A TREE-SEARCH BASED HEURISTIC FOR A COMPLEX STACKING PROBLEM WITH CONTINUOUS PRODUCTION AND RETRIEVAL

Sebastian Raggl[(a)], Beham Andreas[(b)], Fabien Tricoire [(c)], Michael Affenzeller[(d)]

[(a,b,d)] Heuristic and Evolutionary Algorithms Laboratory, University of Applied Sciences Upper Austria, Softwarepark 11, 4232 Hagenberg, Austria
[(b,d)] Institute for Formal Models and Verification, Johannes Kepler University Linz, Altenberger Straße 69, 4040 Linz, Austria
[(c)] Department of Business Administration, University of Vienna, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria

[(a)]sebastian.raggl@fh-hagenberg.at, [(b)]andreas.beham@fh-hagenberg.at, [(c)] fabien.tricoire@univie.ac.at
[(d)]michael.affenzeller@fh-hagenberg.at

ABSTRACT
We present a real world steel stacking problem featuring non-instantaneous crane movements, continuous production and retrieval and stacking constraints based on the dimensions as well as temperature of the slabs. An exact Branch & Bound solver as well as three tree-search based heuristics is developed. Random benchmark instances derived from the real world problem are used to evaluate the performance of the heuristic solvers and compare them to the exact solver.

Keywords:
Stacking Problem, Branch & bound, Heuristic

1. INTRODUCTION

Stacking problems arise in many sectors of industry in a great number of variants. A good overview of the different problem variants including solvers and complexity considerations is given in (Lehnfeld & Knust, 2014). They differentiate loading, unloading, premarshalling and combined problems. According to their categorisation the problem considered in this paper is a combined loading unloading problem.

The work most closely related to this one is probably (Rei & Pedroso, 2013). They also describe a stacking problem with continuous production and delivery but with instantaneous crane movements and without any capacity or stacking constraints. They develop a stochastic tree search algorithm in order to tackle big problem instances.

The Stacking Problem considered in this paper arises in the operation of a steel factory. Steel slabs are casted according to a fixed schedule and have to be put onto a delivery stack according to a fixed list of delivery lots. The slabs can be moved by a crane which can only move a single slab at a time and only access the slabs on top of a stack. Every crane movement requires time to perform and the stacking is subject to some restrictions.

Figure 1 shows a possible layout of a stacking area with a single caster at the left, nine partially used buffer stacks in the middle, and a crane moving a slab to the handover stack at the right.

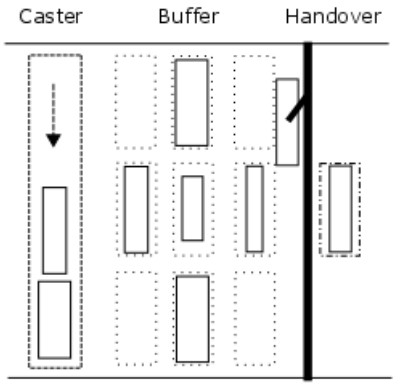

Figure 1: Example of a stacking area

If crane movements were instantaneous and the last slab was already casted this problem can be viewed as a variant of the Block Relocation Problem with additional stacking constraints. In this case the caster is simply viewed as another stack. The original authors of the BRP (Kim & Hong, 2006) use a Branch & Bound algorithm and a heuristic rule based on the expected number of additional relocations, called ENAR. Improvements on this approach come from (Ünlüyurt & Aydın, 2012). Two binary linear programming are introduced in (Caserta, et al., 2012) where one only considers strictly necessary moves and the other one also allows so called voluntary moves. Those models are improved upon by (Petering & Hussein, 2013) and (Expósito-Izquierdo, et al., 2015).

The remainder of this article is organised as follows. In Section 2 we describe the problem in detail. In Section 3 we present an exact Branch & Bound algorithm as well as a heuristic approach. Section 4 compares the two solvers in terms of solution quality and runtime using randomly generated problem instances derived from real world problem instances. Finally we discuss the results of the experiments and present an outlook on further research.

