Crane Scheduling for a Plate Storage

Jesper Hansen Jens Clausen

January 24, 2002

Abstract

Odense Steel Shipyard produces the worlds largest container ships. The first process of producing the steel ships is handling arrival and storage of steel plates until they are needed in production.

This paper considers the problem of scheduling two cranes that carry out the movements of plates into, around and out of the storage. The system is required to create a daily schedule for the cranes, but also handle possible disruptions during the execution of the plan.

The problem is solved with a Simulated Annealing algorithm.

1 Introduction

We first describe the plate storage at Odense Steel Shipyard and the resulting planning problem in section 2. This is compared to work on related problems in section 3. Afterwards we introduce the complex objective function of this problem in section 4 and our heuristic for solving the problem will be explained in section 5. We end with a section on achieved results and a conclusion.

2 The Problem

The plate storage is organized in 8 times 32 stacks. Approximately 20 plates are stored in each stack on average. $\frac{1}{4}$ of the stacks are used for special purpose plates for instance stacks with identical plates. The rest of the storage are used for plates which for the main part have different sizes. Each plate is ordered for a specific purpose and it is known at which date it is needed in production. Changes in the overall plan for the yard, can however influence the due dates of the plates on the storage, which means that the due date of a plate can change several times while it is in storage.

Two gantry cranes carry out the movements of plates. The cranes share tracks and hence can not pass each other. For each of the 32 rows the cranes can reach each of the 8 stacks. The cranes are equipped with an electromagnet which can lift only the top plate from a stack and move it to some other stack or remove it from the storage. Since the cranes are gantry cranes there is a limit on the height of the stacks.

The plates arrive by ship in bulks and are put in 8 arrival stacks by a dedicated crane and are then moved into the storage by the gantry cranes.

Each day a set of plates must leave the storage to be cut up. The order in which the plates are leaving is of no importance. The plates are put on a conveyer belt with a capacity of 8 plates, which is called the exit-belt. The exit-belt works as a queue and plates are drawn from the queue with time interval λ. We assume in this paper that λ is a constant, but in reality it is a function of the dimensions and order of the plates put on the exit-belt. An illustration of the storage can be seen in figure 1 on the following page.

If a specific plate is requested, which is not on the top of a stack, the plates on top must be moved to other stacks before the requested plate can be removed from the storage. This procedure introduces a number of *unproductive* movements, which we call *dig-up movements*.

The planning task is now to create a schedule of movements for the 2 cranes without collision, that delivers the plates needed for the given day and minimizes costs.

3 Related Research

A lot of different problem classes within scheduling and routing have been studied in the litterature. Many of them have similarities with the problem considered in this paper. One problem is the Travelling Salesman Problem (TSP) where a salesman must visit a number customers or rather nodes

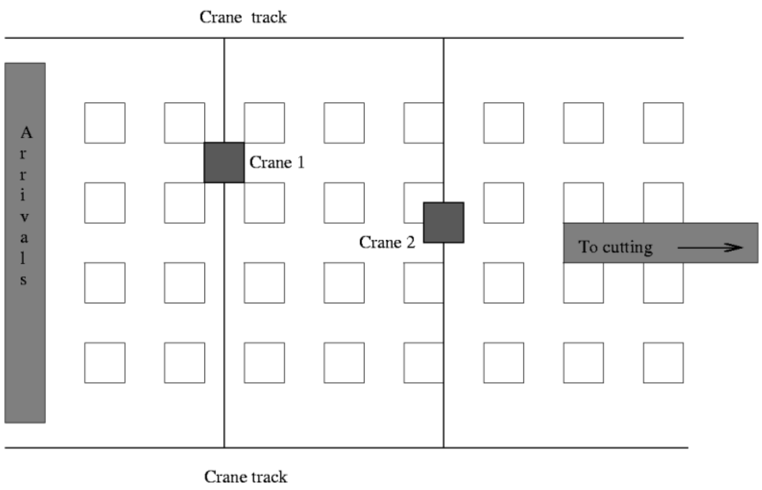

Figure 1: Illustration of the storage layout.

in a graph. For the Stacker-Crane Problem (SCP) a crane must move items from a node to another node in other words visit a subset of arcs in a graph. In our problem the items are steel plates. If the problem includes M cranes or salesman we call it M-SCP or M-TSP. The SCP was introduced by Frederickson et. al. in [1].

