An Algorithm Selection Benchmark of the Container Pre-Marshalling Problem

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 on 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 and stacking literature [2, 10, 7], first introduced in [6]. The CPMP deals with the sorting of containers in a set of stacks (called a *bay*) of intermodal containers based on their exit times from the stacks, such that containers that must leave the stacks first are placed on top of containers that must leave later. This prevents *mis-overlaid* containers from blocking the timely exit of other containers. 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 without exceeding the maximum height of each stack. 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 [11] presents two state-of-the-art approaches, based on A* and IDA*. We use parameterized versions of these approaches 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) [8] can assist domain experts in developing useful features for algorithm selection approaches. Finally, we augment the existing CPMP instances with extra instances from a new instance generator. [3].

[3] An extended version of this paper is available at https://bitbucket.org/eusorpb/cpmp-as/downloads/asl_pm_extended.pdf.

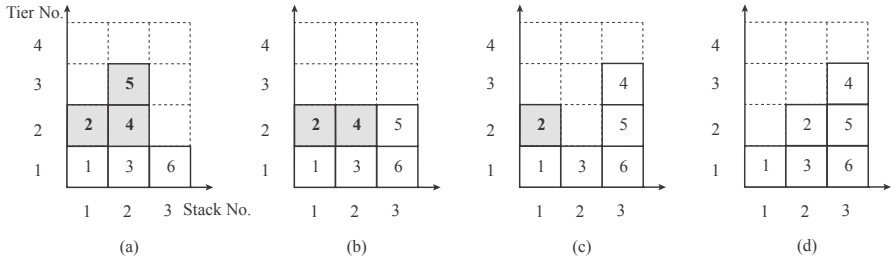

Fig. 1: An example solution to the CPMP with mis-overlays highlighted. (Reproduced from [11]).

2 The container pre-marshalling problem

Given an initial layout of a bay with a fixed number of stacks and tiers (stack height), 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. Every container is assigned a group that indicates when it must leave the bay. A mis-overlaid container is defined as a container with a group that is higher than the group of any container underneath it, or a container above a mis-overlaid container.

Consider the simple example of Figure 1, which shows a bay composed of three stacks of containers in which containers can be stacked at most four tiers high. Each container is represented by a box with its corresponding group.[4] This is not an ideal layout as the containers with groups 2, 4 and 5 will need to be relocated in order to retrieve the containers with higher groups (1 and 3). That is, containers with groups 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. In particular, effective pre-marshalling of containers can help reduce delays moving containers from the terminal yard onto vessels, as well as from the yard onto trucks or trains. We refer to [11] for more information and a discussion of related work.

[4] We note that multiple containers may have the same group, but in order to make containers easily identifiable, in this example we have assigned a different group 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 [5]. Although there are many algorithm selection approaches, they are not the only important component in a selection approach. 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. In [8], the authors theorize how latent features gathered from a matrix decomposition can systematically help domain experts augment a set of features with more effective ones. Specifically, [8] shows that latent features can be determined using an existing set of structural features. Features that assist algorithm selection techniques in making correct predictions can then be identified, thus guiding a domain expert towards the features that work best on his or her problem.

The idea proposed by [8] uses 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 of solver performance. 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.

If for a given instance it were possible to predict the latent features, using this decomposition we could multiply the feature vectors 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 value is large and those instances where the value is small. It is then up to a domain expert to try to analyze this difference and propose a new, expanded set of features for the algorithm selection approach to take advantage of.

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-overlaid

8. Bortfeldt & Forster lower bound

9–12. Min/max/mean/stdev container group counts

13–16. Min/max/mean/stdev group of top non-mis-overlaid container in each stack

17. Container density in stacks 1 through $\#Stacks/3$
18. Tier-weighted groups
19. Largest group L1 distance from top left (average)
20. Pct. 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.

# Insts.	Source
Training set (Total: 527)	
267	BF [1]
260	CV [3]
Testing set (Total: 547)	
257	Expo. (Gen.) [4]
163	BF-like*
127	CV-like

Fig. 3: Instances used.

