An Algorithm Selection Benchmark of the Container Pre-Marshalling Problem (Extended version)

Kevin Tierney[1] and Yuri Malitsky[2]

[1] Decision Support & Operations Research Lab, University of Paderborn, Germany
tierney@dsor.de
[2] IBM T.J Watson Research Center, USA
ymalits@us.ibm.com

Abstract. We present an algorithm selection benchmark based off of optimal search algorithms for solving the container pre-marshalling problem (CPMP), an NP-hard problem from the field of container terminal optimization. Novel features are introduced and then systematically expanded through the recently proposed approach of latent feature analysis. The CPMP benchmark is interesting, as it involves a homogeneous set of parameterized algorithms that nonetheless result in a diverse range of performances. We present computational results using a state-of-the-art portfolio technique, thus providing a baseline for the benchmark.

1 Introduction

The container pre-marshalling problem (CPMP) is a well-known, NP-hard problem in the container terminals literature [7, 2, 6] that was first introduced in [5]. In the CPMP, stacks of containers (called a *bay*) at a container terminal are sorted by a crane such that containers that must leave the stacks first are placed on top of containers that must leave the stacks later. This prevents *mis-overlaid* containers from blocking the timely exit of other containers. A crane moving the containers can only access the top container on each stack, each stack has a maximum height, and the maximum number of stacks is fixed. The goal of the CPMP is to find the minimal number of container movements necessary to ensure that all of the stacks are sorted by the exit time of each container from the stacks. Solving the CPMP assists container terminals in reducing delays and increasing the efficiency of their operations.

A recent approach for solving the CPMP to optimality [8] presents two state-of-the-art approaches, based on A* and an IDA*, that we use to form a benchmark for algorithm selection. We introduce 22 novel features to describe CPMP instances and show how the approach of latent feature analysis (LFA) [**?**] can assist domain experts in developing useful features for algorithm selection approaches. Finally, we augment the existing CPMP instances with instances from a new instance generator.

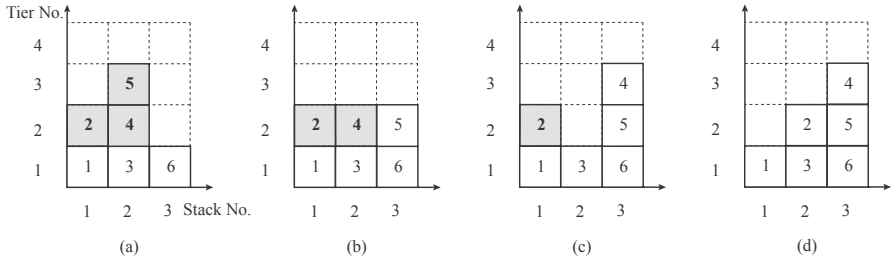

Fig. 1: An example solution to the CPMP with mis-overlays highlighted. From [8].

2 The container pre-marshalling problem

Given an initial layout of a bay, the goal of the CPMP is to find the minimal number of container movements (or *rehandles*) necessary to eliminate all mis-overlays in the bay. Formally, a bay contains S stacks which are at most T tiers high. Each container in the bay is assigned a priority, $p_{st} \in \mathbb{N}_0$ ($s \in S, t \in T$). We set $p_{st} = 0$ if there is no container at the position s, t. Containers with a smaller priority value are retrieved first, meaning they must be above containers with a larger priority value in a configuration with no mis-overlays. Thus, a bay has no mis-overlays iff $p_{st} \leq p_{s,t+1}$ for all $s \in S$, $1 \leq t < |T|$. As previously mentioned we focus on a single bay, thus no movements between bays are allowed and all containers are assumed to be of the same size.