If k plates is to be moved from a stack s_o to other stacks it means that k arcs from s_o must be visited, but in an order such that the arc corresponding to the top plate must be visited before the arc corresponding to the next plate and so on. I.e. We have precedence relations between the arcs leaving s_o. Many scheduling and routing applications have precedence relations for instance pickup and delivery of items. If more items can be picked up before delivering, the problem is in the category of Vehicle Routing Problems (VRP). For instance if customers in the M-TSP problems have a specified demand and the salesmen or vehicles have a limited capacity then the problem is called the VRP.

In our application a plate may be moved more than once if it is put on a stack from which plates are going to be moved later on. This means that extra arcs that must be visited by a crane are dynamically inserted in the graph. Insertion of extra arcs are depending on the order in which the arcs are traversed and we have precedence relations between arcs corresponding to movements of the same plate.

Another complicating factor is the the destinations for the dig-up movements are decision variables. In the graph setting it means that the head of an arc from node s corresponding to a dig-up movement from stack s is not given, but is part of the optimization problem.

In our application the two gantry cranes share tracks and in our schedule we must avoid collisions of the two cranes. Problems of scheduling cranes and robots have many applications and some work have been done in this area. Fujita et. al. [2] are solving a very similar problem of scheduling overhead cranes transporting coils. Differences in the models are mainly how the robots or cranes move. For instance in the Hoist Scheduling Problem (HSP) the hoists are moving on a line and cannot cross each other on that line. Gantry cranes share tracks and cannot cross each other as well and the gantry normally only have one hoist, but could have more that as well cannot cross each other. In order to avoid collision of the cranes extra *positioning* and *wait movements* may have to be inserted in the sequence of movements. In other cases crane or robot arms can only move around its centre where they share some common space where collision of the arms must be avoided. See Chung-Yee et. al. [3] for an overview of current trends in scheduling.

4 Objective Function

Costs include salary to the crane operators, power for the cranes to move and activation of the electro-magnetics as well as maintenance on the cranes. In other words, make-span of the schedule, total traveled distance and number of movements. Minimization of the time when the last plate is put on the exit-belt is also important to be able to start the following processes as early as possible. One days production is planned at a time, so in order to minimize the long term costs, we measure the quality of the *state* of the storage after each day. The state is evaluated by 3 different criteria:

1. How well are the stacks sorted by plate due date? More specifically, how many dig-ups have to be done for each stack with the given plate due dates. For stack (2,4) in figure 2 on page 6 plate p_2 is the first

to be removed from the storage on the November 1. This requires the dig-up of p_1. The next plate to be removed from the stack is p_4, which causes the dig-up of p_3. The result is 2 dig-ups.

We have defined the cost of a digup to be the estimated cost of power used for lifting and dropping a plate plus the cost of power for moving the crane to the nearest stack and back plus the cost in salary for operating the crane in the amount of time according to the digup movement.

2. How close are the plates to the exit-belt compared to the due date of the plates? When minimizing traveling distance it is better always to move plates in direction of the exit-belt and when the due date of a plate is close in time, we want it to be close to the exit-belt. More formally the cost for a given plate is:

$$\frac{\textit{distance to exitbelt}}{\mathrm{diff}(\textit{plate due date},\ \textit{today}) +\ 1}$$

where $\mathrm{diff}(date_1, date_2)$ returns the difference in days between two dates. Let us assume that the exit-belt is located at the coordinates (1,4), then stack (2,4) is 1 closer than (3,4). If for instance today is 01/11/2001 then the cost for plate p_1 is $\frac{1}{4}$, for p_4: $\frac{1}{6}$ and p_6 it is $\frac{1}{3}$.

The above exit distance can be converted to time and the cost is then achieved by multiplying with the salary of the operators.

