and write, $\mathcal{B}_{s,t} = 0$ if slot $\mathcal{B}_{s,t}$ is empty. A stack $\mathcal{B}_s, s \in \mathcal{S}$ is said to be *valid* iff

$$\mu_s(c) \leq \mu(c), \qquad \text{for } c \in \mathcal{C}, \tag{1}$$

$$\mathcal{B}_{s,t} = 0 \Rightarrow \mathcal{B}_{s,t+1} = 0, \qquad \text{for } t \in \mathcal{T} \setminus \{H\} \tag{2}$$

Thus, a stack is valid if it does not contain more containers of priority c than are available (Eq. 1) and if a slot at position t is empty, all subsequent slots must also be. A stack $\mathcal{B}_s, s \in \mathcal{S}$ is called *perfect*, if it is valid and

$$\mathcal{B}_{s,t} \geq \mathcal{B}_{s,t+1}, \qquad \text{for } t \in \mathcal{T} \setminus \{H\} \tag{3}$$

holds, i.e. no container in the stack lies underneath a container with higher priority (no containers are blocked). Furthermore, container bay $\mathcal{B}$ is valid, iff

$$\mathcal{B}_s \text{ is valid}, \qquad \text{for } s \in \mathcal{S} \tag{4}$$

$$\sum_{s \in \mathcal{S}} \mu_s(c) = \mu(c), \qquad \text{for } c \in \mathcal{C} \tag{5}$$

thus, if each stack in $\mathcal{B}$ is valid (Eq. 4) and for every priority $c \in \mathcal{C}$, the bay contains exactly the number of corresponding containers (Eq. 5). A bay is called *perfect* if all stacks are perfect.

The bay can be altered by performing particular moves (actions): the topmost container from one stack can be moved to the top of another stack. Therefore, we define the container relocation $r = (i,j)$ as the movement of the topmost container of stack $\mathcal{B}_i$ to the top of stack $\mathcal{B}_j$. A move is *valid* iff $\mathcal{B}_{i,1} \neq 0$ and $\mathcal{B}_{j,H} = 0$, i.e., at least one container is stored in the i-th stack and the number of containers in the j-th stack is less than the maximum height.

The aim of the PMP is to find a sequence of moves that transforms the initial bay $\mathcal{B}$ into a bay without blocking containers. Thus, $\sigma = (r_1, \ldots, r_K)$ is a solution to the PMP if every move $r \in \sigma$ is valid and σ transforms $\mathcal{B}$ into a perfect bay. A solution $\sigma^* = (r_1, \ldots, r_{K^*})$ is said to be optimal if $K^* \leq K$ for all valid solutions σ. Let us denote by $\mathcal{K} = \{1, \ldots, K\}$ the set of steps in a solution and by $\mathcal{K}_0 = \mathcal{K} \cup \{0\}$.

3 A Constraint Model for the PMP

We can easily solve the PMP via Constraint Programming (CP) by iteratively trying to find a solution with exactly $k \geq 0$ container moves, as outlined in

Algorithm 1: PMP(bay layout $\mathcal{B}$)

Input: initial bay layout $\mathcal{B}$

```
1 k ← getLowerBound(B);
2 while true do
3 |   σ ← solveDecisionProblem(k, B);
4 |   if σ ≠ NIL then
5 |   |   return solution;
6 |   k ← k + 1;
```

Alg. 1. When starting with a lower bound for k and iteratively increasing k if no solution could be found, the first returned solution provides an optimal solution for the original PMP formulation. If the lower bound on k is tight, the number of iterations is kept low. Therefore, we utilize the lower bound computation methods proposed by Bortfeldt and Forster [5] deriving the lower bound according to different features of the container bay, such as the number of blocking containers.

Our constraint model is based on an AI planning perspective [3, 2], where we consider *states* that can be altered by a limited set of *actions* (moves). Starting with an initial state (the initial bay layout), we search for a sequence of moves, until a desired end state (perfect bay) is reached. In our model, the main decision variables concern the container bay states.

