An adaptive constructive solution generator for ship scheduling DSS

Geir Brønmo
SAS, Copenhagen, Geir.Bronmo@iot.ntnu.no
Arne Løkketangen
Molde University College,Norway, arne.lokketangen@himolde.no

Abstract
It is well known that problems solved by a decision support system (DSS) is often only an approximation to the real problem, simplifying some of the features, to be able to find solutions to the problem. A decision maker, DM, might in these circumstances be equally interested in a set of good, diverse, solutions, than in only the optimal solution as produced by the model. As modern search techniques are able to produce a plethora of solutions, some distinguishing measure based on structural difference between solutions is needed in addition to the quality measure normally used. We present a self-learning restart heuristic for solving ship scheduling problems, adapted to a DSS. The heuristic is based on a set partitioning formulation of the shipping scheduling problem where columns are heuristically generated a priori. The resulting problem is solved repeatedly by a constructive heuristic with learning weights on both ships and cargoes, guiding the solution process towards diverse high quality solutions. A new diversification measure is introduced. Finally, a selection of the solutions from the construction process is subjected to local search.

1. Introduction

Proper planning of shipping operations is a profitable task. Ships are major capital investments, and it is therefore crucial that their capacity is utilized in the best possible manner. However, scheduling research has received less attention within shipping compared to air and land based transportation. Within ship scheduling research at an operational level, it is customary to divide shipping into three modes of operation; liner, tramp and industrial. Liner is characterized by shipping companies who operate with a published schedule, like a bus. Tramp is characterized by shipping companies who follow available cargoes, like a taxi. Tramp shipping companies normally have long term contracts with goods owners and maximize profit on cargoes from the spot market. Within industrial shipping the goods owners take care of the transportation themselves. Christiansen et al. (2004) give a thorough survey on ship scheduling.

TurboRouter is a decision support system, DSS, for tramp shipping companies produced by MARINTEK, SINTEF Group, in Trondheim, Norway. The system is described by Fagerholt (2002), and has a user friendly GUI facilitating the presentation of the scheduling data for the shipping company. Additionally, the system contains a planning module where schedule suggestions are provided by heuristic local search algorithms. In TurboRouter, communication and interaction with the user is a central feature, which is useful within shipping because of the difficulty of mathematically modeling and implementing all side constraints. There are thus always aspects of the problem that that is left outside of the implemented model. In such conditions it should be easy for the planner to make changes to a computer generated plan based on his own judgments. It would in these circumstances also be useful to have a set of diverse, high quality plans for the decision maker, DM, to choose from.

Ship scheduling has received surprisingly little attention in OR litterature, given the massive attention that has been given to routing and scheduling in general the latter

This paper was presented at the NIK 2007 conference. For more information see http://www.nik.no/

decades. Most of the attention has been given to the industrial shipping mode; see for instance Scott (1995), Fagerholt and Christiansen (2000) and Cho and Perakis (2001). Only a few papers have been reported within the tramp mode, see Kim and Lee (1997) and Bausch et al. (1998). Christiansen et al. (2004) state that tramp is getting more and more significant in the shipping industry. Hence, the need for decision support systems is increasing. Both Kim and Lee (1997) and Bausch et al. (1998) present decision support systems tailor made for one company. The TurboRouter system is more generic, and is in daily use by several shipping companies.

Ship scheduling problems are usually highly constrained, due to the very physical nature of the transportation mode, and finding a feasible schedule servicing all customers might itself be a difficult task. This implies that a local search heuristic that stays feasible at all times can have problems moving far in the solution space from a starting solution, and the final result is also dependent on the quality of the starting solution. For local search heuristics that allow search in infeasible space, (see Cordeau, 2001), feasibility might be very difficult to recover. TurboRouter is for these reasons designed to search only in feasible space.

We present therefore an important enhancement to the TurboRouter system. An adaptive restart heuristic with learning capabilities is developed for implementation into the system. This heuristic is designed to serve two purposes; to find diverse and high quality schedules. The different schedules serve as advanced, diverse starting points for the local search heuristic, giving high quality plans for the decision maker to choose from. This is particularly useful within a system like TurboRouter. It would be a great advantage for the planner to be able to consider several structurally different, high quality plans before making his decisions.

Ship scheduling problems are dynamic by nature. The ship scheduling decision maker normally makes plans several times during the planning horizon, for instance every time a new spot cargo becomes available. This means that the schedule that is made at the point of planning is quite likely to be changed after the starting executing the plan. It also indicates that each time the decision maker tries to solve the short term ship scheduling problem; he solves the problem with only minor changes to the data.

