Journal of Heuristics, 8: 585–599, 2002

A Genetic Algorithm with a Compact Solution Encoding for the Container Ship Stowage Problem

OPHER DUBROVSKY, GREGORY LEVITIN AND MICHAL PENN
Faculty of Industrial Engineering and Management, Technion, Haifa, 32000, Israel

Abstract

The purpose of this study is to develop an efficient heuristic for solving the stowage problem. Containers on board a container ship are stacked one on top of the other in columns, and can only be unloaded from the top of the column. A key objective of stowage planning is to minimize the number of container movements. A genetic algorithm technique is used for solving the problem. A compact and efficient encoding of solutions is developed, which reduces significantly the search space. The efficiency of the suggested encoding is demonstrated through an extensive set of simulation runs and its flexibility is demonstrated by successful incorporation of ship stability constraints.

Key Words: ship stowage, genetic algorithm, compact solution encoding

1. Introduction

Container ships have revolutionized shipping and transformed the shipping business from work intensive, to machine intensive. Until recently, container ship loading was still done manually, by skilled personnel, using mostly experience and rules of thumb. A few software packages that support container ship planning have been introduced e.g. Saginaw II and Perakis (1989), Sha (1985), Shields (1984), and Dunbleton (1980). These software packages help to store details about each container and calculate loads and moments on the ship. However, attempts to fully automate the planning by using heuristics, are unsatisfactory.

Today, a modern container ship may carry 5000 containers or more and visit 10–25 ports, unloading and loading containers as it goes. The containers are stacked one on top of the other in columns, and can only be unloaded from the top of the column using large port cranes. The charge to the ship for moving containers could be as high as $200 per container move. Thus, a key objective of stowage planning is to minimize the number of container movements. The subject of our study is to devise a plan that reduces as much as possible the number of unnecessary movements. However, it must be emphasized that the stowage plan must also contemplate other aspects, such as, stability, stress, ballast, longitudinal crane moment, and others.

A related problem that can be modeled similarly is the problem of dispatching trams in a storage yard. In this problem, the incoming order of trams and the order in which trams of different types are supposed to leave during the day are given. The aim is to minimize the total number of unnecessary movements of trams, see Blasum et al. (1996) and Winter and Zimmerman (1997).

Binary linear programming formulations of the container stowage problem appear in Avriel and Penn (1993) and Botter and Brinati (1992). Finding an optimal solution using these binary models is quite limited, because of the large number of binary variables and constraints needed for the formulation. For the very special case of one uncapacitated column, an optimal algorithm was developed, see Aslidis (1990). Also, it is shown in Avriel, Penn, and Shpirer (2000) that the minimum overstowage (shift) problem is NP-complete. Therefore heuristic methods producing "good" solutions have to be explored. In Avriel and Penn (1993) a heuristic, called the Whole Column Heuristic Procedure was described. This procedure was unsatisfactory since it involves some binary linear programming. In Avriel et al. (1998) a different heuristic, called the Suspensory Heuristic Procedure, was developed and tested on a large number of simulation runs. The computation times and the quality of the results were very satisfactory. However, this heuristic solves only a simplified version of the problem. Its major drawback is its inflexibility in dealing with problems where some of the simplifying assumptions are removed. For this reason a simulated annealing algorithm and a branch and bound algorithm were used to solve the shifting problem (Flor, 1998; Horn, 2000). Their major advantage was their flexibility in handling a variety of constraints added to the basic problem. Unfortunately, only small sized problems could be solved by these heuristics. In addition, the simulated annealing algorithm yielded poor results.

This motivated us to try a different approach, namely, a genetic algorithm (GA), to solve the problem. It was felt that the genetic algorithm approach could handle well the planning of container ship loading due to its parallel and non linear nature of search, and that it can handle a variety of constraints to be added to the simplified version of the problem. Special care in the design of the algorithm was given, to provide a compact and an efficient encoding, to enable parallel implementation and to allow flexibility in adding constraints. We demonstrate this flexibility, by showing how to handle the stability constraint. In order to study the quality of our algorithm we have adapted it to solve the simplified problem, so it could be evaluated against the heuristics previously developed.

Recently, a GA implementation for the stowage problem, different from ours, was carried out by Todd and Sen (1997), and they developed a multi-criteria genetic algorithm for solving the stowage problem. In Section 4, we compare the two implementations and indicate the advantages of our approach.

The extensive testing of the GA algorithm shows its ability to obtain good results even when subject to constraints. This is a significant improvement over the prior developed heuristics, which reached good solutions but could *not* deal with crucial constraints.