3. How many plates are there in each stack ? We want the plates to be as evenly spread out on the stacks as possible, which will result in less dig-ups when changes in due dates occur. Generally the cost for a stack is:

$$\frac{(\textit{no. of plates in stack})^2}{\textit{max. no. of plates in stack}}$$

where the maximum no. of plates in a stack is a user supplied upper limit on the number of plates common for all stacks. We observe that the cost per stack will be in the interval from 0 to the maximum no. of plates in the stacks. If we have an upper limit of 10 plates, then the stacks in figure 2 will have a cost of $\frac{16+4+4}{10} = 2.4$ while if p_1 was moved to stack (3,4), the cost would be $\frac{9+9+4}{10} = 2.2$, and hence better.

What we are minimizing is in some sence the worst case number of digups if a plate in the bottom of a stack is requested. The cost of a digup in 1 is therefore used here as well.

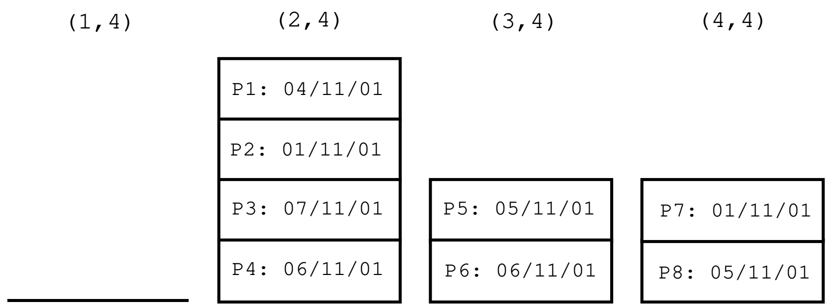

Figure 2: Example of plates in stacks.

We have managed to convert all criteria functions into a common unit, which is then minimized. All the user has to supply are the operating costs of the crane and the salary of the operators.

5 Solution Procedure

Each day we know which plates must leave the storage. These movements of plates out of the storage are called *exit movements*. Given these movements we can identify plates that have to be dug up during that day. Note that these plates can be identified before scheduling any movements.

When plates arrive by ship they must be moved into the storage. Given a day with arrival of plates we assume that all plates are put in 8 stacks outside the storage. This means that we can generate the *arrival movements* before scheduling just as for exit movements. Note that it has not yet been decided to which stacks the plates will be moved, but only that they *will* be moved.

In order to improve the state of the storage, we may perform some sorting of the stacks. Several different sorting strategies may be applied, for instance:

1. Generate movements for all plates in the stack, if it is not sorted. This will result in movement of all the plates in stack (2,4) in figure 2 and none from stack (3,4) and (4,4).

2. Generate movements for plates from the top of the stack until the stack is sorted: The plates p_1, p_2 and p_3 will be moved.

3. Find the plate with minimum due date - in case of ties choose that closest to the bottom . Generate movements for all plates above this plate with later due date: p_1 will be moved.

4. As 3., but now we only generate movements, if the minimum due date is no later than δ days from today. If for instance δ is 1 and today is the 31/10/2001 then p_1 will be moved. If today is before the 31. October then none will be moved.

5. Generate movements for all plates in the stack, if the stack sort cost incurred by the state of the stack is larger than some given value.

We call these movements for *sort movements*. The reason for doing sorting is that in the future less dig-ups will be necessary, and less time will be spent removing plates from the storage. If the due dates of the plates change frequently there is a risk of doing a lot of sorting that later turn out to be wasted. Experiments indicate that strategy 4 with $\delta = 1$ is a good choice, since no dig-ups will be necessary the day after if the due dates are not changed.

We need to be a bit more specific about the term *movement*: A movement is specified by a plate, origin and destination stacks and a crane. In other words it is a specific crane moving from its current position to the origin stack to pick up the plate, afterwards moving to the destination stack, where the plate is dropped. Below we see an example of a movement of plate p from (x, y) to (x', y') as a sequence of *operations* for a crane:

1. Move crane from current position to (x, y).

2. Lift plate at current position.

3. Move crane from current position to (x', y').

4. Drop plate at current position.

In some cases it will be necessary for a crane to move away from the other crane to avoid collision or wait for the other crane to finish some operation. In that case the above described movement will be split up into several distinct movements and special types of movements called *position* and *wait* movements will be inserted. Below, we give an example:

1. Move crane from current position to (x, y).
2. Lift plate at current position.
3. Move crane from current position to (x'', y'') to avoid collision.
4. Wait t seconds.
5. Move crane from current position to (x', y').
6. Drop plate at current position.

