Math Meth Oper Res (1999) 49:501–515

Mathematical Methods
of Operations Research
© Springer-Verlag 1999

Order picking in an automatic warehouse: Solving online asymmetric TSPs

Norbert Ascheuer, Martin Grötschel, Atef Abdel-Aziz Abdel-Hamid

Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Takustr. 7, D-14195
Berlin-Dahlem, Germany (e-mail: ascheuer@zib.de)

Abstract. We report on a joint project with industry that had the aim to sequence transportation requests within an automatic storage system in such a way that the overall travel time is minimized. The manufacturing environment is such that scheduling decisions have to be made before all jobs are known. We have modeled this task as an *online* Asymmetric Traveling Salesman Problem (ATSP). Several heuristics for the online ATSP are compared computationally within a simulation environment to judge which should be used in practice. Compared to the priority rule used so far, the optimization package reduced the unloaded travel time by about 40%. Because of these significant savings our procedure was implemented as part of the control software for the stacker cranes of the storage systems.

Key words: Traveling Salesman Problem, online-algorithm, automatic storage system

1 Introduction

Designing and controlling Flexible Manufacturing Systems (FMSs) provides a wide application field for methods of Combinatorial Optimization. FMSs are very complex, in general. As a consequence, it is (at least at present) impossible to optimize such systems on the whole. Rather, subproblems are identified, modeled mathematically, and optimization algorithms are developed and implemented for these special tasks. See, among others, [14, 15] for surveys. One hopes that the chosen decomposition of the global FMS optimization problem provides acceptable solutions – at least in practice.

Manuscript received: November 1998

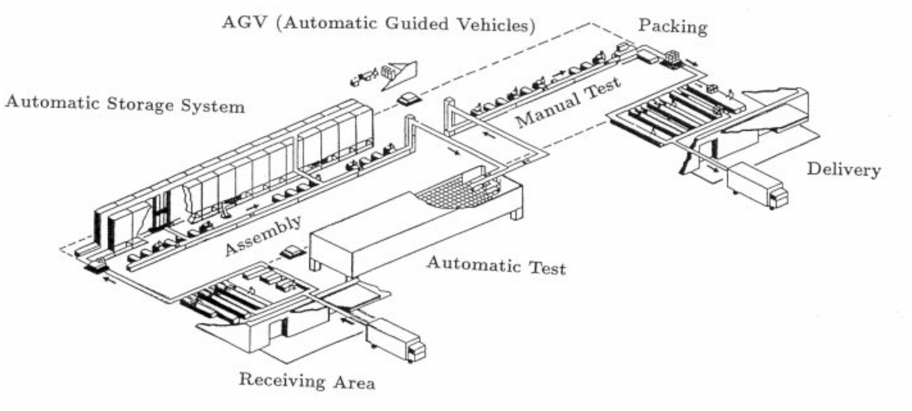

Fig. 1.1. Sketch of the factory layout

The FMS we consider here is part of the production plant of “Werk für Arbeitsplatzsysteme” of Siemens Nixdorf Informationssysteme (SNI) in Augsburg, Germany. This factory is designed to assemble and test a variety of products such as personal computers, keyboards, monitors, etc. See Figure 1.1 for a partial sketch of the factory layout.

After the opening of the plant, the production volume increased and the warehouses turned out to be bottlenecks for the material flow within the FMS. Especially after technical or software malfunctions, transportation tasks started to pile up. It took long to process all tasks; the required parts were often not supplied in time to the assembly lines. This made either manual interference necessary or resulted in a production delay.

The automatic storage systems

The automatic storage system we consider at SNI contains six warehouses, each consists of a two-sided aisle and is equipped with a computer-controlled stacker crane. Each warehouse is divided into an upper and a lower part. The lower part contains the material supply locations (called buffers) for the adjacent assembly line, while the upper part serves for storage purposes (of short to medium term). Material is stored either on pallets, box pallets, oversized “special-purpose pallets”, or in tote-boxes of two different sizes. To simplify notation we will just talk about *containers* whenever we address any of the different types of transportation units. The input- and output docks are located at one end of the aisle. Figure 1.2 shows a side-view, and a view from the top, respectively, of the rectangular warehouse that we consider. (An AGV is an automatically guided vehicle transporting containers to and from the warehouse.)