Our heuristic uses a set partitioning type formulation of the scheduling problem. Ship routes are generated a priori (a subset, or all, feasible single ship routes) and an iterative constructive method is used to solve the set partitioning problem repeatedly. Finally, a selection of the generated solutions is subjected to local search.

In the restart method the strategy is to guide the process towards better solutions using learning weights on the constraints. These weights capture information from prior outcomes of the constructive heuristic, giving more emphasis to constraints that have shown to be difficult to satisfy, i.e. difficult cargoes, while maintaining a focus on the quality of the plan. Additionally, we introduce weights on order_ship combinations to enhance diversity. Order-ship combinations that have been a part of an earlier solution are given a lower priority. The evaluation function for each ship route (or set partitioning column) is altered for each constructive run, adapting the constraint and variable weights. This way the search is both guided towards better solutions and different parts of the search space.

This paper is organized as follows. In section 2 the problem is described and in section 3 the heuristic is presented. Section 4 contains a computational study and discussion, while concluding remarks are given in section 5.

2. Problem Description

The tramp ship scheduling problem can be described like this: Given a heterogenous fleet of ships and a set of transportation requests or cargoes with time windows; find the most profitable assignment of the cargoes to the fleet.

The tramp ship scheduling problem is closely related to the multi-vehicle Pick-up and Delivery problem with time windows (m-PDPTW) that is thoroughly reviewed by Savelsbergh (1995). However, there are some important differences:

- Some requests can be seen as optional, i.e. not all needs to be served
- Maximise profit, not minimise cost
- Multiple time windows
- Ship/port compatibility constraints
- There are no depots, as opposed to most land based routing problems

A tramp shipping company is normally engaged in long-term contracts with goods owners for around 70% of their fleet's capacity, and maximizes profit on spot cargoes. The spot cargoes can be seen as optional, since the shipping company is not committed to lift them. The income of each cargo is decided by either a tonnage rate or a lumpsum. In the first case the actual shipped quantity for each request clearly affects the objective, while in the other case it does not. The multiple time windows arise in ports that do not operate around the clock. In addition some ports have draft restrictions, so that ships of a certain size cannot enter the port. This also means that some ships cannot enter the port when their load is above a certain limit.

Many tramp shipping companies also face constraints that are not modeled in the TurboRouter system. Penalties are often used when a ship arrives too late for a given cargo (soft time windows). There can also be conflicts between cargoes, since certain products cannot be stowed close to one another. Extra costs might be incurred when certain products are stored directly after each other in the same tank, or with less than a certain number of other cargoes in between them. The latter two constraints are for instance seen in chemical and food transportation.

3. Our heuristic solution method

We have based our heuristic solution technique on a set partitioning and packing formulation of the model. Similar approaches using a set partitioning formulation to find the optimal solution has been reported by for instance Bausch et al. (1998), Christiansen and Fagerholt (2002) and Brønmo et al. (2005). When all feasible routes are generated and given a profit, c_j, the optimal schedule can be found by solving the following set partitioning and packing problem. In this formulation A_1, A_2 and B are matrices of 0's and 1's. Each column, represented by x, represents a route, and each row in A_1 represents a mandatory contract cargo. . Each row in A_2 represents an optional spot cargo that might be serviced, and each row of B represents a ship.

$$\begin{array}{lll} \text{Max} & c^T x & (1) \\ \text{s.t.} & A_1 x = 1 & (2) \\ & A_2 x \le 1 & (3) \\ & Bx \le 1 & (4) \\ & x : x_{ij} \in \{0,1\} & (5) \end{array}$$

The objective function is defined by equation (1), while constraints (2) states that each contract cargo is lifted once and only once. Constraints (3) make sure that each optional cargo is not serviced more than once, while constraints (4) ensure that maximum one route per ship is allowed. Finally, constraints (5) impose binary requirements on the variables. For the sake of simplicity we call the presented formulation a set partitioning formulation. Constraint (2) thus represents the set partitioning constraints of the problem, while constraints (3)-(4) are thus the set packing constraints. If we introduce explicit slack variables to the set packing constraints, we have a set partitioning problem. For the sake of simplicity, we will refer to (1)-(5) as a set partitioning problem in the following. a_{1ij} = 1 or a_{2ij} = 1 means that cargo *i* is represented in route j, while b_{ij} = 1 means that route *j* is performed by ship *i*. The vector *c* contains the potential profit for each route

Our heuristic can be divided in three separate phases. In the first phase, the routes are generated. If the problem is small (few cargoes, short time horizon), or highly constrained, all routes can be generated. Otherwise, heuristic rules are implemented to reduce the number of generated routes. The second phase generates a number of solutions to the set partitioning problem using a stochastic greedy constructive method with learning capabilities. Finally, a selection of the set partitioning solutions is translated into the original problem solution formats and then improved by a local search algorithm.