2. PROBLEM DESCRIPTION

There is a set of K casters $\mathrm{C} = \{c_k\}_{k=1}^{K}$ which produce N steel slabs $\mathrm{S} = \{s_i\}_{i=1}^{N}$ of different dimensions according to a fixed schedule. The slabs are to be placed onto a single handover stack h where they are picked up according to fixed delivery lots. A delivery lot consists of the due-time and the order in which the slabs of that lot should be in. At the due-time all slabs of a lot must be on the handover stack so a transporter can deliver them all to their destination. It is important to note that if the casting schedule and the order of the slabs in the transport lots match up the problem becomes trivial, but this is not always possible.

When a slab comes out of a caster and cannot be put directly onto a delivery stack can be put on one of M so called buffer stacks $\mathrm{B} = \{b_j\}_{j=1}^{M}$. There is a single crane which can transport a slab from the top of a buffer stack or front of a caster to a handover or buffer stack using a certain amount of time given that none of the stacking constraints described in 2.1 are violated. Note that the crane is not allowed to take a slab from the handover stack or to put a slab back onto the caster.

The objective is to deliver all the slabs according to the given delivery lots while performing as few crane movements as possible.

2.1. Stacking constraints

The height of the buffer stacks is limited but since the slabs have different dimensions this does not translate to a fixed amount of slabs per stack.

In order to guarantee stability of the stack as well as to prevent deformation of the slabs there is a limit on the allowed length and width difference between each slab and all the slabs beneath it in the stack. Note that this constraint only has to be checked when a slab is placed on a buffer stack because the delivery lots are fixed.

Another factor that determines if a slab can be stacked onto another is the temperature difference between the two. This constraint is necessary because the cooling is vitally important for the steel quality and reheating a slab by placing a hotter slab on top can mean that the slab is no longer usable. The temperature of a slab is calculated using a simple logarithmic cooling scheme starting at the casting temperature and time.

2.2. Time constraints

Every slab has a production timestamp and a delivery timestamp. The most basic temporal constraint is that a slab cannot be moved before its production or after its delivery. Since the caster can only hold a certain number of casted slabs and the casting schedule is fixed, there is a time window in which a slab can be taken out of the caster. This window is between the casting time of the slab and the casting time of the n+1 slab. Otherwise the caster would have to be stopped which should be avoided at all costs.

Similarly, there is a time window for putting slabs on the handover stack, which begins with the delivery time of the previous transport lot and ends with the delivery time of the slab.

3. SOLVING THE STACKING PROBLEM

The solution of a stacking problem consists of a list of crane movements called moves from here on. A move is described as a tuple of the source and the target. Since there are different types of locations we distinguish between five different kinds of moves:

- *Put* (C, B)
- *Remove* (B, h)
- *PutDirect* (C, h)
- *Relocate* (B, B)
- *Delivery* $(\mathrm{h}, -)$.

Put moves take the first slab from a caster and put it on buffer stack. *Removal* moves are taking a slab from a buffer stack and putting it on the handover stack. Of course when a slab from the caster can be directly put onto the handover stack it would be nonsense and even potentially impossible to put it on a buffer stack first. This is what a *PutDirect* move is for.

A *Relocation* move is one that moves a slab from one buffer stack to another. The BRP literature differentiates two kinds of *Relocation* moves, namely forced and voluntary moves (Caserta, et al., 2012). If the slab that has to be removed next does not lie on the top of a stack the slabs above it have to be relocated. This is called a forced move. Not considering voluntary moves can lead to improved run time at the cost of potentially worse solutions.

Delivery moves are special in a number of ways. They always move all the slabs of a transport lot at once. We considered them to be instantaneous because we are not concerned with the actual delivery process but instead only care when the handover stack is free to use again.

All moves that put slabs on a buffer stack are subject to the stacking constraints described in 2.1. A slab can only be put on the delivery stack if the stack is either empty or already contains all the slabs that are in lower positions in the same delivery lot.

The combination of the current time and the creation and delivery times of all slabs determine which moves are valid. On the other hand the exact time at which each move is performed is irrelevant as long as the constraints are respected. Given a valid sequence of

moves, it is easy to calculate a time window within which each move must be performed.