4 Algorithm selection benchmark

We now describe our benchmark in detail[5]. Four optimal parameterizations of the A* and IDA* approaches in [11] form the basis of the benchmark. Due to space limits, we refer interested readers to [11] for the 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 (very similar algorithm parameterizations), but performance variations are nonetheless significant.

We create a set of training and test instances out of existing pre-marshalling instances from [1] (BF) and from [3] (CV) as well as instances we generated. We filter out instances where all algorithms timeout/memout or are too easy. An overview is provided in Figure 3. Our BF-generated instances are not exactly the same as in [1] because their instance generation is not completely described.

The features used in our dataset are given in Figure 2, split into three categories. 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 can be computed quickly. Our feature generation code (and instance generator) is available at `https://bitbucket.org/eusorpb/cpmp-as`. We note that other features are certainly possible, such as probing features.

Original features are created in the standard way for algorithm selection benchmarks, based on domain knowledge. The first 5 features address the problem size and density of containers. Feature 6 counts the number of mis-overlaid containers, a naive lower bound to the problem, whereas Feature 7 counts how many stacks contain mis-overlaid containers. Feature 8 provides the lower bound from [1], analyzing indirect container movements in addition to the mis-overlays

[5] This benchmark is available in the algorithm selection library (`www.aslib.net`) under the name "PREMARSHALLING-ASTAR-2013"

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: Performance of CSHC trained on the three feature sets.

present in feature 7. Features 9 through 12 offer information on how many containers belong to each group. Features 12 through 15 attempt to uncover the structure of the groups of the top non-mis-overlaid container on each stack.

LFA features are constructed based on the suggestions of the latent features. Feature 17 is the density of containers on the "left" side of the instance. We note that this feature is likely "overtuned" to the algorithms in our benchmark. Feature 18 measures whether containers with high group values are on high or low tiers by multiplying the tier of a container by its group, summing these values together and dividing by the maximum this value could take (namely if the highest group container was in each slot). Feature 19 measures the L1 (manhattan) distance from the top left of a problem to each container in the latest exit time, averaging these distances if there are multiple containers in the latest exit 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. Features 21 and 22 come from LFA iteration 2. Feature 21 counts how many stacks with more than two containers are mis-overlaid, and Feature 22 counts "low" ($\leq max\text{-}group/4$) valued containers on the top of stacks.

5 Computational results

We evaluate our features using the cost-sensitive hierarchical clustering (CSHC) approach from [9]. Table 1 provides the performances[6] of a CSHC based portfolio when trained on the three datasets versus the best single solver (BSS) and the 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 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,

[6] All runtime data was generated on an AMD Opteron 2425 HE processor running at 2.1 GHz with a one hour timeout.

but the time taken to solve them on average. Thus, multiple iterations of the latent feature analysis process can lead to even better features, although there are clearly diminishing returns.

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. For future work, the latent feature analysis process could be more formalized. A number of open questions remain, such as what criteria to use to gauge the performance of a new feature during a single iteration of the latent feature analysis process. Further challenges are to determine the number of iterations to perform and what kind of performance/man-hour trade-off exists for each iteration past the first.

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. Kotthoff, L.: Algorithm selection for combinatorial search problems: A survey. AI Magazine 35(3), 48–60 (2014)
6. Lee, Y., Hsu, N.: An optimization model for the container pre-marshalling problem. Computers & Operations Research 34(11), 3295–3313 (2007)
7. 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)
8. Malitsky, Y., O'Sullivan, B.: Latent features for algorithm selection. Symposium on Combinatorial Search (2014)
9. Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M.: Algorithm portfolios based on cost-sensitive hierarchical clustering. IJCAI (2013)
10. Stahlbock, R., Voß, S.: Operations research at container terminals: a literature update. OR Spectrum 30(1), 1–52 (2008)
11. 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)