32nd European Modeling & Simulation Symposium
17th International Multidisciplinary Modeling & Simulation Multiconference

ISSN 2724-0029 ISBN 978-88-85741-44-7 © 2020 The Authors.
DOI: 10.46354/i3m.2020.emss.027

Effects of Arrival Uncertainty on Solver Performance in Dynamic Stacking Problems

Sebastian Raggl[1,2,*], Andreas Beham[1], Stefan Wagner[1] and Michael Affenzeller[1,2]

[1]University of Applied Sciences Upper Austria, Hagenberg, Austria
[2]Johannes Kepler University, Linz, Austria

[*]Corresponding author. Email address: Sebastian.Raggl@fh-ooe.at

Abstract

In this paper we present a dynamic stacking problem with uncertainty. We developed a simulation environment, an optimizer for solving it, and performance measures to determine the success of the optimizer. The problem requires handling incoming blocks, stacking them efficiently, and meeting deadlines for delivery, while not knowing exactly when blocks will arrive or when they will be ready for delivery.
The optimizer models the problem as a dynamic Block Relocation Problem and solves it using a branch&bound based heuristic. The simulation and optimizer run concurrently and the distribution of random variables is not disclosed to the solver and must, therefore, be estimated.
We study the influence of uncertainty on the solver and show that the degree of uncertainty has a significant impact on the performance of the overall system. We also experiment with different measures to estimate uncertain arrival times and show that the choice of measure is important for achieving good performance.
Keywords: dynamic optimization problem;stacking;uncertainty

1. Introduction

Dynamic stacking problems have not yet been covered widely in the literature, while static problem variants and solution approaches that solve a given scenario with minimal relocation effort have been discussed at lengths. Nevertheless, a multitude of applications in the real world exist, where uncertainties and dynamic effects may not be ignored. The domains range from container stacking and retrieval in harbors to stacking steel products after casting or rolling. An overview of uncertainties and dynamic aspects has been given (Beham et al., 2019). In order to address these aspects, stochastic planning models and strategies on how to handle disruptions are needed. In the context of dynamic optimization problems (DOP) the application of algorithms to optimize decisions at runtime is called "online solving", while the overall performance observed after consecutive applications of online algorithms is called "offline performance" (Nguyen et al., 2012). The latter is an important criterion in determining success or failure to control the dynamic problem. However, this performance is available to an online solver only in hindsight.

A brief literature review on dynamic stacking problems respectively stacking problems with uncertainty is presented in this work. A dynamic container reloca-

tion problem (DCRP) has been introduced (Akyüz and Chung-Yee, 2014). However, the only dynamic aspect in the DCRP is the rolling planning horizon. There are no uncertainties, and all events occurring within the planning horizon are known. The stacking problem (SP) (Rei and Pedroso, 2013) extends the DCRP by associating a time window (release, due) to each block. This time window describes its earliest availability at a source stack and its latest possible relocation to a handover stack. The steel stacking problem (SSP) redefines a block to be a material, i.e., slab, coil, bloom, sheet, etc., which gains additional attributes such as temperature, length, width, height, and weight that are relevant to a number of stacking constraints (Raggl et al., 2018). In the SSP *two* time windows are associated with each material as there are both, release and due date, for the source and the handover stack. Additionally, the SSP features non-instantaneous crane movements.

Uncertainties have also been described, for instance in form of uncertain weights of containers in a port application (Kang et al., 2006). Also handover priorities may be uncertain, for instance in the online block relocation problem (BRP) only the next to be retrieved block is known, while the order of all other blocks is unknown and a leveling heuristic with a known competitive ratio has been described (Zehendner et al., 2017). Another work considers time windows for the handover in which trucks randomly arrive to pick up blocks (Ku and Arthanari, 2016). In the stochastic container retrieval problem blocks are assigned to batches that have a fixed and known order, but within a batch the order is random and determined online (Galle et al., 2018).