The above sequence is actually divided into 3 movements:

1. Move plate p from (x, y) to (x'', y'').
2. Wait t' seconds.
3. Move plate p from (x'', y'') to (x', y').

Here the crane should not drop and again lift the plate at position (x'', y'').

Earlier we described how to identify plates that have to be moved on a given day. Given the precedence relations that a plate on top of another has to be moved first, we have a partially ordered set of movements. Assume that we in some way have divided the movements into two sets (one for each crane), and found an ordering of the movements in each set which fulfills the precedence relations. In order to determine start- and end-times of the movements, we simply simulate the movements while inserting wait, positioning and extra dig-up movements as necessary.

In figure 3 on the following page we see an activity diagram of the process of simulating a movement for a crane. The cranes work in parallel, so the simulator must keep track of which crane first finishes its operation and then determine the next operation for that particular crane. When entering the box the first activity can be any activity with a star (*) depending on the previous operation done by the crane. If for example the last operation was lift then the next is to go to the destination of the movement. In choice situations, arrows with no legend cover those situations not described by any other legend.

If a collision of the cranes would occur we handle this as depicted in figure 4 on page 10. If possible the crane moves closer to the other crane, which

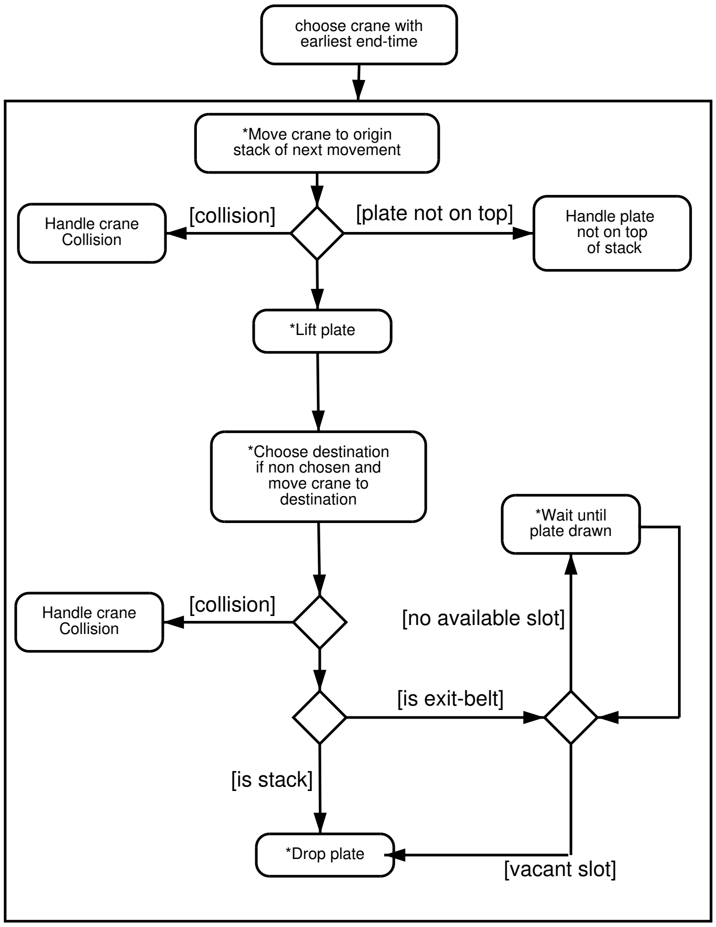

Figure 3: Simulation of a plate movement.