A single stacker crane can travel between the two ends of the aisle of the warehouse. It is capable of carrying one container at a time and travels both horizontally and vertically. The main function of the stacker crane is to transport the containers within the warehouse. Such transportation tasks, e.g., are storage tasks (material is moved from the input dock to some storage lo-

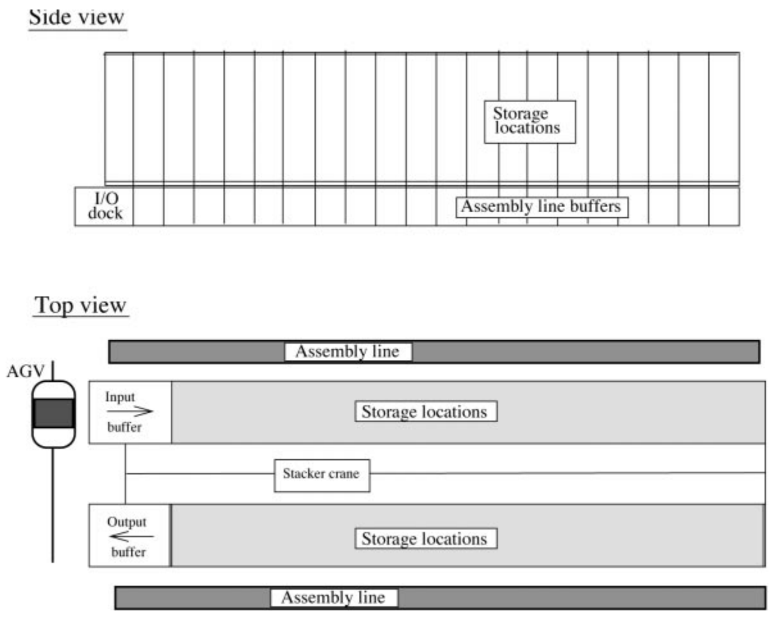

Fig. 1.2.

cations), retrieval tasks (material is moved from some location to the output dock), buffer supplying tasks (material is moved from a storage location to a buffer), etc. More details about the system can be found in Abdel-Hamid [1] and Ascheuer [3].

The engineers, observing the performance of the stacker crane in practice, had the feeling that too many "long" unloaded moves were executed resulting, at times, in unnecessarily long waiting times at the assembly line. After a detailed analysis of the system and a careful consideration of possible alternative objectives it was decided to develop an optimization tool aiming at the minimization of unloaded travel times with the hope that this would improve the overall performance of the storage system. We thus concentrated on the *sequencing problem* described below.

The sequencing problem

At an arbitrary point in time the stacker crane is idle or performing some transportation task. If it is not idle the control system typically maintains a *pool* of transportation tasks (called *orders* from now on) that have been generated and whose execution has not started yet. The original control system used a priority rule to choose an order from the pool. More precisely, each type of order (storage, retrieval, etc.) is assigned to a certain priority class. The orders with highest priority are considered first. Within one priority class the orders are executed according to the First-In-First-Out (FIFO) principle. An analysis of the system showed that the priority based strategy resulted in long unloaded moves between the orders. In total there existed six priority-classes at SNI. Under heavy load conditions, the order-pool contained 10–15 tasks on the average and up to 50 orders maximally.

We proposed to replace this rule by the following approach: Whenever an order is generated or modified, schedule the currently available orders in such a way that the total crane utilization time is as short as possible. This utilization time consists of three components:

- Total loaded time (the time the crane travels with loads to execute an order),
- total unloaded time (the time the crane travels without loads between two orders), and
- transfer time (the time for the crane to load or unload a container at the I/O-dock, at a storage location, or at a buffer place).

Clearly, the rule used to execute the orders does not affect both the transfer time and total loaded time. Thus, we have to consider unloaded travel time only.

This leads us to consider the following *(two-dimensional) order picking problem*: For each order, compute the time the crane travels unloaded between this order and the other orders in the pool. Schedule these orders such that the sum of times needed for the unloaded moves is as small as possible.

Related problems

Several warehousing problems have been studied in the literature. We briefly survey some of them. For an extensive survey on warehousing problems we refer to van den Berg [22].

One of the most "popular" *order-picking systems* that have been analyzed can briefly be described as follows: We are given a set of parallel double-sided aisles with cross-overs at each side of the aisles. An order consists of items that have to be picked at some position in the aisles. Each order-picker has the capacity to visit a set of pick-positions on one tour. The problem is to find a sequence in which the pick-positions are visited, such that the total time to complete the order is as small as possible. As only horizontal travel-time is considered this leads to a special case of the traveling salesman problem (TSP) that is solvable in polynomial time (see Ratliff and Rosenthal [20] and Van Dal [21], among others).