We published a dynamic stacking problem with uncertain arrival and retrieval time windows that can only be solved online (Raggl et al., 2020). We implemented a simulation, defined challenging benchmark instances and compared two online solvers for the problem using the relevant performance indicators. This work, uses the same dynamic stacking problem and investigates the effect of increased uncertainty of the arrival rate on solver performance. We then show effects of addressing such uncertainties by using different estimators for the arrival frequency. Taking such approaches to the extreme we observe side effects that actually lead to worse performance.

2. Simulation-based Dynamic Stacking

The dynamic stacking problem that we consider in this work is composed of three types of stacks at which blocks may be placed such that they are positioned right above each other. Only the topmost block at each stack may be accessed by a crane which may only load a single block at a time. Blocks have a certain (known) due date and a certain (unknown) ready date which precedes the due date. They must reside within the system until their respective ready date and should leave the system before their due date.

Arrival stack This stack is served by the upstream process, which inserts new blocks to the bottom and thus acts as a first-in-first-out (FIFO) queue. If the arrival stack is full, the next block is lost and the upstream process is suspended.

Buffer stacks These stacks act as a buffer between the arrival and the handover stack. Each block consumes one unit of height and there is a maximum height per stack - here it is the same among all buffer stacks.

Handover stack This stack is filled by the crane and cleared by the downstream process. It can only hold one block and thus, the clearing decision is made immediately upon dropping off a block there, which is also called a *delivery*. Such a delivery takes some time after which the handover stack is ready again.

This problem is described in form of a simulation model that is implemented using the Sim# simulation framework (Beham et al., 2014). Sim# is a process-based discrete-event simulation framework, where a process is simply a C# method that manipulates its local state as well as the simulation's state.

The dynamic stacking problem is implemented using a pseudo-realtime simulation environment, meaning that the advancement of the simulation time may incur a real-world delay. This is useful when the solver interacting with this dynamic stacking problem should be tested in a real-world like scenario, i.e., the solver process runs concurrently to the simulation. However, the pseudo-realtime simulation may also run in virtual time, i.e., as fast as possible, in which case a synchronous interaction between simulation and solver may be achieved. Still, the availability of the decision within the simulation can be simulated by accounting for a delay that is equal to the solver's runtime.

2.1. Processes

The processes within this simulation govern the change of the system state. In this dynamic stacking problem there are four processes that we describe here.

1. An upstream process that produces new blocks at the arrival stack.
2. A block process that will determine the block's readiness.
3. A crane process that will execute the crane movements.
4. A downstream process that will clear the handover stack.

Upstream

The upstream process spawns new blocks at the bottom of the arrival stack in random intervals. If the arrival

stack is observed to be full after such an interval, that block is considered to be "waste" and the arrival process is suspended. The arrival process is resumed after the crane picks up a block from the arrival stack and thus creates room for a new block, which is spawned again after a random interval. An important parameter of this process is the arrival rate ARR. We use a log-normal distribution to model the stochastic of arrivals. In this experiment, we change the coefficient of variation of ARR and evaluate the effect on the solver. A solver may observe the last 100 sampled inter-arrival times which it can use to generate an estimate of the uncertainty.

Block

The block process is initiated when a new block is created and marks the block ready when its ready date has passed. An important parameter is the mean customer required lead time DUE and the mean relative time span RDY after which it becomes ready. The customer required lead time will be sampled from a log-normal distribution with mean DUE and the relative readiness time will be sampled from a uniform distribution with mean RDY. The actual readiness time is not disclosed and thus unknown to a solver.

Crane

The crane process is initiated by the solver. In this process the moves will be performed according to a schedule specified by the solver. Before a move is performed it is checked for validity. When an invalid move is encountered, the crane process skips it and continues with the next valid move. Moves may become invalid for a number of reasons, such as forbidden destinations, i.e., relocation *to* the arrival stack, violating the height restrictions and more.

When a new schedule is sent, the crane will abort its current schedule after completing the move that is currently in progress. The crane takes some time to move horizontally between stacks and some time to lower and raise the hoist in order to pick up or drop off a block. The longer the distance, the longer it takes for both directions. The actual time to perform a single move in one of those directions is described by parameters HOR and VER for the mean horizontal movement (crane) and the mean vertical movement (hoist) respectively. Both HOR and VER are described by log-normal distributions in this work. Again, solvers may observe the last 100 relocation times and estimate the uncertainty.