Consider the simple example of Figure 1, which shows a bay composed of three container stacks where containers can be stacked at most four tiers high. Each container is represented by a box with its corresponding priority.[3] This is not an ideal layout as the containers with priority 2, 4 and 5 will need to be relocated in order to retrieve the containers with higher priority (1 and 3). That is, containers with priority 2, 4 and 5 are mis-overlaid. Consider a container movement (f, t) defining the relocation of the container on top of the stack f to the top position of the stack t. The containers in the initial layout of Figure 1 can reach the final layout (d) with three relocation moves: $(2, 3)$ reaching layout (b), $(2, 3)$ reaching layout (c) and $(1, 2)$ reaching layout (d) where no mis-overlays occur.

Pre-marshalling is important both in terms of operational and tactical goals at a container terminal. We refer to [8] for more information and a discussion of related work.

[3] We note that multiple containers may have the same priority, but in order to make containers easily identifiable, in this example we have assigned a different priority to each container.

3 Latent feature analysis (LFA)

Given a set of solvers for a problem and a set of instance, algorithm selection is the study of finding the best performing solver for each instance. There are a variety of approaches that can be used to make this decision, including machine learning techniques as well as scheduling algorithms. For an overview of this area, we refer the reader to a recent survey [**?**]. Regardless of the selection technique employed, the quality of features in differentiating instances is critical to the success or failure of any algorithm selection strategy.

Features are normally created based on the knowledge of domain experts. As an alternative to this unsystematic approach to feature generation, a recent paper [**?**] theorized how latent features gathered from matrix decomposition could help guide researchers to the kinds of structural features they should be looking for. Following the guidelines of the original work, this paper puts those ideas into practice.

The idea proposed by [**?**] motivated the use of Singular Value Decomposition (SVD) to find the latent features that best describe the changes in the actual performance of solvers on instances. SVD is a method for identifying and ordering the dimensions along which data points exhibit the most variation, which is mathematically represented by the following equation: $M = U\Sigma V^T$, where M is the $m \times n$ matrix representing the original data. In our case, we consider an M where there are m instances each described by the performance of n solvers. This means that the $m \times n$ orthonormal columns of U can be interpreted as a latent feature that describes that instance. The columns of the V^T matrix refer to each solver, with each row presenting how active, or important a particular feature is for that solver.

Fundamentally, this decomposition means that if for a given instance it were possible to predict the latent features, we could multiply by the existing Σ and V^T matrices to get back the performance of each solver. While this is of course impossible in practice, we can use an existing set of structural features to predict these latent features. By then studying these predictions, we can identify exactly which latent features are currently difficult to predict accurately and even identify which latent feature we should focus on getting right to maximize the quality of the resulting prediction.

It is assumed that if we are unable to accurately predict a latent feature using our existing features, then our feature set is missing something critical about the underlying structure of an instance. By computing the correct value for this latent feature and sorting all training instances based on it, we assume that there must be something different for the instances where the latent feature is large and those instances where the value is small. It is then up to the researcher to try to analyze this difference and propose a new expanded set of features for the algorithm selection approach to take advantage of.

4 Algorithm selection benchmark

We now describe the optimal algorithms, datasets from the literature and features from two iterations of latent feature analysis that used in our benchmark. We have made this benchmark available in the Algorithm Selection Library (`www.aslib.net`) under the name "PREMARSHALLING-ASTAR-2013".

4.1 Algorithms

We include four algorithms, which are parameterizations of the A* and IDA* approaches in [8]. All four algorithms solve the CPMP to optimality, returning either the number of moves required to sort the bay or proving that the bay is not sortable. We parameterize a symmetry breaking heuristic present in both the A* and IDA* algorithms using either a greater than or less than constraint in a symmetry breaking heuristic. Due to space constraints, we refer interested readers to [8] for algorithm and heuristic details.

An interesting aspect of the pre-marshalling benchmark in relation to other benchmarks, such as those based on SAT, CSP, QBF, etc. is that the portfolio of algorithms is not particularly diverse, but performance variations are nonetheless very large. Most algorithm selection benchmarks consist of either a set of diverse algorithms, or a set of diverse algorithm parameterizations. In the case of the CPMP benchmark, the heuristics used within the IDA* and A* approaches are essentially the same, except for small variations to the way the heuristic functions as mentioned above. This is especially interesting because it opens possibilities for performing algorithm selection in lieu of algorithm configuration with a parameter tuner (e.g., GGA [**?**]).