More complex situations arise when one considers a single-aisled, double sided Automatic Storage and Retrieval System (AS/RS). The stacker crane has to pick orders, but now horizontal and vertical travel time is considered. Depending on the capacity of the crane either storage and retrieval request have to be combined or an optimal route through several pick positions has to be calculated. The latter problem can be modeled as a TSP in the plane. The scientific literature mainly focuses on the development of analytic expressions for calculating the expected travel time for command cycles under the assumption that the retrievals are sequenced due to a certain strategy (e.g., FIFO, nearest-neighbor heuristic). These expressions are then evaluated in a simulation environment with randomly generated data (see [13, 16, 19], among others).

Burkard et al. [9] consider a system with seven parallel aisles, crossovers at

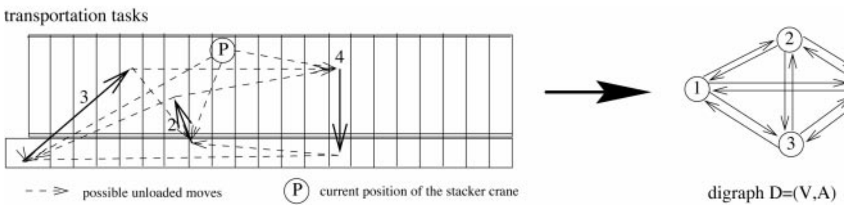

Fig. 2.1. Modeling the sequencing problem

one end of the aisle and three stacker cranes. As each stacker crane is capable of moving into each aisle, the cranes might block each other. Their aim was to develop a control-system that guarantees a "conflict-free" routing of the cranes, minimizing the stacker cranes' idle movements and waiting times. Their solution approach is to build up a decision tree for a restricted time horizon. In that tree branches correspond to taken decisions and are weighted by the seconds a stacker crane remains idle caused by that decision. The solution corresponding to a shortest path from the root to a leaf is accepted as the current control for the next time period.

2 Modeling

We introduce a directed graph $D = (V, A)$ with $V = \{1, \dots, n\}$ where nodes $2, \dots, n$ represent the orders in the pool. Node $1 \in V$ corresponds to the current task the stacker crane is performing, or to the current position of the stacker crane in case it is idle. An arc $(i, j) \in A$ represents the possibility to execute task j directly after task i. (For technical reasons some arcs (i, j) may not be present.) With each arc $(i, j) \in A$ we associate a weight c_{ij} that indicates the cost of executing task j after task i. This cost-coefficient equals the time the crane travels without load between the destination of task i and the source of task j. As the stacker crane stops at the position where the last order of the pool is achieved we set $c_{i1} = 0$. Otherwise the arcs $(i, 1)$ $\forall i \in V \backslash \{1\}$ are weighted with the travel time to the dwell point, i.e., the point where the stacker crane is positioned strategically whenever it is idle. Typically, $c_{ij} \neq c_{ji}$ holds. Figure 2.1 illustrates the modeling procedure on a small example with three transportation tasks 2, 3, 4 (represented by solid arrows). The point P designates the current stacker crane position and corresponds to node 1 of the associated digraph.

It is easy to see that an optimal solution of the *asymmetric traveling salesman problem* (ATSP) on this directed graph $D = (V, A)$ corresponds to a sequence of orders of the pool such that the total unloaded time is as small as possible.

Online aspects

The model described above is only a snapshot of the production stage at a certain point in time and does not take the dynamic changes of the system into account. In reality, the order pool is not static, it is altered over time and,

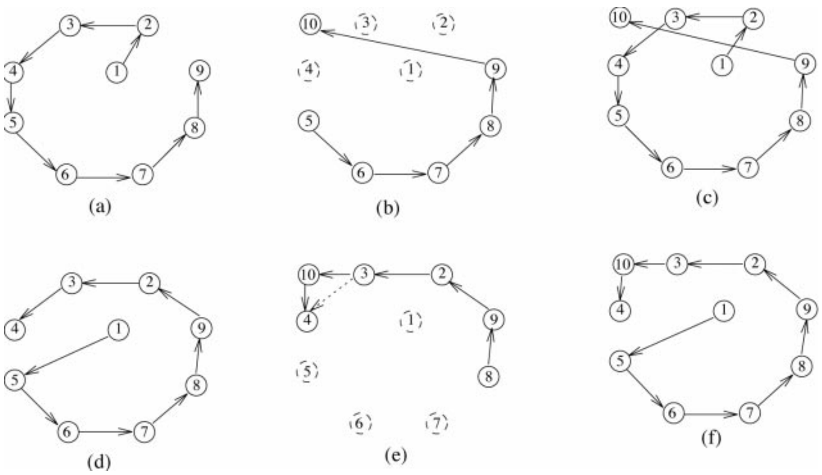

Fig. 2.2.

thus, the ATSP is changing dynamically as follows. Suppose we have an order pool and that the orders of this pool are sequenced optimally by solving the corresponding ATSP. While executing this sequence new tasks for the stacker crane are generated. It is not known a priori at what time what type of order is generated. As soon as a new task is generated the control system has two possibilities:

(a) Either, to perform the sequence as it was calculated and to collect the generated tasks in a new pool to be optimized after all orders of the current pool are executed (this is sometimes called IGNORE strategy), or
(b) to update the sequence each time a new task is generated, which is a REPLAN strategy.