Downstream

The downstream process is initiated when the crane drops-off a block at the handover stack. The handover stack then becomes unavailable until it is cleared. During this time no new blocks may be delivered. Similar to the above processes, data about the last 100 clearing intervals are available to the solver for uncertainty estimation. The stochastic variable HND describes the average handover time.

2.2. Performance Measurement

The performance of the described dynamic stacking problem is described in several dimensions. We describe some of these in more detail and derive a lexicographic objective function from some of these performances that closely matches with the priorities observed in a comparable real-world scenario. The following key performance indicators (KPIs) are updated live as the simulation is running. Thus, the KPIs represent the performance of the system up to the current simulated time.

Blocked arrival time (BAT) counts the total time that the arrival process was suspended. The higher this time, the worse the performance as blocking the upstream processes may have severe consequences. For instance, when continuously casting steel, blocking the caster is highly undesirable as there are long setup and potentially cleaning operations necessary when restarting.

Total blocks on time (TBT) counts all blocks that have been delivered before their due date, as well as those that are not overdue and still at a stack. The more blocks that have been delivered on time, the better the performance.

Crane manipulations (CM) counts the total number of block relocations performed by the crane. Picking up and dropping off blocks is a critical process where accidents are more likely to occur. Thus, the less manipulations are necessary, the better.

Mean service level (MSL) is the relative number of blocks that are on time among all those that are delivered.

For comparing solvers, we chose a lexicographic objective that includes the first three KPIs: (1) minimize BAT, (2) maximize TBT, (3) minimize CM. This objective function represents the priorities that we have observed in real-world cases in steel stacking where it is of utmost importance to avoid stalling the continuous casting process. Furthermore, it is important to adhere to the due times as much as possible and also deliver as much as possible in terms of the total quantity. And third, excessive restacking should be avoided in order to minimize probabilities of accidents and also conserve energy and thus crane manipulations are also to be minimized.

3. Solving the Stacking Problem

The solver receives the world state from the simulation, translates it into a dynamic Block Relocation Problem (dBRP), and solves that. Then it uses the dBRP solution to create a schedule for the crane to execute and sends

it back to the simulator. While the crane is working on a schedule we avoid sending another one because, due to the lack of synchronization between the solver and the simulation, this easily results in invalid moves. This means that once a schedule is sent to the crane it is fixed. To ensure that the solver can react to changes such as new blocks arriving or becoming ready, we have to limit the number of moves per schedule. On the other hand, communicating multiple moves at once eliminates the communication overhead and leads to better crane utilization, so we need to balance these two factors.

The buffer stacks, as well as the arrival stack, are translated to stacks in the dBRP. The handover stack is simply the target of any remove moves. The arrival and handover intervals, as well as the crane move time, are estimated from the last 100 occurrences observed by the simulation.

Since the simulation does not dictate an order in which the blocks should be removed, we have to determine the handover priorities ourselves. There is a large number of ways to assign priorities to blocks. We determine the handover priorities by first sorting all the blocks using a lexicographic ordering of four factors:

1. Is the block ready?
2. Can it be put on the handover in time?
3. Time until due date?
4. How many blocks are above it in the stack?

Given a list of blocks sorted this way, we could use the index in the list as the priority, but using unique priorities is quite limiting. Especially for blocks with a due date in the far future, it does not make sense to force the solver to use a strict removal order, because this order is likely to change anyway. So to determine the priorities we start with priority zero and iterate over the blocks in the list. Every time one of our four factors differs from a block to the previous one, we increment the priority counter. To model the increasing uncertainty as time goes on, we consider due times of two blocks to be equal if they differ by less then 20%.