This paper is organized in the following way. In Section 2 we give a definition of the container ship stowage problem. In Section 3 a brief description of GA is given. Section 4 is devoted to the description of the compact solution encoding technique and adaptation of GA to the problem. In Section 5 we analyze the results obtained.

2. The container ship stowage problem

Consider a container ship consisting of a single bay with R rows labeled $r = 1, \ldots, R$ (row 1 is the bottom row) and C columns labeled $c = 1, \ldots, C$ (column 1 is the first column on the left) for stowage of containers. Also assume that all the containers are of the same

standard size. Note that in practice the containers can be of different sizes and of different kinds, such as refrigerated containers or containers that contain hazardous material. Also, in practice, the ship has cargo lids that form the deck, so that the containers can be stored above or under the deck.

The ship starts its service route at port 1 with its bay empty of containers and terminates at port N after all its containers are unloaded. Let $T = [T_{ij}]$ be the $(N-1) \cdot (N-1)$ **transportation matrix**, where T_{ij} is the number of containers originating at port i with destination port j. T is an upper triangular matrix and we assume it is known before the ship starts its service route. We further assume that T is feasible, that is, all the quantities to be shipped can be stowed in the bay along the route. This is not a real restriction since feasibility can be easily checked. We say that a container is a **j-container** if its destination is port j. A k-container, stored on top of a j-container where $k > j$, is called a **blocking container**. This is since the k-container blocks the j-container to be unloaded in port j.

Shifting is defined as the temporary removal from and placement back (unloading and reloading) of containers onto a stack of containers. The need for shifting arises, for example, in a vertical stack of containers if there is a k-container that blocks a j-container that is placed below it. Such shifting can be done at port j, and then it is called a **necessary shift**, or at some earlier port before j, and then it is said to be a **voluntary shift**. Voluntary shifts are used for preventing costlier shifts in future ports. The **container ship stowage** problem is to find an arrangement that would minimize the total number of shifts.

3. Genetic algorithms

3.1. General description

The GA is a search technique originally inspired by biological genetics. For a brief introduction to Genetic Algorithms see Austin (1990), and for a more detailed one consider Goldberg's comprehensive book (Goldberg, 1989). Recent developments in GA theory and practice can be found in Bäck (1996). GAs were applied for solving various difficult combinatorial optimization problems, for example, the bin-packing problem (Cheng-Yan Kao and Feng-Tse Lin, 1992; Khuri, Schutz, and Heitkotter, 1995; Reeves, 1994) and the Traveling Salesman Problem (Johnson and McGeoch, 1997).

Unlike various constructive optimization algorithms that use sophisticated methods to obtain a good single solution, the GA deals with a set of solutions (population) and applies to each solution simple procedures of crossover, mutation and quality evaluation. Solutions in the GA are encoded using finite length strings. Here, we basically use the GENITOR version of GA (Whitley, 1989).

First, the initial population of bitwise 200 randomly constructed solutions (strings) is generated. Within this population new solutions are obtained during the genetic cycle using a crossover operator. Crossover produces a new solution (offspring) from a randomly selected pair of parent solutions providing inheritance of some basic properties of the parents in the offspring. Each new solution is decoded and its objective function (fitness) values are estimated. These values, which are a measure of quality, are used to compare different solutions.

The comparison is accomplished by a selection procedure that decides which solution is better: the newly obtained one or the worst solution in the population. The better solution joins the population and the worse one is discarded. If the population already contains an equivalent solution, then the solution is discarded. As a result, the population size remains constant, and the average fitness value of the population increases monotonically or remains unchanged over the cycles of evolution.

If the population fitness value remains unchanged for more than N_c cycles, a "cataclysm" procedure wipes out most of the population, leaving only the best solutions, and creating random strings to replace the destroyed ones. This enriches the population with new genetic material and helps to continue the genetic evolution in the new genetic cycle.

The GA terminates after N_b "cataclysms" occurred without improvement of the best-in-population solution. The final population contains the best solution achieved. It also contains different near optimal solutions which may be of interest in the decision making process.

3.2. Parallel implementation

The genetic algorithm approach is ideal for parallel implementation. At each iteration, most of the computing work is done on combining the genes of the two parents into the two offspring and evaluating the offspring's quality. Creating and evaluating new pairs of offspring could be done in separate parallel processes, as it is done in nature. Thus, by using multiple processors, one for each offspring, it is possible to speed up the process considerably. In fact, by implementing the algorithm on a parallel computer using 3 processors, a speedup of 270% was achieved over runs using one processor only.