3.1 Variables

We use two kinds of variables to represent the PMP. First, variables $bay^k_{s,t}$ represent the bay state after performing move k, with $k \in \mathcal{K}_0$, $1 \leq s \leq W$, and $1 \leq t \leq H$. All variables $bay^k_{s,t}$ range over the domain $\mathcal{C}_0$ (set of containers including the empty slot). Thus, $bay^K_{s,t}$ represents the final bay state and $bay^0_{s,t}$ corresponds to the initial layout.

Second, we introduce 0-1-variables $move^k_{s,t}$ that indicate whether a container is moved to the t-th tier of the s-th stack during move k (with $1 \leq s \leq W$, $1 \leq t \leq H$, and $k \in \mathcal{K}$).

3.2 Constraints

First, we assign the initial state of the bay, $\mathcal{B}$, to the variables bay^0:

$$bay^0_{s,t} = \mathcal{B}[s][t] \qquad \text{for } (s,t) \in \mathcal{S} \times \mathcal{T} \tag{6}$$

Next we state that no container may disappear or appear twice in a bay after each move k. More specifically, we state that each priority c must occur as often as its multiplicity $\mu(c)$:

$$\left|\left\{ bay^k_{s,t} : bay^k_{s,t} = c \right\}\right| = \mu(c) \qquad \text{for } c \in \mathcal{C}_0, k \in \mathcal{K} \tag{7}$$

We implemented Constraint (7) using multiple occurrence constraints. Furthermore, the multiplicity of the empty slot is $\mu(0) = S*H-\hat{c}$ where $\hat{c}$ is the number of containers in the bay. Constraints (8) ensure that unchanged slots stay the same: all slots that are neither '0' at steps $k-1$ and k have obviously not been changed and must stay the same:

$$bay_{s,t}^{k-1} \neq 0 \wedge bay_{s,t}^{k} \neq 0 \Rightarrow bay_{s,t}^{k-1} = bay_{s,t}^{k} \quad \text{for } (s,t) \in \mathcal{S} \times \mathcal{T}, k \in \mathcal{K} \tag{8}$$

Since expressing Constraints (8) with standard available (global) constraints resulted in considerably more, additional variables (and constraints), we decided to implement a user defined constraint which straightforwardly checks whether the Constraints (8) are fulfilled. Furthermore, we state that only one container may be moved in each step k:

$$\sum_{(s,t)\in\mathcal{S}\times\mathcal{T}} move_{s,t}^{k} = 1 \qquad \text{for } k \in \mathcal{K} \tag{9}$$

and we link the *bay* variables with the *move* variables, i.e. we ensure that the corresponding *move* variable is set when a container is moved to a (new) position in step k.

$$move_{s,t}^{k} \leq bay_{s,t}^{k} \qquad \text{for } (s,t) \in \mathcal{S} \times \mathcal{T}, k \in \mathcal{K} \tag{10}$$

$$move_{s,t}^{k} \leq 1 - \min\left\{1, bay_{s,t}^{k-1}\right\} \qquad \text{for } (s,t) \in \mathcal{S} \times \mathcal{T}, k \in \mathcal{K} \tag{11}$$

$$move_{s,t}^{k} \geq \min\left\{1, bay_{s,t}^{k}\right\} - bay_{s,t}^{k-1} \qquad \text{for } (s,t) \in \mathcal{S} \times \mathcal{T}, k \in \mathcal{K} \tag{12}$$

$$\tag{13}$$

Finally, we specify the perfect bay setup by stating that priority of slot t in stack s has to be less or equal to the priority in slot $t-1$.

$$bay_{s,t}^{K} \leq bay_{s,t-1}^{K} \qquad \text{for } (s,t) \in \mathcal{S} \times \mathcal{T} \setminus \{1\} \tag{14}$$

Note that this is feasible since we represent empty slots by '0'.