Table 1: Example1

Creation time	Due-time/ Position	Initial position
0	20/1	b_1/1
0	10/0	b_1/0
10	20/1	c_1

Table 1 shows an example with a single caster, two buffer stacks and three slabs. Figure 2 shows the same sequence of moves and therefore the same solution distributed in time in three different ways.

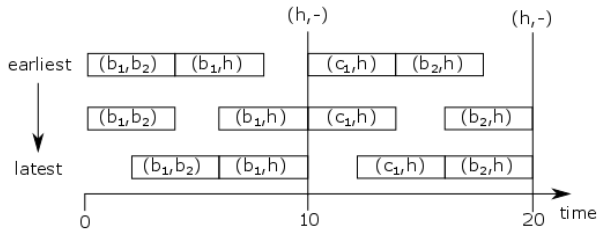

Figure 2: Solution of Example1

3.1. Branch and bound

In order to perform a branch and bound search we need a lower bound. In order to get a valid lower bound we consider that, for every remaining lot rl we need one *Delivery* move. Every slab that is not already at the handover stack needs to be moved at least once. If a slab lies on top of a slab that has to be put on the handover stack before, the upper slab must be moved at least two times. So if $lb(x)$ is the sum of minimal required moves of all slabs at location x and m is the number of moves already performed a total lower bound is:

$$LB = m + \sum_{b \in B} lb(b) + \sum_{c \in K} lb(c) + rl$$

Note that this lower bound is only valid if there is only a single handover stack because if there are multiple handover stacks a slab that is blocking another slab could potentially be placed directly on another handover stack instead of a buffer stack.

We perform a depth first search using all possible moves but we use the same order as described below in order to improve the chances of finding good solutions earlier.

3.2. Heuristic tree search

Due to the time and stacking constraints every move has the potential to generate a situation where there is no valid solution. While choosing a valid move in a BRP will never lead to an invalid solution. So instead of using a heuristic that aims to choose the best move at every step, we propose a heuristic that uses a list of all "good" moves ordered by quality. So in most cases the first move on the list should be the one used in the final solution, but if there is a move that results in a situation where there is no solution, alternative moves can be applied.

To find such a list of moves let us make some observations. Placing a slab on the handover stack is preferable to placing it on a buffer stack because once it is on the handover stack the crane never has to touch that slab again. Moving a slab from a buffer stack can always be done immediately. Moving a slab from the casters or the handover stack might involve waiting for the casting or delivery time of the slab. Therefore *Remove* moves should be done before *PutDirect* moves and *Relocate* moves before *Put* moves. So the moves that will be tried are in the following order:

1. If a caster must be cleared (see 2.2)
    a. Return all *PutDirect* moves.
    b. Return safe *Put* moves.
    c. Return remaining *Put* moves.
2. If a delivery must be performed (see 2.2) return the *Delivery* move.
3. Otherwise return a list of all
    a. *Removal* moves
    b. *PutDirect* moves
    c. Safe *Relocation* moves
    d. Safe *Put* moves
    e. *Delivery* move
4. If there are no safe moves return a forced move.

A safe move is one that does not increase the lower bound LB described in Section 3.1. *Removal* and *PutDirect* moves are always safe because moving a slab away from the handover is not allowed. This means that if the move were unsafe it is also forbidden because it cannot lead to a feasible solution. *Put* and *Relocate* moves are considered safe, when any slab of the source stack has to be moved to the handover stack before the topmost slab and none of the slabs of the target stack have to be handed over before the topmost slab of the source stack.

If there are multiple safe *Put* or *Relocation* moves that will put the same slab on different empty buffer stack only one of them is considered in order to reduce the total count of possible moves. This is however only valid if the crane movement times between all the stacks are equal and the height limits of the buffer stacks are also equal.

Based on this list of good moves we define three different heuristic solvers. The first, we will call it H1, uses the list of good moves as defined above to do a depth first search and return the first solution found.

Note that a solution of H1 is a valid upper bound for the B&B solver described in Section 3.1.