4. Implementation of the GA to the stowage problem

4.1. Encoding of the solution

A crucial issue in implementing genetic algorithms is the choice of the coding of the solutions as chromosomes so that the algorithm will converge to good solutions. Some encodings may cause the algorithm to create infeasible solutions, or may change and enlarge the search space making it difficult to converge.

A straightforward encoding approach was used in Todd and Sen (1997), where a multi-criteria genetic algorithm for the container ship stowage problem was developed. In this approach the solution representation vector has different sections corresponding to each port. Each section contains integer vectors of size P, where $P = R * C$ is equal to the total number of container slots in the ship. Each element in such a vector indicates the destination port of the container that occupies the corresponding slot at the given port. We call such an encoding a **complete encoding**.

The complete encoding scheme, while feasible, has some major disadvantages. First, it creates very long vectors. For example, to encode a ship carrying 1500 containers and visiting 15 ports, one needs a vector of length 22500. Such long vectors cause the search space to be very large, and slow the algorithm convergence. The complete representation is also

storage space consuming, which is especially undesirable when parallel GA implementation is considered.

Also, the complete encoding is time consuming when evaluating the solution quality. This is since the evaluation procedure requires the comparison of the containers' layout at each port, and the calculations of all the loading/unloading operations that provide these changes.

Further, the complete encoding does not take into account the consistency of the containers' allocation while the ship proceeds from port to port. Indeed, in many instances, there is a large number of containers that do not change their positions during the ship route. The complete encoding approach can not prevent drastic changes in the container allocation, and thus leaves the entire burden of allocation consistency preservation to the genetic search.

In addition, the complete encoding can not guarantee the feasibility of the solutions obtained by the crossover operator. Indeed, after applying the standard crossover operator which copies fragments of two feasible parent strings into offspring string, the omissions or duplications of some containers may occur. To provide the feasibility of the newly obtained solution, a repair procedure should be used (Todd and Sen, 1997). However, this procedure destroys the information inheritance mechanism of the GA by changing the fragments inherited from the parents by the offspring string.

The compact encoding technique, suggested in this paper, is aimed to overcome these disadvantages by:

1. Obtaining a compact solution representation that will significantly decrease the search space and storage resource consumption, and will allow the convergence to good solutions within a reasonable time.
2. Simplifying the solution quality evaluation procedure.
3. Preserving the layout consistency along the ship route.
4. Preserving solution's feasibility after the use of the crossover operator.

The key idea behind the compact encoding method is that rather than holding the complete layout, only the changes in the layout that result from loading and unloading of containers along the route are to be held. Since the ship layout has a relatively small number of changes at each port, the solution encoding vectors can be much smaller. Encoding only the changes in the container layout significantly reduces the computation time of the solution's quality evaluation function. In addition, a very simple procedure can be used to insure the feasibility of the solutions obtained by the crossover operator.

In the proposed **compact encoding** method, the integer string representing the entire solution is divided into N sections, one for each port. For any port k, each section contains four parts **n**, **f**, **q** and **g** as follows: Part **n** contains a list of columns where the containers originated at port k are to be loaded to; Part **f** (respectively, **g**) contains a list of columns to which the containers that were unloaded due to necessary shifts (respectively, voluntary shifts) are to be loaded; Part **q** contains a list of columns from which the containers should be unloaded, due to voluntary shifts.

Note that any voluntary shift is determined by parts **q** and **g** of some section in the following way: The top container located at the column indicated by the j-th position of part **q**, should be loaded to the column indicated by the j-th position of part **g**.

To perform the ship unloading and loading operations at a given port, two auxiliary vectors W_p and w_{pr} are used. The **port waiting list $\mathbf{W_p}$** contains the destinations of the containers to be loaded at port p. Initially, W_p is equal to the list of all the destinations of the containers originated at port p. This list is obtained from the transportation matrix (note that the cardinality of W_p equals the length of the **n** part of the solution encoding string section corresponding to port p). The **column waiting list** w_{pr} contains the destinations of the containers aimed to be loaded to the c-th column while the ship calls at port p. This list is constructed according to the solution representation string.

When the ship calls at port p, first, all the p-containers, i.e. destined to the current port, are to be unloaded. Observe that there are cases where necessary shifts are unavoidable. If so, we add all the blocking h-containers, with $h > p$, to the port waiting list. Thus, $W_p \leftarrow h$. Then, after unloading the p-containers and those blocking them, some voluntary shifts may occur. In such a case, the upper containers located in the columns that are specified by the **q** part of the section, are moved to the waiting lists of the columns specified by part **g**. That is,

$$w_{pg(j)} \leftarrow U(q(j)), 1 \leq j \leq J$$

where $U(x)$ is the upper container located in column x, J is the total number of voluntary shifts, $q(j)$ and $g(j)$ are the columns indicated by the **q** and **g** parts of the section.