3.3 A Specialized Search Heuristic

Variable Selection Search is only performed on the *bay* variables; the *move* variables are set by the constraints. We apply a (semi) static variable ordering assuring that all variables bay^k for step k are instantiated before variables for step $k' > k$ are selected. This way we search the bay states step by step. However, since the value selection (that determines which container to move within the bay) is dynamic, the variable order *within* one step k is determined heuristically, as explained below.

Value Selection A smart value selection heuristic requires some knowledge about which container move would be beneficial in the current bay state. Therefore, we employ the heuristic of Bortfeldt and Forster [5] that, given a bay layout, returns a sorted list of promising moves $\mathcal{M}$, where the potentially best moves are first in the list. The list $\mathcal{M}$ is calculated by classifying moves according to how they improve or impair the current bay state. For instance, a move that renders an imperfect stack *perfect*, is called a *bad-good* move, since it transforms a bad into a good state. In the same fashion, *good-bad*, *good-good* and *bad-bad* moves are defined. Note, that moves that worsen the bay state, *good-bad* and *bad-bad*, are often essential to 'dig out' containers and thus reach a solution.

In summary, we compute $\mathcal{M}$ for the current state at step $k \in \mathcal{K}$, and try out every move $m \in \mathcal{M}$, starting with the first (the most promising one). Then, the selected move m determines which state variable to search on next, as well as its value. More specifically, we set the state variable of the target position of the moved container with the container value. This choice sets all other state variables, which is essential since therefore mainly *one* variable-value choice is needed to reach the next step $k + 1$.

4 A robust variant of the PMP

In real-world settings, the exact arrival time of a vessel is quite uncertain. In fact, most vessels are expected to arrive within a time window instead of at an exact time. Since the container priorities in the classical PMP are based on the scheduled arrival *times* instead of *time windows* of vessels (and those time windows often overlap), the *initially expected* priority of a container may differ from the *actual* priority. Thus, perfect bays that are obtained through the classical PMP approach are easily rendered imperfect, if the arrival time windows of vessels are not considered. We therefore aim at finding a sequence of container moves that produces a final bay setup that is *robust* concerning expected vessel delays.

The (expected) arrival time of a vessel can be represented by some probability distribution, for instance a normal distribution with the scheduled arrival time as mean and the expected delay as standard deviation. However, we simplify this notion by considering the arrival time as a simple time window (where the arrival at any time within the time window has the same probability and thus represents a uniform distribution). Based on the arrival time window, we can deduce a *priority range* for the container that will be collected. More specifically, a container c has a *priority range* $p_c = \{a_l, a_u\}$ where a_l is the earliest possible priority, and a_u is the latest. For instance, a priority range of $p_c = \{4, 6\}$ denotes that container c may be the fourth, fifth or sixth container to be removed from the container bay. Figure 3 illustrates a bay where for some containers the priorities are given as ranges.

A stack $\mathcal{B}_s, s \in \mathcal{S}$ is called *robust*, if it is valid and

$$p_l^{\mathcal{B}_{s,t}} \geq p_u^{\mathcal{B}_{s,t+1}}, \qquad \text{for } t \in \mathcal{T} \setminus \{H\} \tag{15}$$

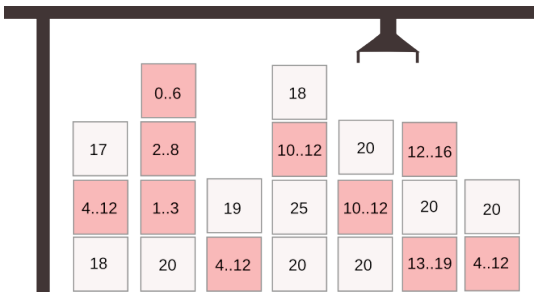

Fig. 3: Container bay where the priorities of some containers are ranges

holds, i.e. the upper bound of the priority of the container at $\mathcal{B}_{s,t+1}$ that is stacked upon container $\mathcal{B}_{s,t}$, has to be smaller or equal to the lower bound of container c_2. We say that the container bay $\mathcal{B}$ is robust, iff it is valid and all stacks are robust.