The biggest difference to the static BRP is that blocks on the arrival stack must be taken from there before it runs full and is blocked. Additionally to the usual removal and relocate moves, we need to perform insertion moves, that take a block from the arrival stack and put it on a buffer or handover stack. These insertions generate a conflict between our three objectives. In respect to blocked arrival time (BAT), it seems to make sense to clear the arrival stack as soon as possible. But filling up the buffer stacks, by immediately moving arriving blocks to a buffer stack, also leads to more relocations (CM) needed to remove blocks that become ready. This leads to delayed deliveries and a decrease in TBT. It is therefore important, to perform each insertion as late as possible, to reduce the pressure on the buffer stacks, but early enough to avoid blocking the arrival stack.

To get the possible moves for every dBRP state we use the following procedure. First, we calculate the latest time the next insertion move can be performed. It is the time of the previous arrival plus the number of free spaces on the arrival stack times the estimated arrival time. If there is free space on the arrival stack and we have time for at least two crane moves, before we need to do an insertion, we will not perform an insertion yet. If we detect that doing an insertion now will prevent us from performing the next removal we do not allow the insertion. Otherwise, we add all insertion moves to the list of possible moves. If there is enough free space in the buffer stacks, we force the solver to perform the insertion by not adding any other moves. If we are not forced to do an insertion we look for remove moves and immediately return those if there are any. If there are neither insertions nor removals, we need to perform relocations. We consider so-called safe moves, enabling safe and forced moves known from the static BRP (Tricoire et al., 2018), and also use the same schema for prioritizing possible moves as in the static case (Raggl et al., 2018).

We solve the dBRP model outlined above, using a branch&bound based heuristic algorithm with the number of crane manipulations (relocations) as the objective. Blocked arrival time and total blocks on time are only optimized indirectly. BAT by not allowing other moves when an insertion is needed and the due times are respected because they are used to determine the block priorities. It would be possible to optimize for those KPIs more directly by including them in the objective, but that means we cannot use lower bounds from the BRP which speed up search significantly (Tricoire et al., 2018). The crane schedule is constructed using at most the first two moves from the dBRP solution. If we encounter a removal, where either the handover or the block is not yet ready, we stop there.

4. Results

As discussed above, the critical decision of when to perform insertion moves is based on estimates for the inter-arrival and crane move times. Both of these are uncertain and the distribution and its parameters are unknown to the solver. But the simulation provides the last 100 measurements of both times, so this information is used. There is a number of possible measures we can use to estimate the times, e.g., mean, median, min, max or quartile. We therefore want to investigate two questions:

- How does the solver perform under different degrees of uncertainty?
- How does the choice of estimator influence the solver performance?

To answer these questions we generate random prob-

Table 1. Simulation model variables

List of random variables		
ARR		arrival rate
DUE		due dates, i.e., required lead time
RDY		ready dates (relative to due date)
HOR		crane movements (horizontal)
VER		hoist movements (vertical)
HND		handover intervals
Fixed parameters		
$\mathbb{E}$[RDY]	= 0.75	Expected readiness factor
CV[RDY]	= 0.2	Coefficient of variation of readiness factor
$\mathbb{E}$[HOR]	= 2.0	Expected crane move time along whole runway
CV[HOR]	= 0.2	Coefficient of variation of crane move time along whole runway
$\mathbb{E}$[VER]	= 0.5	Expected hoist move time from top to ground
CV[VER]	= 0.2	Coefficient of variation of hoist move time from top to ground
$\mathbb{E}$[HND]	= 5.0	Expected handover interval
CV[HND]	= 0.2	Coefficient of variation of handover interval
$\|B\|$	= 6	Number of buffer stacks $b \in B$
H_b	= 6	Maximum height of buffer stack b
H_a	= 4	Maximum height of arrival stack a
$\mathbb{E}$[ARR]	= 0.07	Expected arrival rate
CV[ARR]	$\in \{0.25, 0.5, 0.75, 1.0\}$	Coefficient of variation of the arrival rate
$\mathbb{E}$[DUE]	= 480	Mean customer required lead time
CV[DUE]	= 0.2	Coefficient of variation of customer required lead time

lem instances using the parameters shown in Table 1. In order to retain interpretability of the results we choose to only change the coefficient of variation of the arrival rate. All other random variables get a fixed coefficient of variation of 0.2. To estimate arrival time and move time we use the mean of the observed values. We use 4 different coefficents of variation, namely 0.25, 0.5, 0.75 and 1.0 and for each of these settings we generate 40 random instances for a total of 160 instances. The solver gets a time-limit of 250 milliseconds to solve each dBRP instance. All experiments were performed on a Lenovo ThinkStation P520 with an Intel Xenon 8 Core 3.7Ghz running Windows 10. The simulation is written in C# and executed using dotnet core 3.0 while the solver is written in Rust and compiled using rustc 1.43.