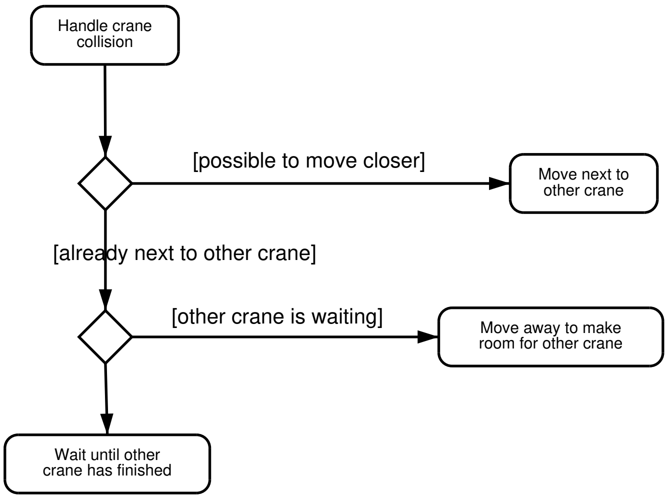

Figure 4: Simulation of a plate movement.

will always be closer to the destination of the crane. If the crane is already next to the other crane we either wait or move away for the other crane to finish its operation.

The case when the requested plate is not on top of the stack is shown in figure 5 on the following page. A dig-up of the top plate is done by the crane, unless the current movement of the other crane is to dig-up the top plate. In that case we wait for the other crane to take the plate.

If the plate is not in the stack it means that it has not yet arrived from some other stack. If the current movement of the other crane it to move the plate to the stack, then we wait for the other crane to finish. Otherwise we remove the movement from the sequence as well as any following movements of that plate if any exists. Note that the movement cannot be an exit-movement, since plates involved in exit movements are always present in the origin stack of the movement.

For movements other than exit-movements we have to decide to which stack to move the plate. This is done just after lifting the plate. All stacks are evaluated with regard to several criteria:

1. Preferably we want to put the plate on a stack where the plate with

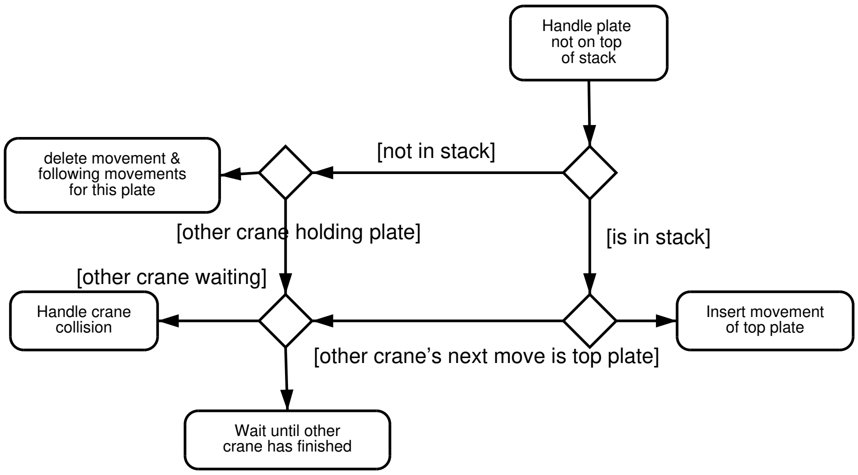

Figure 5: Simulation of a plate movement.

the minimum due date has a due date that is later than or equal the due date of the plate we are moving. In that case the stack sorting will not deteriorate. On the other hand we want the difference to the minimum due date in the stack to be as small as possible, because this will make it easier to find suitable stacks for other plates.

For example if we in our sequence are moving plate p_7 followed by p_8 in figure 2 on page 6, we have to decide to which stack of (2,4) and (3,4) to move p_7. The plate p_7 will not deteriorate the sorting of either stacks in (2,4) or (3,4), but it is still better to move it to (2,4): If p_7 is moved to (3,4), then moving p_8 will cause an extra dig-up at a later stage.

2. Minimizing traveled distance to the stack and further on to the source stack of the next movement. For this criteria (3,4) is better than (2,4), since the crane afterwards will move back to (4,4) to pickup p_8.

3. Minimizing stack height. Again (3,4) is better than (2,4) according to the stack height criteria on page 3.

4. Minimizing distance to the exit-belt. Here (2,4) is better than (3,4) since stack (2,4) is closer to the exit-belt placed in (1,4).