Thereafter, the containers from the port waiting list are distributed among the column waiting lists according to the columns indicated by the corresponding positions of the **n** and **f** parts of the section:

$$w_{ps(m)} \leftarrow W(p), 1 \leq m \leq M,$$

where **s** is the concatenation of the two lists, **n** and **f**, and M is the total number of containers in the port waiting list.

After all the containers are distributed among the column waiting lists, they are loaded to the corresponding columns in order. That is, for any k in w_{pc}, all the j-containers that belong to w_{pc} with $j > k$, are stored below the k-container. Now, if a column to which the containers should be loaded is completely filled up, the extra containers are moved to the closest available column with a higher index (here, we consider column $C + 1$ as column 1). An example of a solution decoding, based on the partial solution representation, is presented in Appendix A.

It should be noted that the total number of necessary shifts is not known before running the solution decoding procedure. Therefore, the length of part **f**, of any section, should be chosen such that the greatest possible number of necessary shifts could be handled. This implies that part **f** usually contains redundant elements, that is, elements that are not used by the decoding procedure. Also, note that the maximum number of voluntary shifts allowed is specified by the length of the **q** and **g** parts of the section. However some shifts may become impossible if some of the columns listed in the **q** part are empty at the corresponding port after unloading is completed. (Such elements of **q** should be skipped).

4.2. Choice of the GA parameters

A sensitivity analysis with respect to each parameter was carried out. To set the Genetic Algorithm parameters, a large number of simulation runs were done for different values of each of the parameters in consideration. Since these simulation runs are time consuming we based our sensitivity analysis on the following two problems. The first problem was a relatively easy one with $R = 10$, $C = 30$ and $N = 10$. We have used this problem to get some insight on the influence of the various parameters on the performance of the algorithm. The second problem was a more difficult one with $R = 10$, $C = 100$, $N = 11$ and a transportation matrix which is based on real data. The amount of cargo carried was as high as 98% of the ship's carrying capacity during the route. About 300 runs were performed on the second problem.

Three parameters significantly influenced the algorithm's convergence. The optimal values of these parameters had to be empirically found before we could start rigorous testing of the algorithm. These parameters were the population size, the crossover rate and the cataclysm rate. The method employed to fine-tune the parameters was as follows:

First a few runs with large changes in these parameters were made. After finding the range that gave the best results in terms of convergence rate and final value, more extensive and detailed runs were made. These runs were made by fixing two of the parameters to the previous found values and incrementing the third over a large range of values. This process was run three times totaling a few hundred runs.

4.2.1. Population size. The results obtained show that the best population size was 200. This is similar to the sizes mentioned in the literature dealing with the GENITOR version of GA (Wainwright and Blanton, 1993).

4.2.2. Crossover rate. As mentioned before we have used a bitwise crossover. In order to fix the value of the probability p, of choosing the bit from the first parent, 44 runs were performed with the probability value increased by 0.1 each few runs. These results are demonstrated in figure 1. The horizontal axis of the chart indicates the various probability

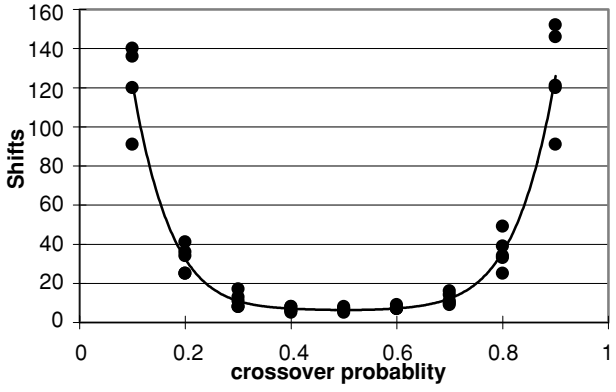

Figure 1. Number of shifts vs. crossover probabilities.