4.2 Datasets

We divide existing pre-marshalling instances into a training and test set based on well-known, existing data from the literature and generated instances similar to literature instances. We do not include instances in which all four parameterized algorithms timeout/memout, or in which all solvers finish within a second.

In the training set, we include 267 instances from [1] and 260 from [3]. In our test set, we use the instance generator from [4] to make 257 instances[4]. We further created an instance generator to mimic the instances from [1] and [3], allowing for an additional 163 and 127 instances, respectively. We note that our instances mimicking those from [1] are not exactly the same, as the authors do not completely describe their instance generation process. In total, we have 527 training instances and 547 test instances. We note, however, that researchers testing algorithm selection approaches can combine these datasets and use cross-validation. We avoid this in order to show the value of the latent feature analysis process to generate features.

1. Number of stacks
2. Number of tiers
3. Tiers/stacks ratio
4. Container density
5. Empty stack percentage

6,7. Percent of all {slots, stacks} that are mis-overlayed
8. Bortfeldt & Forster lower bound

9–12. Min/max/mean/stdev container priority counts
13–16. Min/max/mean/stdev priority of top non-mis-overlayed container

17. Left container density
18. Tier-weighted groups
19. Largest group L1 distance from top left (average)
20. Percentage contiguous empty space including one empty stack

21. Mis-overlaid stack ($\geq$ 2 containers) (percentage)
22. Low-group containers near stack tops (percentage)

Fig. 2: Features for the CPMP.

4.3 Features

The features used in our dataset are given in Figure 2. We split our features into three categories. The first set of features, 1 through 16, were designed before performing latent feature analysis. Features 17 through 20 were created based on our first iteration of latent feature analysis, and features 21 and 22 using our second iteration. All of the features we propose are easy to compute and take no longer than 0.001 CPU seconds (in total) even on large instances. Our feature generation code (and instance generator) is available at `https://bitbucket.org/eusorpb/cpmp-as`. We now describe the features in more detail.

Original features The first 5 features address the problem size and density of containers. Problems with a high container density are likely to pose different challenges than those where there is lots of empty space. Features 6 and 7 look at the lower bound on the number of moves necessary to fix the bay in two different ways. Feature 6 simply counts the number of mis-overlayed containers, which is an obvious lower bound to the problem. Feature 7 provides the lower bound from [1], which analyzes indirect container movements in addition to the mis-overlays present in feature 7. Features 8 through 11 offer information on how many containers belong to each priority group. Some instances have a one-to-one mapping of groups to containers, whereas other instances have large numbers of containers in particular priority groups. Finally, features 12 through 15 attempt to uncover the structure of the priorities of the top non-mis-overlaid container on each stack. It is possible that low values of this feature could lead to more

[4] We do not use the instances from [4] because the original instances were lost.

difficult problems, or at least problems where more moves are required, as low valued containers will have to trade places with higher valued ones.

LFA iteration 1 features Our first feature added in LFA iteration 1 is the density of containers on the "left" side of the instance. Given S stacks, we define this as stacks 1 through $\#stacks/3$. We note that this value was determined through the analysis, but has no formal basis. This feature is somewhat tuned to the algorithms themselves and may not generalize to other pre-marshalling approaches, as the symmetry breaking heuristics can focus on certain sides of the problem. Feature 18 attempts to measure whether containers with high group values are on high or low tiers. It does this by multiplying the tier of a container by its group, summing these values together and dividing by the maximum this value could take. Feature 19 measures the L1 (manhattan) distance from the top left of a problem to each container in the largest group (i.e., the highest exit time), averaging these distances if there are multiple containers in the group. The final feature from iteration 1 computes the percentage of empty space in the instance in which an area of contiguous empty space includes at least one empty stack. This gives a rough indication of the distribution of containers in stacks across the problem.