Please note, that a given container bay with priority ranges may not have a robust configuration. For instance, consider a 4×4 bay with 5 containers that all have the same priority range 0..5. This bay cannot be altered to a robust bay configuration, since in some stack, two containers with range $\{0..5\}$ have to be stacked over another. Therefore, it can be useful to determine beforehand if a robust configuration exists, which can be easily formulated using CP. This is part of our current work.

To the best of knowledge, very little has been investigated into studying robust or stochastic variants of the pre-marshaling problem. In very recent work, Borjan et.al. [4] present a mathematical model for the dynamic version of the Container Relocation Problem (CRP), and also consider uncertainty. The CRP is very similar to the PMP, where the goal is to additionally empty the bay while performing relocations.

5 A Constraint Model for the Robust PMP

The constraint model for the robust PMP is based on the PMP model from Sec. 3. The main difference lies within the notion of container priorities: in the robust PMP, we consider container priorities as range $\{lb..ub\}$ where $lb < ub$ and lb denotes the lower bound of the container's priority, and ub the upper bound.

Furthermore, we assign a unique id $c \in \{1, \ldots, C\}$ to each container. This is necessary in order to track containers to assure that no container occurs multiple times or disappears during a move. More specifically, in the PMP model (Sec. 3), we were able to assure that all containers stay in the bay after each move by checking the multiplicity of each priority (Constraint (7)). However, in the robust PMP, priorities are ranges and considering the multiplicity of ranges is much more expensive. Therefore, we assign a unique id to each container to enhance modeling.

The priority range for container c is denoted by p_i^c with $i \in \{l, u\}$ where p_l^c is the lower bound, and p_u^c is the upper bound. Note, that the empty slot '0' has priority $p_l^0 = p_u^0 = 0$ and containers c with a single priority value v have the same upper and lower bound $p_l^c = p_u^c = v$.

5.1 Variables

We apply the same set of variables as for the classical PMP (see Section 3.1). However, the meaning of the *bay* variables is slightly different: while in the classical PMP model the domain $\mathcal{C}$ of variables *bay* represented the container priorities, in the robust PMP its domain represents the container *id*.

5.2 Constraints

The constraints for the robust PMP are the same as for the classical PMP, with the exception of the final bay configuration. Therefore, after stating constraints (6)-(12), we constrain the final bay setup by:

$$p_u^{bay^K_{s,t}} \leq p_l^{bay^K_{s,t-1}} \qquad \text{for } (s,t) \in \mathcal{S} \times \mathcal{T} \setminus \{1\} \tag{16}$$

More specifically, Constraint (16) states that the upper bound of the priority range of the container in slot t has to be less or equal than the lower bound of the container in slot $t-1$. This is necessary to avoid an overlap within the container priority range of two stacked containers. For instance, container c_1 with priority range $p^{c_1} = \{0..4\}$ may not be stacked upon container c_2 with $p^{c_2} = \{2..9\}$ since the ranges overlap: $\{0..4\} \cap \{2..9\} = \{2..4\}$. Therefore, if for instance, c_1's priority is realized with 3 and c_2 with 2, then c_1 would block c_2.

6 Preliminary Results

We conducted a computational evaluation to assess the performance of the presented CP models. Note that the CP models were implemented in Java 7 using the freely available Choco framework [8] version 2.1.5 as CP solver. We first discuss results for the PMP model and then continue with the robust PMP model.

6.1 Evaluating the CP Model for the PMP

We compare the PMP model from Sec. 3 to one of the fastest current approaches [12]: a dynamic program combined with branch-and-bound, denoted DPBnB. Each approach is given a maximum of 600s computation time to find an optimal solution for the test instances.