In agreement with the management we decided to follow the second approach. This implies that as soon as a new task is generated, a new sequence has to be calculated. Obviously, this new sequence consists of the new task and the tasks of the old pool that have not been executed yet.

In the following example the dynamic nature and the arising difficulties concerned with this are illustrated.

Example (2.1): Let us consider a pool containing 8 orders to be performed (cmp. Figure 2.2). The tasks are labeled 2 to 9. Node 1 represents the initial position of the stacker crane. Now, the corresponding ATSP is solved to optimality and the corresponding shortest sequence starting with node 1 is shown in Figure 2.2(a). We start performing the tasks. After t time units tasks 2, 3, and 4 are already performed, 5 is executed, and a new task 10 is generated. A new sequence starting at node 5 and visiting nodes 6–10 has to be calculated. Again, the optimal sequence is calculated (Figure 2.2(b)). The resulting overall travel path is given in Figure 2.2(c).

Now, suppose that, in the initial step, we made a mistake and followed the non-optimal path shown in Figure 2.2(d). Again, after t time units the new task 10 is generated that has to be achieved. If the new node associated with task 10 is inserted best possible as shown in Figure 2.2(e), then the overall travel path is the one shown in Figure 2.2(f). This path is shorter than the one of Figure 2.2(c). □

In other words, computing an optimum solution for every subproblem "on the run" does not necessarily lead to an optimum sequence of orders of the whole day.

In classical optimization theory one generally assumes that an algorithm has complete knowledge of the input data. In our application, this would imply that the control system needs to have, in the morning, full information about the orders generated throughout the whole day. Although the FMS is computer-controlled, this is not realistic. It is quite unpredictable when containers arrive at the I/O-dock, and the time needed for the manual assembly is not completely predictable either. The control system simply has to take decisions every time that new data occurs without any knowledge about future requests. The algorithm has to decide which order to perform next, based on information on the current order pool only. Once the execution of a task has started, this decision cannot be revised (even if it might later turn out to result in a bad overall performance of the system). In the literature, problems of this type are called *online problems*. Thus, the sequencing problem is in fact an *online ATSP*.

Today's theory (and practice) of online optimization is still far from being satisfactory. It is not clear what type of mathematical models are appropriate for a particular online situation. Even if the modeling issue is resolved, the current theory does not provide reasonable criteria for selecting a solution algorithm. This is also true for our application. We have therefore decided to use simulation, based on real-world data provided by SNI, to compare different solution approaches. See Fiat and Woeginger [10] and Borodin and El-Yaniv [8] for recent surveys on online algorithms; Ascheuer et al. [4, 5] for a discussion of the applicability of competitive analysis to real-world transportation problems; Ausiello et al. [6, 7] for competitiveness results for the online-TSP.

3 Computational studies

The ATSP is known to be an $\mathcal{NP}$-hard problem. For the instance sizes that arise in our application (not more than 50–60 nodes are present at a time) exact algorithms, however, are known that solve the problem to optimality in a reasonable amount of computing time (see [11, 12, 17, 18], among others). In the sequencing problem an ATSP has to be solved as soon as the order pool changes. Note again, that an algorithm that solves each subproblem to optimality is just a heuristic algorithm for the online ATSP (of the whole day).

Computational experiments were performed in order to compare the online behaviour of different heuristics. To this end, we developed and implemented a simulation model and algorithms to solve the online ATSP. Real-world data were provided by SNI. They helped to gain insight into the practical performance of the algorithms and to choose the ones for use in practice.