3.1 Generating routes

Even given a reasonable number of cargoes, the number of possible cargo subsets for each ship, and hence the number of columns in the set partitioning constraint matrix, is potentially very large. Time and capacity constraints limit the number of feasible routes, and hence the number of columns drastically. For some shipping companies the scheduling problem will be so constrained that all routes can be generated without excess usage of computational resources. But in many cases this is not possible, and we have to concentrate on generating a promising subset of routes.

For each cargo combination a travelling salesman problem (TSP) with time windows and pick-up and delivery must be solved. The size of the cargo enumeration tree and the constrainedness of the TSP are the key factors that decide whether it is possible to generate all feasible routes or not. These factors vary with the size of the fleet and type of operation the shipping company is engaged in. If the shipping company is rather small, and only carry full loads, the number of cargoes and hence the enumeration tree, is small. On the other extreme, it would be impossible to generate all feasible schedules for instance for large shipping companies transporting chemical products (with up to 60 load compartments!).

We generate routes by enumerating all feasible cargo sets for a given ship. For a given cargo set we use a greedy insertion algorithm to find the route and schedule. The algorithm starts with a list of the cargoes sorted by time windows, chooses the earliest that is not yet placed in the schedule, and finds the best insertion of the cargo in the current schedule of the ship. Then it continues until all cargoes are assigned to the given ship. Since all feasible cargo set and ship combinations are enumerated, this method does not necessarily reduce the number of generated columns, but the computational complexity of the column generation is drastically reduced. However, the profit of each set partitioning column (i.e. ship plan) is not necessarily optimal.

Using an insertion algorithm for finding the schedule of a given cargo set and ship combination can easily be incorporated into the cargo set enumeration tree, since moving between nodes in the tree represents one of three possible actions:

- insert a new cargo into the plan
- remove one cargo and insert another cargo in the plan
- remove one cargo from the plan

When all columns are generated - although not exactly - an exact solver could be used to find the optimal solution of the set partitioning model and hence the optimal selection from the given columns. However, we believe that a DM needs more than a single high quality solution from a decision support point of view. The method we will present in the following section is thus designed to present several high quality solutions for the decision maker to choose from.

3.2 Set partitioning heuristic

A constructive heuristic is developed to solve the set partitioning formulation of the problem. The design of the heuristic is based on ideas from research work on satisfiability problems (SAT). Løkketangen and Glover (1997) developed a surrogate constraint method for SAT. It is often difficult to find a fleet schedule that services all cargoes. This is in many cases the most important part of the ship scheduling problem. In surrogate constraint analysis, the complex SAT constraints are heuristically transformed into ‘nice’ constraints that greatly resemble the constraints we see in the set partitioning reformulation of our ship scheduling problem. Consult Glover (1977) for a thorough explanation of surrogate constraints.

The outline of this heuristic is as follows. A set of candidate routes is defined. Initially this is the set of all the generated routes. The profit of each of the routes in the set is evaluated, and one of them is selected. The choice of one route excludes all other routes for the given ship and all routes that contain at least one of the cargoes in the chosen route. These routes are removed, and the set candidate set is reduced before the next iteration. The construction is finished when the candidate set is empty. The method is then restarted a given number of times with a modified evaluation function.

The route evaluation is based on the combined objectives of obtaining feasibility (serving all orders) and getting as good an objective function value as possible. Between each restart of the heuristic, the evaluation is changed in order to learn from difficulties or successes obtained in the previous iterations. Each cargo has associated with it a weight, w_i. These weights are used for learning, and are adjusted before each restart. The initial value is 1, and is increased by the parameter s_w before the next restart if the cargo is not represented in the solution. The order-weights are normalized every iteration of the constructive procedure. In this paper we use the normalization function $1/(N_i{*}n)$, where N_i is the number of ships covering order i and n is an integer parameter. This way more emphasis is given to orders that can be serviced by progressively fewer ships during the construction phase.