Figure 1 shows that the more we increase the coefficient of variation on the arrival time, the more blocked arrival time (BAT) we can observe. This makes sense because whether or not an insertion move should be performed is decided based on the estimated inter-arrival time and the crane-move time. If the solver overestimates the time until the next insertion move is needed, the arrival stack can get blocked. This is increasingly likely as the variance increases since we use the mean of the observed arrival times as our estimate. Since blocking the arrival stack leads to fewer blocks being produced, both, the total blocks on time (TBT), as well as the number of crane moves (CM), decreases. A higher variation in arrival frequency does however not significantly affect the service level (MSL).

In order to reduce the likelihood of overestimating the time it takes, for the next block to arrive, we can change the measure we use for estimating the inter-arrival time. We experimented with replacing the mean with the first quartile and the minimum of the observed values. We did not change anything else and used the same problem instances as before. Figure 2 shows the average and standard deviation of our KPIs when using the three different estimates. Using the first quartile gives, as expected, a modest improvement over the mean in terms of BAT as the uncertainty increases. This improvement is achieved by focusing more on insertions and less on removals and we can see this by looking at the mean service level. While the MSL is reduced by using the first quartile as an estimator, the TBT does not show a difference between the quartile and the mean. This is because using the quartile means blocking less and therefore producing more blocks.

Estimating the arrival time using the observed minimum is even more conservative than using the first quartile and so we would have expected the BAT to be even lower. Instead, we observe that the arrival is blocked much longer as the uncertainty rises. Additionally, it is not as consistent between instances. The decline in service level is again explained by the solver being forced to perform insertion moves much earlier. While the lower TBT is caused by both this shift in focus as well as less overall blocks produced due to the higher BAT.

To explain why using the minimum leads to a higher BAT when we expected it to lead to less blocking, we turn to Figure 3. There, we see both, the BAT as well as the buffer fill-level over a single representative problem instance, with the highest level of variance on the arrival intervals we studied (CV[ARR] = 1.0). The BAT at the end is highest for minimum, lower for mean and lowest for the first quartile just like we have seen in Figure 2. But crucially, the number of times the arrival is blocked is lowest for minimum and highest for the