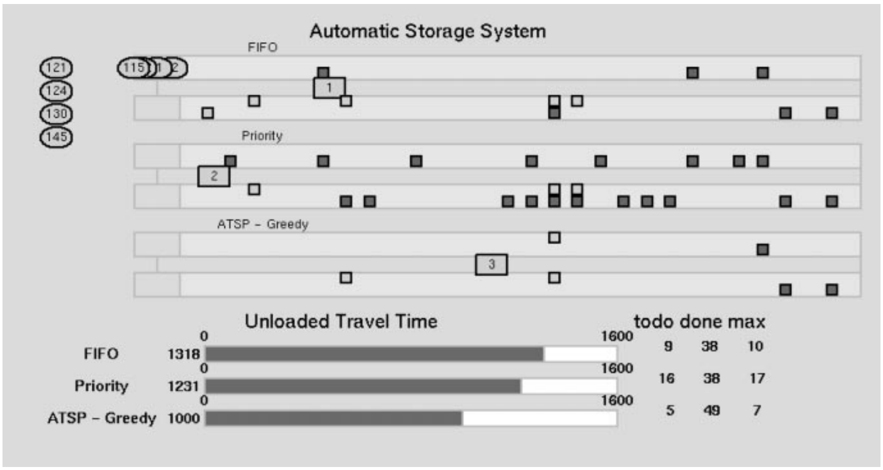

Fig. 3.3. Screenshot of simulation system for automatic storage systems

Table 3.4. Validation results for the simulation-model

	Tasks	ØSNI	ØSIM	ØDEV	MAX DEV	% DEV	Time
1	416	76.27	76.83	2.42	20	0.73	11.47
2	423	75.45	76.34	1.96	11	1.17	11.60
3	403	78.81	78.95	2.01	9	0.18	11.17
4	399	76.19	76.58	2.09	9	0.51	11.08
5	450	77.11	76.73	2.06	20	0.49	12.30

A simulation model

As it is too risky and too expensive to perform experiments with a modified control system in the FMS itself, we developed a simulation program for the automatic storage system. The main aim of this program is to compare the online behaviour of the different heuristics and to convince the engineers of our industry partner that the proposed optimization approach will improve the performance of the system. Therefore, not only a fast, but also an exact simulation program was needed. To achieve this aim we implemented the simulation program using the simulation library AMSEL [2]. See Figure 3.3 for a screenshot of the simulation program.

Furthermore, it was inevitable to perform a (time-consuming) validation of the simulation model. For that reason, for 5 days every generated task and each move of the stacker crane were recorded. This production data served as a basis for our studies. The generation times and the coordinates of the generated tasks were used as input to the simulation program. The moves of the stacker crane in the simulation were then compared to the real moves. The results of the validation process are summarized in Table 3.4.

A transportation task consists of the (unloaded) positioning move, the pick up of a container, the (loaded) move with the container, and the delivery of the container. The average time that was needed for a task on day 1 was 76.27 seconds (see column ØSNI in Table 3.4). This time was obtained by measuring all operations of the day and averaging them out. In total 416 transportation tasks were recorded during day 1. The simulation model produced an average task length of 76.83 seconds (column ØSIM) for this day resulting in 0.73% deviation (column% DEV). Deviations are bound to occur since always some interruptions and manual interactions occur that are not taken into account by the simulation model; see also column ØDEV for the average deviation and column MAX DEV for the maximum deviation in seconds that occurred throughout a day. The CPU-time needed for running a simulation of two working shifts is approx. 12 seconds on a SIEMENS PC MX300; see column TIME. More details can be found in [3]. Our industry partner considered these validation results to be very satisfactory and decided to accept proposals that are based on the use of this simulation model.

Heuristics

As already mentioned, we compared different strategies for sequencing the tasks to the old priority based rule. To be more precise, we embedded several well known TSP-heuristics and an exact branch&bound algorithm into the simulation program. Whenever a new transportation task is generated one of these routines is called and the tasks are performed according to the generated sequence. As soon as the simulation system generates a new task this routine is called again, etc. We used the real life data that was provided by SNI. As the simulation program is considered to model the manufacturing system and the dynamic behaviour of the system exactly enough we are able to compare these approaches and to take qualitative conclusions. We compared the following strategies:

priority:	SNI-priority rule.
random:	Generation of random sequences.
optimal:	Each subproblem is solved to optimality using the branch&bound code by Fischetti and Toth [11].
greedy:	Greedy heuristic.
greedy + 2opt:	Greedy heuristic with additional improvement heuristic (2-opt heuristic).
greedy + 3opt:	Greedy heuristic with additional improvement heuristic (3-opt heuristic).
fit-in:	The newly generated task is inserted in the best possible way into the existing sequence.
farins:	Farthest insertion heuristic.
listins:	List insertion heuristic: The nodes to be inserted in the best possible way are taken in the order $1, 2, \ldots, n$.
randins:	Random insertion heuristic: The nodes to be inserted in the best possible way are chosen randomly.
bestins:	Best insertion heuristic: Choose always the best possible insertion.
shuffle:	Shuffle heuristic: Start with sequences of length 1, "shuffle" them together to sequences of length 2, "shuffle" these together to sequences of length 4, etc.