In addition to the order weights we have introduced order_ship weights u_{iv} in order to guide the search towards diverse solutions. The initial value is 1, and the value is increased by the parameter s_u if the given order-ship combination was part of the last solution.

We have also introduced an element of forgetting into the learning procedure. We define a parameter f, (<1), that is multiplied with both order weights and order-ship weights every restart of the construction.

The value function of a route, Z_r, is defined as the profit of each route c_r, multiplied by the sum of the weights of the orders covered by that route divided with the number of orders in the route. This gives emphasis to both selecting profitable routes and covering difficult orders. Recall that the w_i's will be incremented to give more emphasis on orders difficult to cover. Finally we divide by the sum of the order ship weights. Let *I*(*r*) be the set of orders covered by the route *r* and let *v* be the ship associated with the route. The evaluation function is given in equation (6).

$$Z_r = c_r \frac{\sum_{i \in I(r)} w_i f(N_i, n)}{\sum_{i \in I(r)} u_{iv}} \qquad (6)$$

Routes are selected with Probabilistic Move Acceptance as introduced by Løkketangen and Glover (1996). The routes are sorted by decreasing evaluation value. A selection probability *p* is defined. Consider the best route not yet considered. Draw against *p* to decide whether or not to choose the route. Continue until a route is chosen or all candidates have been considered. If the latter is the case, choose a random route. The overall heuristic can be stated as follows:

1. Initialize cargo learning weights (all w_i and u_r set to 1)
2. Initialize the candidate set
3. Evaluate all routes in the candidate set and heap them.
4. Select the best route, subject to Probabilistic Move Acceptance
5. Insert the selected route into the current solution.
6. Remove all infeasible routes from the candidate set.
7. If the candidate set is not empty, GoTo 3.
8. If all cargoes are serviced, update the column learning weights (u_r). Otherwise, update cargo learning weights (w_i)
9. If we have not reached the maximum number of solutions, GoTo 2.

3.3. Improving solutions by local search

In multi-start local search heuristics a number of different initial solutions are generated by a constructive heuristic and improved by local search. See Marti (2003) for a general description of this framework. Multi-start heuristics are efficient for problems where solutions can be easily constructed and where it is difficult to make local search neighborhoods that can move far in the feasible search space. We believe this is the case in this paper since we consider highly constrained ship scheduling problems.

We use the current local search heuristic in TurboRouter to improve a selection of the solutions generated by the procedure described above. The local search is described by Brønmo et al. (2005) and is composed by five neighborhood operators that can be described as operators on the current solution:

1-resequence removes one cargo from its ship, and tries to find a better insertion to the same ship, while 2-resequence removes two cargoes from the same ship and tries to find a better insertion to the same ship. Reassign removes one cargo from its ship and tries to find a better insertion to the other ships. 2-interchange tries to swap two cargoes from different ships, while 3-interchange tries to swap three cargoes from three different ships.

The local search heuristic is divided into a quick and an extended version. The quick version only use the three neighborhoods 1-resequence, reassign and 2-interchange, while the extended local search heuristic uses all neighborhoods. In both versions the neighborhoods are given frequencies, so that different emphasis can be given to neighborhoods of different computational complexities.

Brønmo et al. (2005) generate initial solutions by a partly randomized insertion heuristic, and choose a number of the best initial solutions to improve by local search. We choose a slightly different strategy, where we focus on the structural diversity of our initial solutions as well as the quality. Let A and B represent two different solutions to the problem, and let I be the set of available contract and spot cargoes. Now, let U_i^{AB} be 1 if cargo *i* is lifted by the same ship or is not lifted in both solutions *A* and *B*, and 0 otherwise. The distance, d_{AB}, between the two solutions *A* and *B* can now be defined as follows:

$$d^{AB} = \frac{1}{|I|}\sum_{i \in I}(1 - U_i^{AB}) \qquad (7)$$

The distance can be described as the number of cargo-ship combinations that are different between the two solutions divided by the total number of cargoes. For other discussions of distance measures between solutions, see Sørensen (2006), and Løkketangen and Woodruff (2005).

Our selection strategy can now be described as follows: Define a minimum distance d and a maximum number of selected solutions m. Select the best initial solution. Then, work through the list of initial solutions sorted by objective value and measure the distance to all the solutions that are selected so far. If the distance to all selected solutions is greater than *d*, include this solution in the set of selected solution. The selection procedure terminates when *m* is reached or when the list of initial solutions is exhausted.