As test instances, we decided to rely on the instance generator provided by Expósito-Izquierdo et al. [9] via [1]. Unfortunately, the original instances used in [9] are not publicly available. The instances are grouped into sets according

		DPBnB		CP	
		#solved	time[s]	#solved	time[s]
04 x 04 050	001	100	0.03	100	1.79
	002	100	0.02	100	0.55
	003	100	0.02	100	0.49
	004	100	0.02	100	0.43
04 x 04 075	001	100	0.12	–	–
	002	100	0.10	51	117.41
	003	100	0.06	87	46.96
	004	100	0.06	85	26.16

Table 1: Results for solving the PMP on instances with 4 stacks of maximal height 4 with either 50% or 75% of container coverage and 4 difficulty levels (001 is the most difficult level).

to the *bay size* (4×4 stands for 4 stacks of maximal height 4), the *utilization rate* (either 50% or 75% of the available tiers are filled with containers), and the *difficulty level* (1 to 4) that is a rough estimation on how disordered the bays are (with 1 being the hardest). Each of these sets contains 100 (pairwise different) instances.

The experiments were carried out on a single core of an Intel® Core™2 Quad CPU Q9300 with 2.50GHz and 3GB RAM.

Table 1 documents how often each algorithmic setup reached an optimal solution, as well as the average computation times (over all 100 instances). Columns report the number of instances solved to optimality and the average computation time for each approach; rows represent the set of 100 instances with respect to their size, coverage rate, and difficulty level.

We observe that the CP model is not yet competitive with the current state of the art, in particular on dense instances, where the container bay contains a lot (75%) of containers. While for less dense instances (50% coverage), the CP model can solve all instances, it only manages to solve 85% of the easiest dense instances, and for the most difficult level, cannot solve a single instances within the given time limit.

We believe that one major drawback of the current CP formulation is the lack of a mechanism (in form of a constraint or search heuristic) that stops search on bay states that have already been reached (and thus should not be part of an optimal solution). This is difficult to formulate effectively as a constraint in CP. However, since the DPBnB approach is able to eliminate these bay states naturally, we are currently working on an integration of this approach into our CP model.

6.2 Evaluating the CP Model for the Robust PMP

We conducted some first tests on the robust PMP model from Sec. 5. The instances for the robust PMP are based on the instances for the PMP that we

instance	# moves	time [s]	nodes	backtracks
0	5	0.186	41	66
1	6	0.625	474	933
2	5	0.222	66	116
3	5	0.125	41	51
4	5	0.938	1301	2561

Table 2: Preliminary results for the robust PMP on instances with 50% coverage and medium difficulty level.

modify: we first randomly assign each container to a vessel (train, truck or ship) and assign each vessel an arrival time that is reflected by the priority of the container in the PMP instance. Then we deduce each container's priority range by the vessels' arrival time window. More specifically, we create an arrival time window for each vessel that is based on a distribution of expected delays.

The experiments were carried out on a single core of an Intel® Core™ i7 CPU M 640 with 2.80GHz and 1GB RAM.

The preliminary results are summarized in Tab. 2 that shows the number of moves to render the bay perfect, the solving time, as well as some information on search. We see that we are able to solve these rather small instances quickly, however, we still need to assess our approach on larger and more complex problem instances.

7 Conclusions

In this paper, we introduced a new problem to the Constraint Programming community, the container pre-marshaling problem (PMP). The PMP is concerned with finding a minimal sequence of container relocations such that the resulting container bay hosts no blocked containers. Our contributions are two-fold: first, we present the first constraint model for the PMP and second, we introduce a robust variant of the PMP and show how the CP model for the PMP can be easily and naturally extended to the robust formulation.

In an initial experimental evaluation, we assess both constraint models. We observe that the PMP model is still not competitive and requires enhancement. In particular, we compare the CP model to a dynamic programming based approach that represents the current state of the art. Our current (and future) work is focused on improving the constraint model by integrating features of the mentioned DP-based approach into the CP model. Some initial experiments on the robust PMP (for which there are no alternative approaches yet) show good results on small instances.