Note that the main difference between the fit-in heuristic and all other optimization algorithms is that it uses the already existing sequence, whereas the others start the recalculation from scratch.

Normal load conditions

First, we used the original data that was supplied by SNI to compare our optimization strategy (using *optimal*) with the priority rule used so far. The results are summarized in Table 3.5.

The achieved improvements varying between 6 and 8% (see column IMP.%) were rather disappointing. But by analyzing the input data, it turned out that the ATSPs that had to be solved were rather small (see columns MAX. TASKS and ØTASKS). On the average there were only 2–3 tasks present at a time, i.e., on the average there was nothing to optimize. This resulted from the fact that during this week there was a low production volume and the system was working without major breakdowns. Although the overall improvement was not too big the peaks in the order-size were cut off. E.g., on day 5 the maximum number of tasks to be present at the same time reduced from 14 to 9, and the average number reduced from 2.79 to 2.00. This indicates that under heavy load conditions large improvements could be achieved.

Heavy load conditions

As the process of gathering all the necessary data within the manufacturing system is a technically difficult and time consuming process, we and our industry partners decided to use the data already available and to artificially create heavy load conditions. Within the manufacturing system these typically occur after a breakdown of the stacker crane when tasks start to pile up. Thus, we artificially created some breakdowns by modifying the input data for the simulation program.

Key to Tables 3.5–3.7:

TASKS:	Total number of transportation tasks (orders) generated.
TIME:	Average unloaded travel time of the stacker crane (in seconds) using either the priority rule (*priority*) or solving each subproblem to optimality (*optimal*).
MAX. TASKS:	Maximal number of orders in the pool using either *priority* or *optimal*.
ØTASKS:	Average number of orders in the pool using either *priority* or *optimal*.
IMP. %:	Reduction of time for the unloaded moves in % calculated by $\frac{\mathsf{prio}-\mathsf{opt}}{\mathsf{prio}} \cdot 100$, where prio (resp. opt) stands for the average unloaded time using *priority* (resp. *optimal*).

Table 3.5. Minimizing the unloaded travel times (normal load conditions)

		priority			*optimal*			
	Tasks	Time	max. tasks	ØTasks	Time	max. tasks	ØTasks	Imp. %
1	416	20.72	11	3.05	19.00	8	2.08	8.30
2	423	20.42	12	2.76	19.08	8	1.82	6.56
3	403	20.37	11	2.93	18.87	7	2.19	7.36
4	399	20.09	11	2.44	18.49	9	1.91	7.96
5	450	20.91	14	2.79	19.26	9	2.00	7.89

Table 3.6. Minimizing the unloaded travel times (heavy load conditions – artificial breakdown)

		priority	*optimal*	
	Task	Time	Time	Imp. %
1	19	19.21	11.79	38.63
2	10	19.40	13.40	30.93
3	17	17.59	9.76	44.51
4	27	19.59	10.59	35.94
5	20	19.40	11.55	40.47
				38.10

Experiment 1: Static system

First, we inserted a breakdown of one hour and collected all the tasks that were generated during that time period. We only took these tasks into consideration for our optimization process. All other tasks were ignored, namely those generated before the breakdown and those during the execution of the tasks. The results are reported in Table 3.6.

Due to the settings of this experiment the entries of max. tasks and ØTasks are the same for both strategies. Therefore, they are omitted from Table 3.6.

Again we compared the priority rule used at SNI to *optimal*, i.e., we solve each subproblem to optimality. The achieved improvements of 30–45% are very impressive, but one should note that this is a static experiment. As we do not take previously generated tasks or tasks that are generated during the performance of these tasks into account, this experiment does not reflect the dynamic nature of the system. But, nevertheless, it shows that there is an immense potential for optimization approaches.

Experiment 2: Dynamic system

Next, we performed a more realistic experiment to construct heavy load conditions in a dynamically changing environment. In contrast to experiment 1, we consider as well the tasks that are generated before the artificial breakdown takes place. During the breakdown the tasks start to pile up (see column

Table 3.7. Minimizing the unloaded travel times (heavy load conditions – artificial breakdown)

		priority			*optimal*			
	TASKS	TIME	MAX. TASKS	ØTASKS	TIME	MAX. TASKS	ØTASKS	IMP. %
1	51	17.31	36	16.90	11.04	30	13.38	36.2
2	49	19.18	27	10.22	14.54	21	7.33	24.2
3	51	19.35	33	15.34	11.10	27	11.03	42.6
4	49	17.72	39	20.74	9.29	32	15.97	47.6
5	50	19.96	36	20.27	13.09	26	13.72	34.4
								37.0