The second solver uses a two stage approach where in the first step a breadth first search is used to generate level after level of the search tree, using all possible moves, until a level is generated that has more than a certain number of nodes. Then each of these partial solutions is completed using H1. We call this solver H(n) where n is the minimum number of nodes that we generate in the first step. It would also be possible to specify the number of levels to generate, but this gives us better control over the runtime. One nice feature of this is that the second step can be parallelized because the partial solutions are totally independent of each other.

Finally we use the B&B solver as described in Section 3.1 but instead of considering all moves we only consider the moves specified before. From now in order to distinguish between the two variants we will refer to the exact solver as E B&B and the heuristic one as H B&B.

4. EXPERIMENTS

We evaluated the solvers presented above using a set of randomly generated test instances of different sizes. The test instances have two casters that are used to produce a varying number of lots with four slabs per lot. The production dates of the slabs, as well as the due dates of the lots are randomly chosen. Then a random start date is chosen and all the slabs are put either in a caster or in one of the M buffer stacks depending on their production dates. Using this schema we generate eight different scenarios with 10 random instances each. The scenarios have 4, 8, 16 or 32 lots and 4 or 8 buffer stacks.

We perform three different experiments using this test instances. In Section 4.1 we compare the runtime and solution quality of our heuristic solvers. Afterwards we compare the best solution of the H B&B solver against the optimal solution found by the E B&B in order to judge the quality of our heuristic solutions.

Finally because the real world application of the solvers requires us to be able to find a solution quickly we compare how the heuristics perform when given a time limit of one minute. So the search is aborted after one minute and the best solution reached so far is reported. The reason we need to be able to find solutions quickly is that it can easily happen that the real-world circumstances change in such a way that our solution is no longer valid. If we require hours to come up with a new solution this is a problem.

All solvers were implemented in C# using the .NET Framework 4.6.1 using the Task Parallel Library. All tests were run on a Dell Latitude E6540 with an Intel i7 4810MQ CPU @2.80GHz and 16 GB RAM running Windows 7.

4.1. Comparing the heuristics

Table 2 shows a comparison of the average solution quality of the heuristic solvers for eight different scenarios with ten instances each. The N and M columns are the number of slabs and the number of buffer stacks respectively. LB is the average lower bound as described in Section 3.1. The remaining three columns contain the number of moves more than lower bound the algorithms described in Section 3.2 required on average.

The problems get harder the more slabs they contain. Interestingly the problems are also harder if they contain fewer buffer stacks. This may seem counter intuitive because having more stacks also means there are more possible moves. What increases the difficulty is that the likelihood of producing a new conflict when putting a slab on a buffer stack is higher the fewer buffer stacks there are.

Table 2: Best Solutions of Heuristic Solvers.

N	M	LB	H1	H(50)	H B&B
16	4	26.5	0.6	0.6	0.5
16	8	25.8	0.4	0.3	0.3
32	4	55.0	5.8	4.9	2.6
32	8	54.3	3.0	2.6	1.5
64	4	112.3	12.7	11.0	4.4
64	8	111.6	9.5	9.0	4.0
128	4	226.0	22.6	21.9	15.9
128	8	225.2	19.0	18.7	14.4

Even the simple H1 manages to find solutions within 10% of the lower bound for the majority of problem instances. As described above H(50) runs H1 on all the nodes of the first level of the search tree which has more than 50 nodes. For the problem sizes we looked at this means evaluating all possible combinations of the first 2-3 moves. This brings an improvement over H1 for 25 out of 80 problem instances.

Table 3: Runtime of Heuristics in Seconds.

N	M	H1	H(50)	H B&B
16	4	0.00	0.01	0.00
16	8	0.00	0.03	0.00
32	4	0.00	0.03	550.85
32	8	0.00	0.82	782.58
64	4	0.00	0.07	2624.04
64	8	0.00	0.58	2528.75
128	4	0.01	0.36	3600.00
128	8	0.16	9.78	3600.00

H B&B is never worse than H(50) and outperforms it on 54 out of 80 problem instances but as can be seen in Table 3 is the runtime much higher. The runtime of H1 is less than 0.1 seconds for all but one problem instance with 128 slabs and 8 buffer stacks. H(50) takes less than 10 seconds to run for all instances except the one mentioned above. The H B&B solver fails to complete in less than one hour for at least one instance in every problem class except for the two with only 16 slabs. For the two biggest problem classes no instance can be solved in less than one hour.