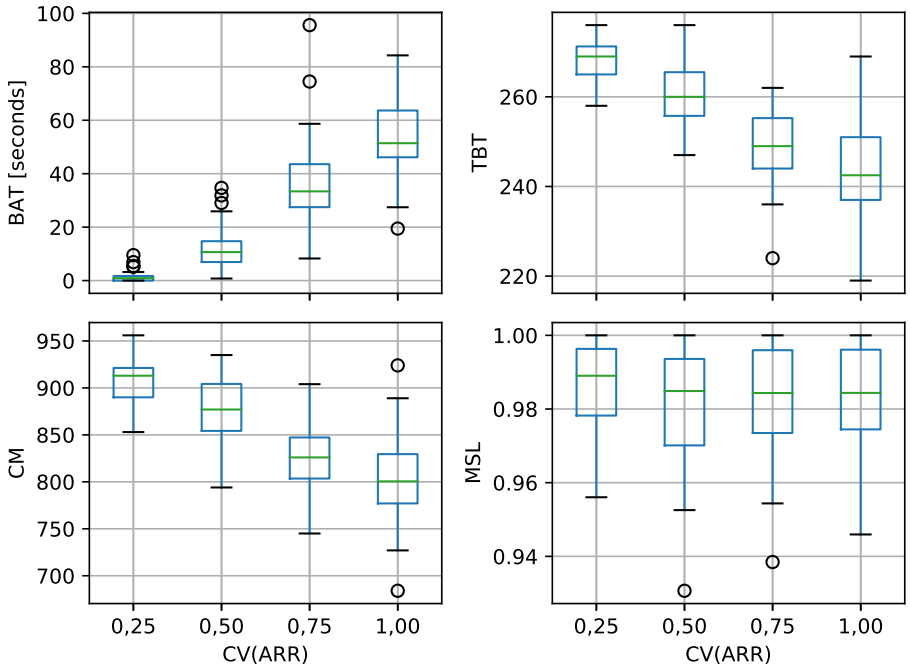

Figure 1. Solver performance with different coefficients of variation for the arrival frequency.

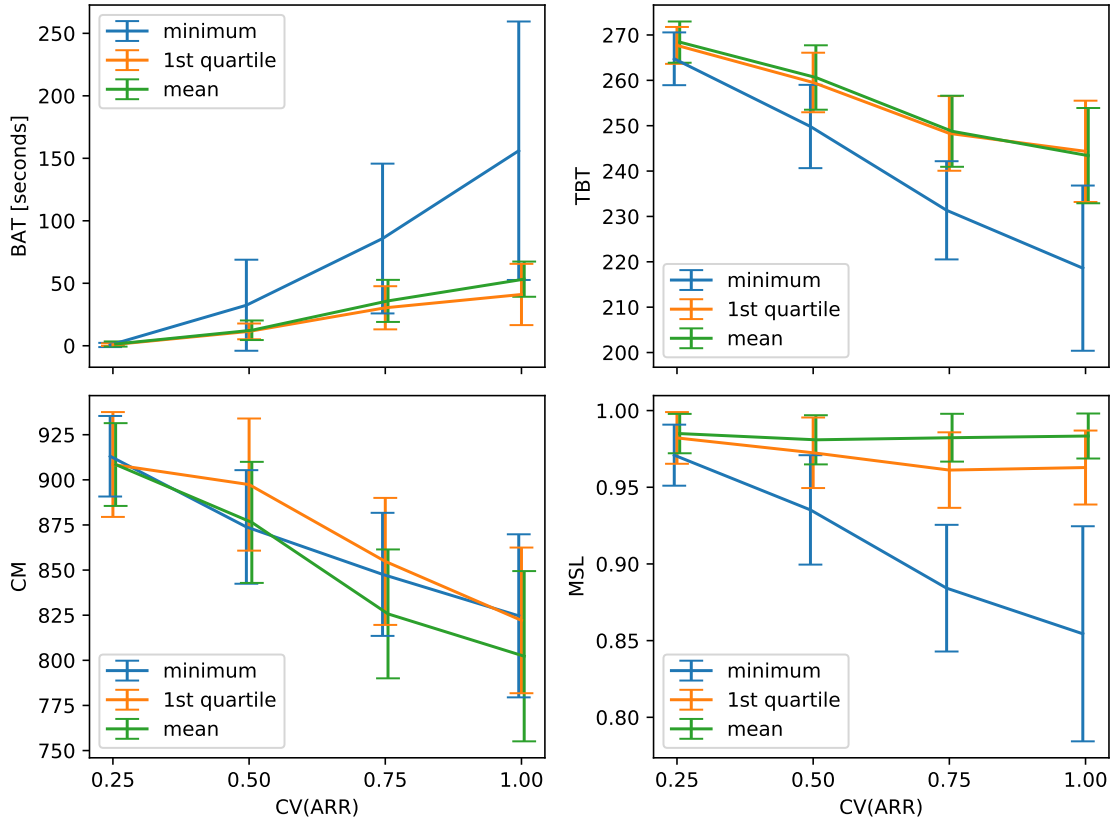

Figure 2. Solver performance when using different estimators with increasing variance on arrival frequency.