For future work, we plan to enhance and extend both our CP models to render them competitive with existing approaches. Furthermore, we want to consider alternative formulations and explore alternative solving approaches for tackling the robust PMP.

Acknowledgments

This work is part of the project TRIUMPH, partially funded by the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT) within the strategic programme I2VSplus under grant 831736. The authors thankfully acknowledge the TRIUMPH project partners Logistikum Steyr (FH OÖ Forschungs & Entwicklungs GmbH), Ennshafen OÖ GmbH and via donau – Österreichische Wasserstraßen-Gesellschaft mbH.

References

1. `http://sites.google.com/site/gciports` (last accessed April 2013)
2. Barták, R., Toropila, D.: Reformulating constraint models for classical planning. In: Wilson, D., Lane, H.C. (eds.) FLAIRS Conference. pp. 525–530. AAAI Press (2008)
3. van Beek, P., Chen, X.: Cplan: A constraint programming approach to planning. In: Hendler, J., Subramanian, D. (eds.) AAAI/IAAI. pp. 585–590. AAAI Press / The MIT Press (1999)
4. Borjan, S., Manshadi, V., Barnhart, C., Jaillet, P.: Dynamic stochastic optimization of relocations in container terminals. working paper, submitted, MIT (2013), http://web.mit.edu/jaillet/www/general/container13.pdf
5. Bortfeldt, A., Forster, F.: A tree search procedure for the container pre-marshalling problem. European Journal of Operational Research 217(3), 531–540 (2012)
6. Caserta, M., Schwarze, S., Voß, S.: Container rehandling at maritime container terminals. In: Böse, J.W. (ed.) Handbook of Terminal Planning, Operations Research/Computer Science Interfaces Series, vol. 49, pp. 247–269. Springer New York (2011)
7. Caserta, M., Voß, S.: A corridor method-based algorithm for the pre-marshalling problem. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G., Ekárt, A., Esparcia-Alcázar, A., Farooq, M., Fink, A., Machado, P. (eds.) Applications of Evolutionary Computing, Lecture Notes in Computer Science, vol. 5484, pp. 788–797. Springer (2009)
8. choco Team: choco: an Open Source Java Constraint Programming Library. Research report 10-02-INFO, École des Mines de Nantes (2010), http://www.emn.fr/z-info/choco-solver/pdf/choco-presentation.pdf
9. Expósito-Izquierdo, C., Melián-Batista, B., Moreno-Vega, M.: Pre-marshalling problem: Heuristic solution method and instances generator. Expert Systems with Applications 39(9), 8337–8349 (2012)
10. Lee, Y., Chao, S.L.: A neighborhood search heuristic for pre-marshalling export containers. European Journal of Operational Research 196(2), 468–475 (2009)
11. Lee, Y., Hsu, N.Y.: An optimization model for the container pre-marshalling problem. Computers & Operations Research 34(11), 3295–3313 (2007)
12. Prandtstetter, M.: A dynamic programming based branch-and-bound algorithm for the container pre-marshalling problem. Tech. rep., AIT Austrian Institute of Technology (2013), submitted to European Journal of Operational Research
13. Rodríguez-Molins, M., Salido, M.A., Barber, F.: Intelligent planning for allocating containers in maritime terminals. Expert Syst. Appl. 39(1), 978–989 (2012)
14. Slaney, J.K., Thiébaux, S.: Blocks world revisited. Artif. Intell. 125(1-2), 119–153 (2001)

15. Stahlbock, R., Voß, S.: Operations research at container terminals: a literature update. OR Spectrum 30, 1–52 (2008)
16. Steenken, D., Voß, S., Stahlbock, R.: Container terminal operation and operations research - a classification and literature review. OR Spectrum 26, 3–49 (2004)
17. Vis, I.F., de Koster, R.: Transshipment of containers at a container terminal: An overview. European Journal of Operational Research 147(1), 1–16 (2003)