4.2. Optimality gap

In Figure 3 we can see that of our 80 test instances only 33 could be solved by the exact B&B within one hour. H1 solved 21 instances to optimality while H(50) managed to solve 23. The H B&B solver found the optimal solution in 30 cases. The solutions to the remaining three instances all required one more move than would have been optimal. Additionally there are two instances where the E B&B managed to find better solutions than the H B&B despite not running to completion.

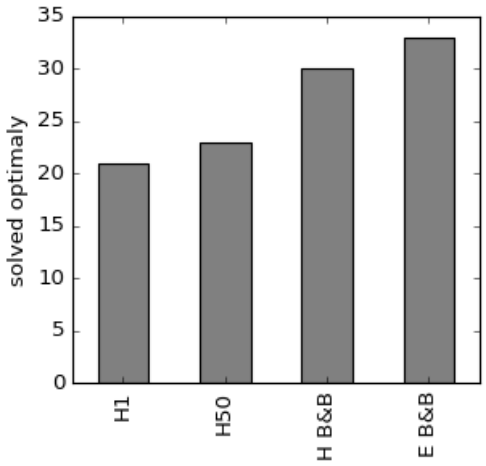

Figure 3: Number of Problem Instances Solved Optimally

It is interesting to look at situations where the heuristic is unable to find the optimal solution. One problem is that sometimes it is necessary to perform otherwise useless moves in order to enable save moves that are vital for the final quality.

Another point is that our heuristics manage contain rules about which moves are “good” and which are not based on general considerations about slab stacking as well as the temporal constraints. The stacking constraints described in Section 2.1 however are only used to forbid moves that would violate them completely but not in deciding which legal move would be better. This can lead to poor decisions where a stack is blocked by a slab that cannot be stacked upon.

4.3. Comparing the heuristics with time limit

In Figure 4 and Figure 5 we show the best solutions found divided by the lower bound for all problem categories after one hour and one minute respectively.

The two smallest problem sizes are the same because they were all solved in less than one minute. Five of the twenty problems with eight lots could not be solved. The one minute solution was worse in three instances by at most three moves.

For the problems with 16 and 32 lots the additional runtime made more of a difference. But even in those larger problems was the biggest improvement that an additional 59 minutes of runtime could bring was eight moves.

What the graphs do not show is that for 49 out of 80 problem instances the quality did not differ at all between the run with a one hour time limit and a one minute time limit. All of this suggests that the order we choose the moves in is good since the heuristic manages find the good solutions early.