Table 3.8. Average unloaded travel-time (in sec)

HEURISTIC	1	2	3	4	5	$\sum$
priority	17.31	19.18	19.35	17.72	19.96	93.52
random	19.50	18.36	17.83	17.10	19.84	92.63
optimal	11.04	**14.54**	**11.10**	**9.29**	13.09	59.06
bestins	11.16	14.66	11.86	12.53	12.58	62.79
shuffle	11.82	15.38	11.28	10.76	13.80	63.04
greedy	11.82	15.42	12.04	11.63	**12.20**	63.11
greedy + 3opt	11.51	15.32	12.02	11.51	12.76	63.12
listins	12.73	15.14	11.88	10.98	12.80	63.53
greedy + 2opt	11.33	15.34	12.10	12.25	12.62	63.64
randins	**10.84**	15.44	11.71	11.77	14.11	63.87
farins	12.45	15.10	13.52	10.53	14.00	65.60
fit-in	14.81	17.72	17.62	13.19	17.27	80.61
improvement	37.4%	24.2%	42.6%	47.6%	38.9%	36.8%

MAX. TASKS in Table 3.7). After the breakdown we start processing these tasks. Whenever a new task is generated we update the sequence according to the strategy used. After normal load conditions are reached the simulation run is stopped. The results are shown in Table 3.7.

The achieved improvements of 25–45% are not as good as in experiment 1, but in view of the dynamic changes this was expected to happen. Still immense improvements are possible. We like to mention that the branch&bound-code could solve each ATSP in at most 3–5 seconds, very often it was even faster. This code is fast enough to be used in the control system at SNI.

Experiment 3: Comparison of heuristics under heavy load conditions

Finally, we used the data of experiment 2 to compare the online behaviour of the different heuristics. Table 3.8 summarizes the results.

The numbers in rows 2–13 give the average unloaded travel time in seconds between the tasks if every subproblem is solved with the heuristic given in the first column. Using the priority rule for the first set of data, results in an average unloaded travel time of 17.31 seconds whereas the use of the greedy heuristic results in an average unloaded travel time of 11.82 seconds. The row improvement gives the improvement of the heuristic giving the best results over the priority rule used at SNI (*randins* for data set 1, *optimal* for data set 2, etc.). As in the previous tables, the improvement value is calculated by the formula ($100 \cdot \frac{\text{prio-best}}{\text{prio}}$) where prio stands for the value achieved by the priority rule and best for the value achieved by the heuristic producing the best result. In the last column the results for the five data sets are summed up.

Some interesting observations can be derived from Table 3.8:

- First, there is almost no difference between the priority rule and the generation of random sequences. Compared to all other strategies these provide really bad results.
- Once any of the other heuristics is applied under heavy load conditions, a significant improvement can be achieved. The best possible improvements vary from 25% to 45%.
- *optimal* seems to work best, but it does not always give the best result. This is due to the online phenomenon that was explained in Example 2.1.
- *farins* and *fit-in* are slightly worse than the other heuristics, all other heuristics perform rather similar (see column $\sum$).
- It does not pay to apply additional improvement heuristics to the greedy solution. In fact, the results may even get worse.
- Although *fit-in* is by far the fastest of the heuristics, it can be seen that it gives the worst results. This indicates that it is worth recalculating the whole sequence.

4 Conclusions

Based on the studies presented in this paper we designed and implemented an *optimization package* which aims at minimizing the travel time of a stacker crane. As the stacker crane should never wait until the optimization program finishes its calculation we decided to implement a 3-phase process:

Phase 1: Perform cheapest insertion of the new order (*fit-in*).
Phase 2: Run an iterative cheapest insertion on a random ordering of the current order pool. Then choose the best sequence of Phase 1 and 2 (*randins*).
Phase 3: Solve the ATSP with the current transportation tasks to optimality and replace the old sequence by the optimal one (*optimal*).

First, we apply the *fit-in* heuristic to obtain (with almost no time delay) a starting solution that will be improved in the following phases. In phase 2 we apply a more sophisticated heuristic (*randins*) to obtain a better solution, and if there is still computing time left we solve the subproblem to optimality (phase 3). If a new task is generated while phase 2 or 3 are active the process is stopped and the best sequence calculated so far is being performed. This package is now in use on six automatic storage systems of SNI. The results obtained within the simulation model were confirmed during heavy load periods in everyday production. The experience with the optimization package in the manufacturing system showed that in most cases there is enough time to complete the calculation, i.e., the process terminates phase 3 rather than interrupting in between.