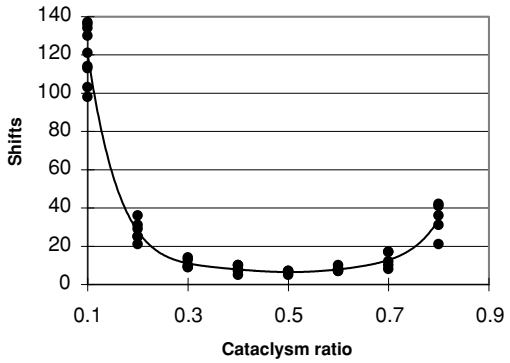

Figure 2. % of destruction in the population vs. number of shifts.

values, while the vertical axis indicates the number of shifts in the final solution. The chart displays the results of all runs made for each value of p. The results obtained indicate that $p = 0.4$ gives the best results with respect to the number of shifts and the running times. Also it turns out that the standard deviation of the number of shifts and that of the running times were the smallest for the best fitted probabilities. This gives an indication of the stability of our algorithm with regard to the crossover rate. Figure 1 indicates as well the robustness of the crossover rate. It shows that if p varies from $p = 0.3$ to $p = 0.7$, then there is only little variation in the final solution.

4.2.3. Cataclysm rate. In order to fix the cataclysm rate (the fraction of the population to be replaced when the cataclysm occurs), 90 different simulations were carried out. The runs were carried out with the cataclysm rate increased by 0.1 each few runs. The results show that 0.5 was the best rate. It was also shown that the standard deviation of the number of shifts was the smallest for the best fitted rate, thus indicating the stability and robustness of the algorithm with regard to the cataclysm rate. These results are demonstrated in figure 2.

As additional numerical tests show, the GA with the chosen values of the parameters outperforms GAs with different values of the parameters for a wide range of stowage problems. This means that the optimal GA tuning for the container ship stowage problem is invariant to the parameters of transportation matrix and to number of rows and columns in the ship.

5. Numerical implementation and simulation results

5.1. Simulation results

An analysis of the GA performance and the effect of incorporating local optimization in the GA were carried out. In order to evaluate the quality of the solutions, the algorithm (without the additional local optimization rules) was compared to the heuristics developed in Avriel et al. (1998). In Avriel et al. (1998) three types of triangulated transportation

Table 1. Comparison of the GA to the suspensory, primitive and multiplication heuristics.

Ship size	Same or better than best heuristic		Within a difference of 5 shifts (%)	Average difference	Std
	No. of runs	(%)			
10×50	50	38	66	3.4	7
10×100	41	41	68	5.8	8.9

matrices, according to the average period of time a container has to stay on board, were considered. These types of matrices were denoted as short distance matrices, long distance matrices and mixed matrices. All these transportation matrices were feasible. Here, runs were carried out on mixed matrices and we used the results obtained for a comparison with the Suspensory Heuristic. We create the mixed matrices in the following way.

5.1.1. Mixed matrices. We choose randomly the order in which the values of the elements of the transportation matrix are determined. At each step, we choose i from the uniform distribution on $(1, 2, \ldots, n-1)$. Next, we choose j from the uniform distribution on $(i+1, \ldots, n)$. Assume the pair (i, j) was chosen. Then, we check the elements of the matrix in an increasing lexicographic order, starting with (i, j), for an element that has not been chosen previously. If we did not find such an element, we return to (i, j) and check the elements of the matrix in a decreasing lexicographic order. Fixing (i, j) we choose T_{ij} randomly such that

$$\sum_{i=1}^{k} \sum_{j=k+1}^{n} T_{ij} \leq R \cdot C$$

for all $k = 1, \ldots, n-1$. Also, we required that $T_{ij} \leq 0.2\ R \cdot C$. The last constraint was added so that the number of elements T_{ij} having zero value will not be too large. Note that the first set of constraints causes the magnitude of the near diagonal elements to be somewhat greater than that of the other elements.

A large number of simulation runs were done for each different value of the parameters in consideration. We consider two different ship sizes, one with 10 rows and 50 columns, and the other with 10 rows and 100 columns. The number of ports varied between $N = 5$ and $N = 14$. In total, 413 runs were made. Running times were from a few minutes on small problems to about one hour on the larger ones. Runs were carried out on a Silicon Power Challenge XL using 3 processors in parallel.

The runs performed for problems with 5–14 ports show that the quality of the results obtained by the GA is similar to that obtained by the heuristics. This is demonstrated in Table 1 which shows that the difference between the results obtained by the GA and the ones obtained by the heuristics is, in most cases, only a few shifts.

5.2. Local optimization

An interesting direction for improving the performance of the GA is to include in the algorithm some local optimization rules. Although we have tackled this issue, as indicated

below, it is unclear to us at this point whether local optimization will make any significant improvement. The motivation for introducing these rules was to try to reduce the search space and thus to improve the GA convergence. The following two local improvements were examined.