LFA iteration 2 features In the second iteration, we determined two additional features. Feature 21, counts how many stacks with more than two containers are mis-overlaid. Finally, feature 22 counts "low" valued containers on the top of stacks, where we define low as being any group less than or equal to the largest group-id (exit time) divided by four. As in feature 17, this is an arbitrary value we determined through the LFA process and has no formal backing.

The features generated through LFA are significantly different than those generated before LFA. A couple concentrate specifically on the left side of the problem, which is clearly in relation to the greater than or less than constraints that play a role in symmetry breaking. Additionally, our LFA features are not the kind of features one would try without some evidence that they might be useful – and a guess-and-check approach would result in many ridiculous features that have little predictive value. We make no claim that this is an exhaustive list of features, indeed there are a number of other possibilities, such as probing features using various heuristics. At the very least, these features offer a good starting point for algorithm selection research into the CPMP.

5 Computational results

We evaluate our features using the cost-sensitive hierarchical clustering (CSHC) approach from [**?**]. Table 1 provides the performances[5] of a CSHC based portfolio when trained on the three datasets versus the best single solver (BSS) and the

[5] All runtime data was generated on an AMD Opteron 2425 HE processor running at 2100 MHz with a one hour timeout.

Solver	Avg.	PAR-10	Solved
BSS	78.6	5,923	458
Original Features	51.6	3,469	495
LFA Iteration 1 Features	46.6	2,741	506
LFA Iteration 2 Features	45.4	2,543	509
VBS	12.8	12.8	547

Table 1: The best single solver (BSS) and the virtual best portfolio (VBS) are provided for comparison.

virtual best solver (VBS), which is a portfolio that always picks the correct solver. CSHC using just the initial arbitrary features already performs significantly better than the BSS, indicating even the original features have some descriptive value. When a CSHC portfolio is trained on the first iteration of features, the performance improves not only in the number of instances solved, but also on the average time taken to solve each instance. This shows that by utilizing the latent feature analysis, a researcher is able to develop a richer set of features to describe the instances. Furthermore, the process can be repeated, as is evidenced by the performance of CSHC on the second iteration of features. Note that the overall performance is again improved not only in the number of instances solved, but the time taken to solve them on average.

6 Conclusion

We presented an algorithm selection benchmark for the container pre-marshalling problem, a well-known problem from the container terminals literature. Our benchmark includes novel features and instances. We further showed that latent feature analysis can help in augmenting problem features. We hope that this benchmark will help further algorithm selection research on real-world problems.

References

1. Bortfeldt, A., Forster, F.: A tree search procedure for the container pre-marshalling problem. European Journal of Operational Research 217(3), 531–540 (2012)
2. Carlo, H., Vis, I., Roodbergen, K.: Storage yard operations in container terminals: Literature overview, trends, and research directions. European Journal of Operational Research 235(2), 412 – 430 (2014)
3. Caserta, M., Voß, S.: A corridor method-based algorithm for the pre-marshalling problem. In: Giacobini, M. et al. (ed.) Applications of Evolutionary Computing. Lecture Notes in Computer Science, vol. 5484, pp. 788–797. Springer (2009)
4. 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)
5. Lee, Y., Hsu, N.: An optimization model for the container pre-marshalling problem. Computers & Operations Research 34(11), 3295–3313 (2007)

6. Lehnfeld, J., Knust, S.: Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. European Journal of Operational Research 239(2), 297 – 312 (2014)
7. Stahlbock, R., Voß, S.: Operations research at container terminals: a literature update. OR Spectrum 30(1), 1–52 (2008)
8. Tierney, K., Pacino, D., Voß, S.: Solving the pre-marshalling problem to optimality with A* and IDA*. Tech. Rep. WP#1401, DS&OR Lab, University of Paderborn (2014)