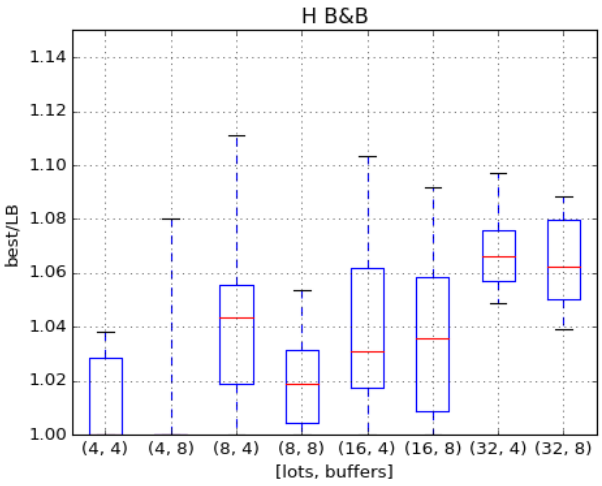

Figure 4: Best Solutions of H B&B after 1 hour

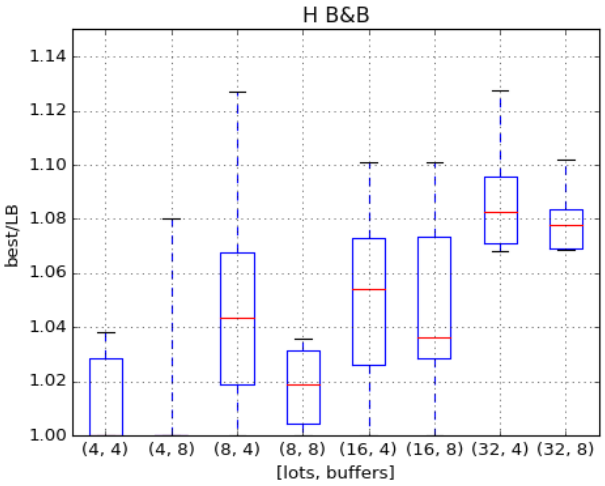

Figure 5: Best Solutions of H B&B after 1 Minute

5. CONCLUSION

The exact solver is unable to solve all but the smallest problems in reasonable time and where it did manage to

find a solution there was barely an improvement over the heuristic.

We showed that our heuristic works and that finding a better solution than the first one which is returned by H1 takes considerably more effort for all but small problems. Running H1 from multiple starting points can improve the result, but the time is better spent on running the H B&B for as long as possible.

While it is good to give the H B&B solver as much time as possible, because it could find better solutions, it is often not necessary to run it to completion because the heuristic aims at looking at the most promising moves first.

6. MANAGERIAL INSIGHTS

Efficiently stacking the slabs coming out of the caster and preparing transport lots previously required an experienced worker. We showed that automatization is indeed possible using the approach presented in this paper. Our heuristic is able to deliver high quality solutions in a reasonable runtime. In addition to this work we are also currently working on optimization algorithms for the transport lot building as well as the transportation and on integrating

In addition to freeing human resources and reducing labour costs are these optimizers also useful for evaluating the impact of planned changes on the performance of the system.

7. OUTLOOK

We are working on applying our results to the real world case. One difficulty is that sometimes it is necessary to bend the rules laid out in Section 2.1 and 2.2 slightly in order to find a solution at all. If that is the case our algorithms cannot find an answer. There are a few possible solutions to this problem. One approach is to relax the constraints and try again if no solution was found. This is problematic because if the algorithm uses all the time we allotted only to be started again afterwards we may not have a solution in time. The better solution would be to allow but penalize violating the constraints. We are currently investigating how we can accomplish that.

ACKNOWLEDGMENTS

The work described in this paper was done within the COMET Project Heuristic Optimization in Production and Logistics (HOPL), #843532 funded by the Austrian Research Promotion Agency (FFG).

REFERENCES

Caserta, M., Schwarze, S. & Voß, S., 2012. A mathematical formulation and complexity considerations for the blocks relocation problem. *European Journal of Operational Research*, 219(1), pp. 96-104.

Expósito-Izquierdo, C., Melián-Batista, B. & Moreno-Vega, J. M., 2015. An exact approach for the Blocks Relocation Problem. *Expert Systems with Applications*, 42(17), pp. 6408-6422.

Forster, F. & Bortfeldt, A., 2012. A tree search procedure for the container relocation problem. *Computers and Operations Research*, 39(2), pp. 299-309.

Kim, K. H. & Hong, G.-P., 2006. A heuristic rule for relocating blocks. *Computers & Operations Research*, 33(4), pp. 940-954.

Lehnfeld, J. & Knust, S., 2014. Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. *European Journal of Operational Research*, 239(2), pp. 297-312.

Petering, M. E. & Hussein, M. I., 2013. A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem. *European Journal of Operational Research*, 231(1), pp. 120-130.

Rei, R. J., Kubo, M. & Pedroso, J. P., 2008. Simulation-based optimization for steel stacking.. In: H. A. Le Thi, P. Bouvry & T. Pham Dinh, eds. *Modelling, Computation and Optimization in Information Systems and Management Sciences*. Berlin: Springer, pp. 254-263.

Rei, R. J. & Pedroso, J. P., 2012. Heuristic search for the stacking problem.. *International Transactions in Operational Research*, 19(3), pp. 379-395.

Rei, R. J. & Pedroso, J. P., 2013. Tree search for the stacking problem. *International Transactions in Operational Research*, 203(1), pp. 371-388.

Ünlüyurt, T. & Aydın, C., 2012. Improved rehandling strategies for the container retrieval process. *Journal of Advanced Transportation*, 46(4), p. 378)393.