1. *Sort*: Sorting the containers before loading them back to the ship, so that the containers will be loaded in order. This method may lessen unnecessary shifts when unloading the ship at the following ports.
2. *Group*: During a voluntary shift, move a group of j-containers, for some j, that are stacked continuously in a column. This procedure makes voluntary moves in batches rather then with single containers. The logic behind this is that if a j-container is moved in order to save shifts later, and if there is another j-container underneath it, the second j-container should be moved as well.

Surprisingly, incorporating these local optimization rules into the GA did not bring any significant improvement. We have pondered over this much, and finally came to the conclusion that these local improvements do not actually reduce the search space. Instead, they create multiple solution vectors that have the same fitness value. Thus the search space is not reduced at all, so finding the optimal solution does not become any easier than in the regular method.

6. Additional constraints

Recall that up to now we have considered the simplified version of the problem. However, as it was indicated previously, the stowage plan must also satisfy constraints. As seen previously, alternative heuristics could not handle constraints. To demonstrate the flexibility of the approach suggested we incorporate in the problem a ship stability constraint. Note that putting too much weight on one side of the ship may create a dangerous tilt. The tilt is calculated in relation to the axis of symmetry going through the ship from bow to stern. Containers on one side of the ship create a positive tilt, which is the product of their weight and distance from the axis of symmetry. Containers on the other side of the ship create a negative tilt. The total tilt of the ship is the sum of all those tilts. The ship's tilt can be balanced, up to a certain threshold level, by filling ballast tanks at the cost of using more fuel on the voyage. Beyond that threshold level, the ship could capsize. It is therefore important to keep the tilt below the dangerous threshold level during all legs of the voyage.

To incorporate the stability constraint, a penalty function was introduced into the algorithm. The penalty function at each port is: penalty*max[0,(abs(tilt)-threshold)], where penalty is a constant, abs(tilt) is the absolute tilt of the ship for the current configuration and threshold is a constant representing the threshold level of the ship. The constant penalty actually represents the exchange rate that we are willing to pay for each unit of tilt that is beyond the threshold level. Since the final solutions were with a few shifts only, we set the constant "penalty" to a value of 20. This favors arrangements with tilts that do not exceed the threshold level. We tried a few other penalty values in the range of 20 and discovered that this value was not sensitive to small changes.

Table 2. Runs without penalty function.

Run number	Iteration	Final shifts	Average of tilt over all ports	Number of ports exceeding threshold	Average of tilt in ports exceeding threshold	Std of tilt in ports exceeding threshold
1	59760	0	24.3	5	17.2	18.2
2	47250	0	16.3	2	18.0	14.1
3	63810	0	14.5	3	12.0	11.5
4	47610	0	17.9	3	12.7	7.6
5	35370	0	8.8	1	12.0	–
6	67140	0	13.7	3	13.7	8.5
7	31050	0	13.0	1	7.0	–
8	58950	0	19.4	4	11.8	7.6
9	32670	0	25.9	4	30.3	21.4
10	60570	0	15.5	3	10.7	3.5
Median	53280	0	15.9	3	12.3	10.0
Average	50418	0	16.9	2.9	14.5	11.6

Twenty simulations were run on a given transportation matrix with 11 ports and a ship size of 10 rows and 100 columns. The ship's width was set to 5 columns abreast and all the containers had a weight of 1. The threshold level was set at 20.

The comparison of the results presented in Tables 2 and 3 allow estimating the influence of tilt constraints on the final solution.

Table 3 summarizes results from 21 runs with a penalty function while Table 2 summarizes runs using the same transportation matrix without the penalty function. All 21 final solutions of the constrained problem reached an arrangement with all tilts conforming to the threshold level. This can be seen by the final penalty value which is zero for all runs. These arrangements came at a cost of a few extra shifts for each route. This is demonstrated when comparing to the 10 runs shown in Table 2. In this case all the final solutions reached arrangements with 0 shifts, but at a cost of violating the threshold tilt at many of the ports.

Tables 2 and 3 also show the difference in the number of iterations needed to reach a final solution. The average number of iterations in the runs with no penalty was 50,418 iterations, while in the runs with the penalty function the average was 259,084, which is about 5 times larger. The running time on a 300 MHz Pentium II for a single problem with tilt constraints was about 30 minutes.

On runs not implementing the penalty function, the overrun over the threshold averaged to 2.9 ports out of the 10 ports arranged. The overruns in these ports were by an average tilt of 14.5 beyond the threshold level (beyond 20).

7. Discussion

As was indicated in the introduction, a few heuristics were developed for solving the stowage problem. The best of which was the Suspensory Heuristic Procedure, which provided very

Table 3. Runs with penalty function.