Although in our practical application significant improvements could be achieved, there is still a need to study the online ATSP more deeply. First of all, it is necessary to develop online heuristics that incorporate the dynamic system better than the heuristics do that were designed for the offline ATSP. Secondly, lower bounds for the best possible online solution have to be compared. This involves another difficult modeling problem. First steps in this direction have been made in [3].

Acknowledgment: Michael Jünger and Gerd Reinelt implemented most of the heuristics, the branch&bound code was implemented by Matteo Fischetti and Paolo Toth [11]. We would like to thank them for making their implementations available to the authors. Furthermore, we would like to thank the engineers of Siemens Nixdorf, especially Christof Gerstenäcker and Herbert Schorer, for their support during the project.

References

[1] Abdel-Aziz Abdel-Hamid A (1994) Combinatorial optimization problems arising in the design and management of an automatic storage system. PhD thesis, Technical University Berlin

[2] AMSEL (1997) A modelling and simulation environment library. Developed at the Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) See http://www.zib.de/ascheuer/AMSEL

[3] Ascheuer N (1995) Hamiltonian path problems in the on-line optimization of flexible manufacturing systems. PhD thesis[1], Technical Univiversity Berlin

[4] Ascheuer N, Grötschel M, Kamin N, Rambau J (1998) Combinatorial online optimization in practice. Preprint 98–07[1], Konrad-Zuse-Zentrum Berlin

[5] Ascheuer N, Grötschel M, Krumke SO, Rambau J (1998) Combinatorial online optimization. Preprint 98–24[1], Konrad-Zuse-Zentrum Berlin

[6] Ausiello G, Feuerstein E, Leonardi S, Stougie L, Talamo M (1994) Serving requests with online routing. In: Proceedings of the 4th Scandinavian Workshop on Algorithm Theory

[7] Ausiello G, Feuerstein E, Leonardi S, Stougie L, Talamo M (1995) Competitive algorithms for the on-line traveling salesman. In: Workshop on Algorithms and Data Structures

[8] Borodin A, El-Yaniv R (1998) Online computation and competitive analysis. Cambridge University Press

[9] Burkard RE, Fruhwirth B, Rote G (1995) Vehicle routing in an automated warehouse: Analysis and optimization. Annals of Operations Research 57

[10] Fiat A, Woeginger GJ (1998) Online algorithms – The state of the art. Number 1442 in Lecture Note in Computer Science Springer

[11] Fischetti M, Toth P (1992) An additive bounding procedure for the asymmetric TSP. Mathematical Programming 53:173–197

[12] Fischetti M, Toth P (1997) A polyhedral approach to the Asymmetric Traveling Salesman Problem. Management Science 43:1520–1536

[13] Goetschalckx M, Ratliff HD (1988) Sequencing picking operations in a man–aboard order picking system. MFL 4:255–263

[14] Graves SC, Rinnooy Kan AHG, Zipkin PH (1989) (eds.) (1989) Logistics of production and inventory. volume 4 of Handbooks in Operations Research and Management Science Elsevier Science B.V., Amsterdam

[15] Grötschel M (1992) Discrete mathematics in manufacturing. In: O'Malley RE (ed.) ICIAM91: Proceedings of the Second International Conference on Industrial and Applied Mathematics, SIAM, pp. 119–145

[1] Avail. at URL http://www.zib.de/ZIBbib/Publications/

[16] Han MH, McGinnis LF, Shieh JS, White JA (1987) On sequencing retrievals in an automated storage/retrieval system. IIE Transactions: 56–66
[17] Jünger M Reinelt G, Rinaldi G (1995) The traveling salesman problem. In: Ball MO, Magnanti TL, Monma CL, Nemhauser GL (eds.) Network Models, volume 7 of Handbooks in Operations Research and Management Science, North Holland, pp. 225–330
[18] Lawler EL, Karel Lenstra JK, Rinnooy Kan AHG, Shmoys DB (eds.) (1985) The Traveling Salesman Problem. John Wiley & Sons Ltd, Chichester
[19] Meller RD, Mungwattana A (1995) Multi-shuttle automated storage/retrieval systems. Technical Report 95–06, Auburn University, AL, USA, Dept. of Industrial Engineering
[20] Ratliff HD, Rosenthal AS (1983) Orderpicking in a rectangular warehouse: A solvable case of the traveling salesman problem. OR 31:507–521
[21] Van Dal R (1992) Special cases of the traveling salesman problem. PhD thesis, University of Groningen, The Netherlands
[22] Van den Berg J (1996) Planning and control of warehousing systems. PhD thesis, University of Twente, The Netherlands