The weighting of these criteria is the same as for the objective function. A random stack is chosen from the best set of stacks.

The simulation ends when all movements have been executed. The schedule can now be evaluated according to the criteria mentioned earlier.

As earlier noted the problem is solved over a period of time, but one day at a time. We have used a standard Simulated Annealing (SA) algorithm as in [4]. The reason for using a SA compared to for instance a Genetic Algorithm or a Tabu Search is mainly that evaluating neighbour solutions is quite expensive, since it involves simulating the crane sequences every time.

Given our solution of two sequences of movements, we only need to specify a way to move to a new neighbourhood solution. Our neighbourhood structure is given by 3 simple *operators*:

- Delete a specified destination for a movement and any following movements of that plate. A new destination will then be picked when simulating.
- Reinsert a movement from a crane sequence at another position in the same sequence.
- Delete a movement from one crane sequence and insert it in the sequence of the other crane.

We only allow neighbour solutions where the precedence relations are fulfilled. For operator 2 we have two cases:

1. If the predecessor/successor are in the same sequence, we just check if the predecessor is earlier in the sequence than the successor.
2. If the predecessor/successor are in separate sequences, the solution will always be legal, since the crane doing the succseeding movement just waits until the other crane has finished the preceeding movement. This can obviously result in poor solutions. Therefore we try to estimate the new start and end times of the movements and if waiting is introduced we reject the neighbour solution.

For operator 3 the two cases are the same, but turned up side down.

6 Results

At present the storage at Odense Steel Shipyard is managed in a block-oriented fashion. A block is a large section of the ship and all the plates that are needed for producing this section is placed in the same stacks on the storage. A block is produced over a period of time, and the plates are therefore also due over a period of time. Hence searching for the plates involves a huge number of dig-ups every day. For comparison we have implemented the block-oriented approach as well as the approach presented in this paper.

For the test scenario presented in this paper the storage is 24×8 stacks with approximately 3600 plates of which 121 are due each day. Plates arrive every 3 days. The scenario runs over 30 days and initially the plates are placed according to the block-oriented approach. Several other test scenarios have been tested, but the results are similar to the one shown here. In figure 6 on the following page "block" indicates the block approach and "selfreg" the approach described in this paper. "dist" indicates that *disturbances* or disruptions in the plate due dates has been introduced. In our scenario 0-10 exit plates swap due dates with other plates on the storage each day and every 5 days 500 plates shift due dates. Generally the new approach is much better than the old. Looking at number of movements the conclusion is even more clear as shown in figure 7 on page 15. On average the improvement in number of movements is around 50% and in make-span around 40%.

The time to plan one days production is approximately two hours in the prototype implementation of the method.

In table 1 on the following page results are shown when increasing the number of exit-movements to leave the storage each day from 121 to 169 and 225. The times are in hours per day in average over 30 days. With the size of the storage not changing you would expect the amount of time spend and total number of movements to increase faster than linear since more plates must be placed in each stack. This is also the case since a plate must be moved 4.57 times on avarage for 121 plates leaving the storage, 5.29 times for 169 plates and 5.88 for 225 plates. This must be compared to the 11.12 movements on average for the block principle. All numbers are found for the case with due date disturbances.