Run number	Stop iteration	Final shifts	Final penalty value	Final value of shifts + penalty
1	261810	2	0	2
2	268290	2	0	2
3	267390	2	0	2
4	244710	5	0	5
5	254520	3	0	3
6	260100	2	0	2
7	262620	3	0	3
8	267480	4	0	4
9	264510	21	0	21
10	264600	5	0	5
11	252720	6	0	6
12	264870	5	0	5
13	258300	8	0	8
14	265680	5	0	5
15	256860	5	0	5
16	249570	16	0	16
17	255150	1	0	1
18	254880	5	0	5
19	251190	2	0	2
20	258750	6	0	6
21	256770	6	0	6
Median	258750	5	0	5
Average	259084	5.4	0	5.4

satisfactory results for the simplified non-constrained problem. However, its drawback was its inflexibility in dealing with constraints. An integer programming based algorithm as well as simulated annealing and a branch and bound algorithm were developed to enable handling a variety of constraints. Unfortunately, these heuristics could solve only small sized problems. This motivated us to use a GA for solving the problem in the hope that it would be able to provide solutions to large scale constrained problems, which previous methods could not manage. As choosing the coding of the solutions is one of the crucial issues in implementing a GA, we put a significant effort into this issue, and came out with the compact encoding, which we believe is very promising. This compact encoding reduces the search space and allows the GA to converge to good solutions within a reasonable time.

As can be seen from the numerical implementation for large sized problems, the results obtained by the GA for the non-constrained simplified problem were obtained in a reasonable time with values similar to those obtained by the previous best method—the

Suspensory Heuristic. Moreover, the GA approach for solving the stowage problem has the possibility of incorporating constraints beyond the simplified problem. We demonstrate this by successfully incorporating the stability constraints. The algorithm provided feasible solutions that were all within the constraint limits. The final results were compared to the ones obtained by the best known heuristics run on identical but non-constrained problems. These heuristics provided good valued solutions, yet they were infeasible due to deviations from the constraints. On the other hand, the GA provided results that were both feasible and close enough to those obtained by the heuristics on the same but non-constrained problems.

As was shown, this algorithm is much more suitable for handling real life container ship stowage problems and may be further developed to include all ship constraints. In further research, one should consider incorporating additional constraints beyond stability.

Appendix A

Solution decoding example

Consider a ship that enters port 2 loaded in the following way:

2		*3*
2	*3*	*2*
5	*5*	*2*
3	**2**	**1**

column

Assume that the port waiting list at port 2 consists of a single 4-container and two 5-containers. Thus, $W_2 = \{4, 5, 5\}$.

A possible solution encoding section corresponding to port 2 is

...	Port 2												...
	n			**f**					**q**		**g**		
	3	2	1	1	...				3	2	1	3	

First, all the 2-containers should be unloaded. Note that a 3-container, which was located above the 2-containers in the first column, is unloaded (necessary shift) and is included into W_2. After unloading is completed, the ship layout is as follows:

	3	
5	*5*	
3	**2**	**1**

column

At this point, all column waiting lists are empty, and $W_2 = \{4, 5, 5, 3\}$. Following the solution encoding section, two additional voluntary shifts are to be performed. First the upper 5-container has to move from column 3 to column 1, and then the upper 3-container has to move from column 2 to column 3. The containers that were unloaded due to voluntary shifts join the corresponding column waiting lists. Therefore, after the completion of the voluntary shifts, the container allocation is

	5	
3	**2**	**1**

column

and the column waiting lists are $w_{21} = \{5\}$, $w_{22} = \emptyset$, $w_{23} = \{3\}$.

We now turn to distribute the containers contained in W_2 to the column waiting lists according to the **n** and **f** parts of the solution encoding section. Note that the i-th column number in the concatenation of parts **n** and **f** is matched to the i-th container in W_2. Thus, the first container of W_2 should be added to w_{23}, the second to w_{22} etc. Hence, after the containers' distribution, W_2 is empty and the column waiting lists are $w_{21} = \{5, 3, 5\}$, $w_{22} = \{5\}$, $w_{23} = \{3, 4\}$. Now, for each column c, the containers in w_{2c} are loaded in order. That is, for any j-container in w_{2c}, all the k-containers that belong to w_{2c} with $k > j$, are stored below the j-container. The ship layout upon leaving port 2 is as follows:

		3
3	*5*	*5*
4	*5*	*5*
3	**2**	**1**

column

Acknowledgments

Partial support was received from the fund for the promotion of research at the Technion. Part of this work was done as part of Opher Dubrovsky M.Sc. thesis, done under the supervision of Gregory Levitin and Michal Penn, in the Faculty of Industrial Engineering and Management, Technion, Haifa, Israel. The authors are grateful to the referee for valuable comments.