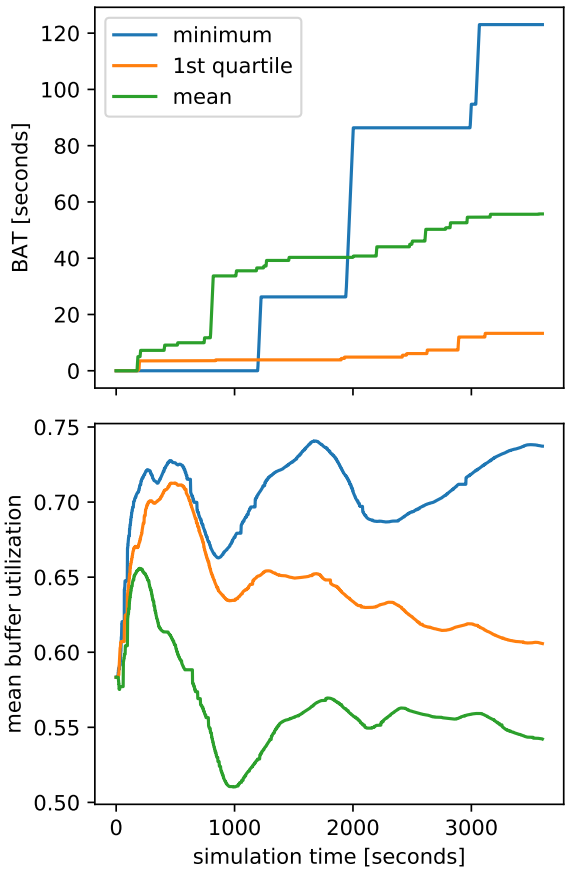

Figure 3. Blocked arrival time and buffer utilization over time for an instance with high variance in the arrival times

mean. So the real question is: Why does using the minimum lead to blocking the arrival for extended periods? The answer lies in the increased buffer fill levels, that are caused by eagerly taking blocks from the arrival stack. What can happen is that all buffer stacks are full and none of the top blocks are ready. This causes gridlock and we have to block the arrival stack until one of the top blocks becomes ready.

Figure 4 shows an example out of the simulation run depicted in Figure 3, where not all the stacks are full but blocking the arrival is still unavoidable. Only the two yellow blocks are ready for delivery and the next block to be removed is B64. The number in parenthesis next to the name of each stack is the remaining capacity for this stack. If we were to insert B102 now, the four blocks above B64 have nowhere to go because then the combined remaining capacity of buffer 1 and 6 is only three. This would generate the deadlock we described before. So our only option is to block the arrival until we performed all the necessary relocations and the removal. Situations like this do not happen as easily when using the mean as our estimate because the solver can focus on removals and manages to keep the buffer utilization down. Using the first quartile causes higher buffer utilization but the solver still manages to remove enough blocks to avoid disaster. But constantly underestimating the time until the next block arrives forces the solver to prefer an insertion over a removal which is fine when there is enough buffer capacity available but can be a critical mistake if not.

5. Conclusions

We developed a simulation environment for a dynamic stacking problem with uncertainty based on a real-world problem in the steel industry. This enabled us to show that increasing the variance in the arrival interval has a large impact on the performance of our solver. For the real-world problem this means that providing the optimizer with more precise information about the arrival times can enable the solver to improve system performance. If that is not possible we showed that we can combat the higher uncertainty by using a more conservative estimate of our random variable. In the case of the arrival times switching the estimator from using the mean to using the first quartile was shown to reduce the time the arrival stack was blocked. We also showed that being overly cautious and using the minimum as an estimate in an attempt to prevent blocking the arrival stack backfires because it means we are running into the capacity limits of our buffer stacks. The solver must carefully balance insertions and removals to achieve good performance and this balance also depends on the degree of uncertainty. It is also very important for the robustness of the solver to handle edge-cases like the one we showed in Figure 4.

In this paper we varied the coefficient of variation and method of estimation of only a single random variable out of six and still observed complex interactions between the simulation, the solver and our KPIs. There is still a lot of research to be done on this fascinating problem.

Acknowledgments

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.

References

Akyüz, H. and Chung-Yee, L. (2014). A mathematical formulation and efficient heuristics for the dynamic container relocation problem. *Naval Research Logistics (NRL)*, 61(2):101–118.