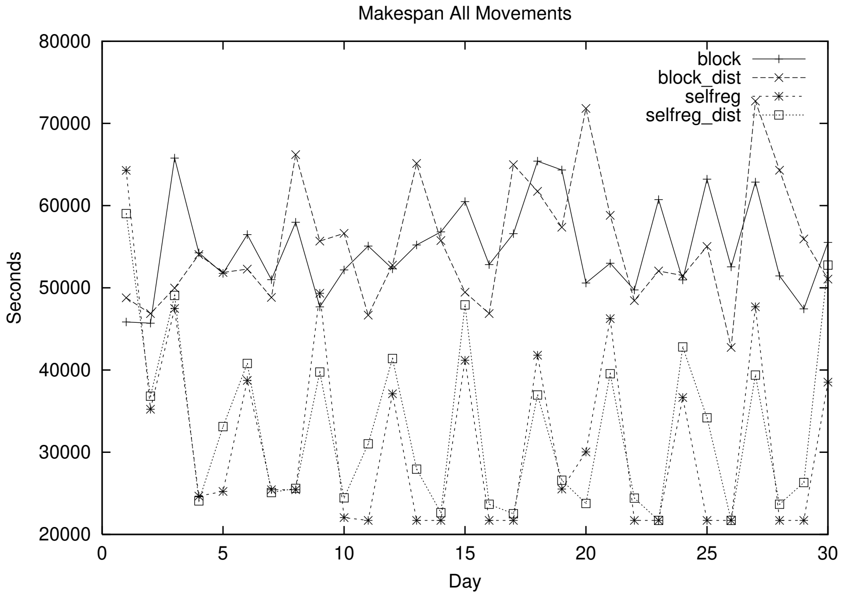

Figure 6: Makespan for block and new appoach.

Principle	Exit	Dist.	Exit Time	Total Time	Movements
Block	121	No	6.03	15.24	1332
Block	121	Yes	6.56	15.34	1346
Selfreg	121	No	6.06	8.72	441
Selfreg	121	Yes	6.24	9.15	553
Selfreg	169	No	8.61	13.97	748
Selfreg	169	Yes	8.92	14.41	894
Selfreg	225	No	12.13	20.87	1250
Selfreg	225	Yes	12.47	20.27	1322

Table 1: Results when increasing number of exit-movements.

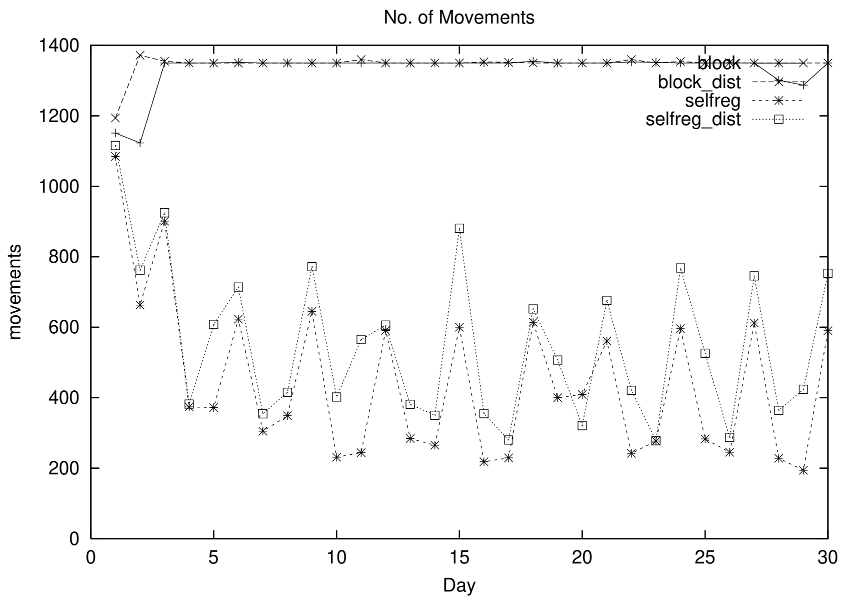

Figure 7: No. of movements for block and new appoach.

7 Conclusion

We have considered a hard real life application. Our pilot results are extremely promising, but we believe that even better solutions can be achieved by refining the heuristics.

Currently research focuses on enhancing the planning software with facilities to handle disruptions dynamically for due dates, break down or changes in operation times of cranes and exit-belt. Here efficiency of the computations has crucial importance, and hence another important line of work is to reduce the running time of the method.

References

[1] G. N. Fredrickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some routing problems. *SIAMM Journal of Computing*, 7(2):178–193, 1978.

[2] F. Fujita, M. Okada, and T. Kaita. Automatic scheduling system for an overhead multi-crane yard. In *IFAC Control Science and Technology 8th Triennial World Congress*, 1981.

[3] C.-Y. Lee, L. Lei, and M. Pinedo. Current trends in deterministic scheduling. *Annals of Operations Research*, 1997.

[4] P. J. M. van Laarhoven. Theoretical and computational aspects of simulated annealing. Technical report, Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands, 1988.