References

Aslidis, A. (1990). "Minimizing of Overstowage in Container Ship Operations." *Operational Research* 90, 457–471.

Austin, S. (1990). "An Introduction to Genetic Algorithms." *AI Expert* 5, 49–53.

Avriel, M. and M. Penn. (1993). "Exact and Approximate Solutions of the Container Ship Stowage Problem." *Computers and Industrial Engineering* 25, 271–274.

Avriel, M., M. Penn, and N. Shpirer. (2000). "Container Ship Stowage Problem: Complexity and Connection to the Coloring of Circle Graphs." *Discrete Applied Mathematics* 103, 271–279.

Avriel, M., M. Penn, N. Shpirer, and S. Witteboon. (1998). "Stowage Planning for Container Ships to Reduce the Number of Shifts." *Annals of Operations Research* 76, 55–71.

Bäck, T. (1996). "Evolutionary Algorithms in Theory and Practice." Evolution Strategies. Evolutionary Programming. Genetic Algorithms. London: Oxford University Press.

Blasum, U., M. Bussieck, W. Hochstattler, C. Moll, H.H. Scheel, and T. Winter. (1996). "Scheduling Trams in the Morning is Hard." Working Paper, Department of Mathematical Optimization, Technical University of Braunschweig, Braunschweig, Germany.

Botter, R.C. and M.A. Brinati. (1992). "Stowage Container Planning: A Model for Getting an Optimal Solution." *IFIP Transactions B (Applications in Techn.)* 5, 217–229.

Cheng-Yan Kao and Feng-Tse Lin. (1992). "A Stochastic Approach for the One-Dimensional Bin-Packing Problems." In *Proc. of the 1992 IEEE Int. Conf. on Systems, Man, and Cybernetics*, Vol. 2, pp. 1545–1551.

Dunbleton, J.J. (1980). "Expert System Applications to Ocean Shipping—A Status Report." *Marine Technology* 27, 265–284.

Flor, M. (1998). "Heuristic Algorithms for Solving the Container Ship Stowage Problem." M.Sc. Thesis, Faculty of Industrial Engineering and Management, Technion, Haifa, Israel (in Hebrew).

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison Wesley.

Horn, G.P. (2000). "A Branch and Bound Algorithm for the Container Ship Stowage Problem." M.Sc. Thesis, Faculty of Industrial Engineering and Management, Technion, Haifa, Israel (in Hebrew).

Johnson, D.S. and L.A. McGeoch. (1997). "The Traveling Salesman Problem: A Case Study." In E. Aarts and J.K. Lenstra (eds.), *Local Search in Combinatorial Optimization*.

Khuri, S., M. Schutz, and J. Heitkotter. (1995). "Evolutionary Heuristics for the Bin-Packing Problem." In *Proc. of the ICANNGA'95*, Ales, France, pp. 285–288.

Reeves, C.R. (1994). "A Genetic Approach to Bin-Packing." In *Proc. of the Second Finnish Workshop on Genetic Algorithms and their Applications*, Vaasa, Finland, pp. 35–49.

Saginaw II, D.J. and A.N. Perakis. (1989). "A Decision Support System for Container Ship Stowage Planning." *Marine Technology* 26, 47–61.

Sha, O.P. (1985). "Computer Aided on Board Container Management." *Computer Applications in the Automation of Shipyard Operation and Ship Design* V, 177–187.

Shields, J.J. (1984). "Container Ship Stowage: A Computer-Aided Preplanning System." *Marine Technology* 21, 370–383.

Todd, D.S. and P. Sen. (1997). "A Multiple Criteria Genetic Algorithm for Container Ship Loading." In *Proceedings of the Seventh International Conference on Genetic Algorithms*.

Wainwright, R. and J. Blanton. (1993). "Multiple Vehicle Routing with Time and Capacity Constraints Using Genetic Algorithm." In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 452–459.

Whitley, D. (1989). "The GENITOR Algorithm and Selective Pressure: Why Rank-Based Allocation of Reproductive Trials is Best." In D. Schaffer (ed.), *Proc. of the 3th International Conference on Genetic Algorithms*. Morgan Kauffmann, pp. 116–121.

Winter, T. and U. Zimmerman. (1997). "Minimizing Shunting Costs in Storage Yards." Internal Report, Department of Mathematical Optimization, Technical University of Braunschweig, Braunschweig, Germany.