Beham, A., Kronberger, G., Karder, J., Kommenda, M., Scheibenpflug, A., Wagner, S., and Affenzeller, M. (2014). Integrated simulation and optimization in heuristiclab. In *Proceedings of the 26th European Modeling and Simulation Symposium EMSS 2014*, pages 418–423.

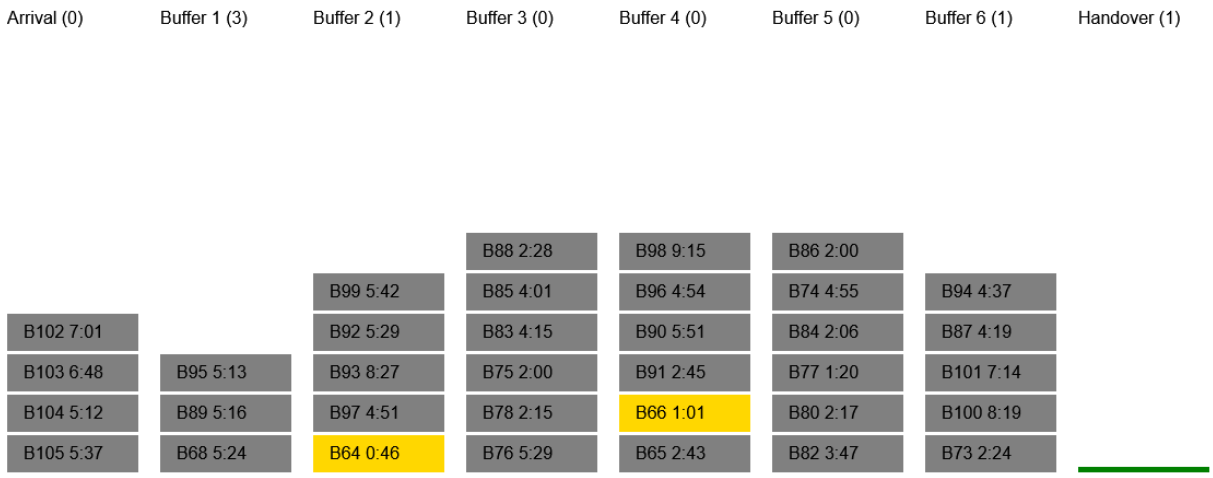

Figure 4. Example of a situation where the arrival is blocked because of high buffer fill-level.

Beham, A., Raggl, S., Wagner, S., and Affenzeller, M. (2019). Uncertainty in real-world steel stacking problems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1438–1440.

Galle, V., Manshadi, V. H., Boroujeni, S. B., Barnhart, C., and Jaillet, P. (2018). The Stochastic Container Relocation Problem. *Transportation Science*, 52(5):1035–1058.

Kang, J., Ryu, K. R., and Kim, K. H. (2006). Deriving stacking strategies for export containers with uncertain weight information. *Journal of Intelligent Manufacturing*, 17(4):399–410.

Ku, D. and Arthanari, T. S. (2016). Container relocation problem with time windows for container departure. *European Journal of Operational Research*, 252(3):1031–1039.

Nguyen, T. T., Yang, S., and Branke, J. (2012). Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, 6:1–24.

Raggl, S., Beham, A., Tricoire, F., and Affenzeller, M. (2018). Solving a real world steel stacking problem. *International Journal of Service and Computing Oriented Manufacturing*, 3(2/3):94.

Raggl, S., Beham, A., Wagner, S., and Affenzeller, M. (2020). Solution Approaches for the Dynamic Stacking Problem. In *GECCO '20: Proceedings of the Genetic and Evolutionary Computation Conference Companion*. (accepted).

Rei, R. and Pedroso, J. P. (2013). Tree search for the stacking problem. *Annals of Operations Research*, 203(1):371–388.

Tricoire, F., Scagnetti, J., and Beham, A. (2018). New insights on the block relocation problem. *Computers and Operations Research*, 89:127–139.

Zehendner, E., Feillet, D., and Jaillet, P. (2017). An algorithm with performance guarantee for the Online Container Relocation Problem. *European Journal of Operational Research*, 259(1):48–62.