4. Computational experience

If all routes are generated, the optimal solution to the problem at hand can be established by solving a set partitioning problem via standardized software. We have tested our heuristic and compared it to the optimization procedure and the multi-start local search method that are described by Brønmo et al. (2005).

4.1. Case descriptions

All the presented cases are based on real world data from the TurboRouter system. Cases 1 to 3 are represented by a chemical commodities shipping company operating between Europe and the Caribbean. In these cases the number of ships varies from 3 to 4.

Case 4 to 8 is collected from a shipping company operating in northern Europe, transporting dry bulk commodities such as rock, iron ore and aluminum, and case 9 is collected from another shipping company operating within the same segment and geographical area.

Table 1 Case descriptions

	Case								
	1	2	3	4	5	6	7	8	9
Case descriptions									
Planning horizon in days	75	110	140	20	20	27	20	27	42
# of cargoes	9	17	21	18	24	30	27	34	19
# of ships	3	4	4	4	7	9	9	11	2

We have modified the test cases slightly in order to simulate a situation where the shipping company faces a good market and the scheduling problem is highly constrained. In some cases we have removed some of the ships, and in other cases we have introduced some additional cargoes. Finally, we have reduced the width of some of the time windows in some of the cases.

In order to find optimal solutions and characterize the test cases, we used the a priori column generation code of Brønmo et al. The results are presented in Table 2. All the cases were solved. The ratio between the first two rows in the table, i.e. the number of feasible schedules per cargo combination, gives a good measure of how constrained the testcase is. If this ratio exactly equals 1, the route of a given cargo set is determined by the constraints. If the case was unconstrained, the ratio would be very large. This means that all the cases presented here are rather constrained, but there is a big difference between case 3, that has an average of 40 routes per cargo and cases 5 and 6 where this average is very close to 1.

4.2. Finding good parameter values

We have performed some experiments with case 7 in order to find sensible parameter values. In these tests we generated 100 initial solutions, where 6 of those were improved by local search (*m*=6), and finally the three best solution was improved by the extended local search. We tuned one parameter at a time in the following order finding the given values: s_w, s_u, *n*, *f*, *d*, *p*.

Figure 1 shows how the final distance varies with the minimum selection distance parameter d for case 7. The final distance is given as an average of the three possible distances between the three best solutions (1-2, 1-3, 2-3).

Figure 1 shows that the final distance did not vary much with the value of *d* for case 7. Based on Figure 1 we chose the value $d = 0.5$ for case 7. The same procedure was used to find values for all the other parameters. We have chosen the following parameter values: $p = 0.8$, $s_w = 0.5$, $s_u = 0.5$, $n = 1$, $f = 0.9$, $d = 0.5$.

4.3. Computational results

The tests were performed on a PC with a Pentium IV, 3.2 GHz processor and 3.0 GB RAM under Windows XP. The multi-start heuristic was developed in C++. We have used the following abbreviations:

- *learning_multi* is the multi-start local search heuristic presented here with the evaluation function f, where the main focus is feasibility
- *random_multi* is the multi-start local search heuristic presented by Brønmo et al. (2005).

Table 2 Information from the set partitioning approach for the multiple cargo cases

	Case								
	1	2	3	4	5	6	7	8	9
Set partitioning info									
# of feasible schedules	327	21 935	749 409	34 426	2 669	21 696	8 791	51 392	67 177
# of cargo set – ship combinations	113	2 360	18 510	10 346	2 668	21 693	6 883	45 530	59 581
avg. # of cargoes per feasible schedule	4.1	5.4	7.7	5.4	3.0	3.8	3.2	4.2	6.4
max # of cargoes in a feasible schedule	5	9	12	9	5	6	5	7	11
column generation CPU (s)	<1	66	3500	36	2	20	9	38	101
set partitioning CPU (s)	<1	<1	3	1.4	<1	1.7	<1	5	27

In both heuristics 100 initial solutions were generated. 6 solutions were subjected to the quick local search. As we have described in Section 3.4, the selection criteria were slightly different between the two methods. In the *random_multi* method, the 6 best solutions were chosen. This corresponds to setting $d = 0$ in the *learning_multi* method. Finally, in both heuristic methods the best solution from the quick local search phase was subjected to the extended local search. Optimal solutions were found by the priori column generation method described in Brønmo et al. (2005). The results are presented in Table 3 and are given as the smallest optimality gap, average optimality gap and average CPU time.

From the table it can be seen that the *learning_multi* heuristic gives high quality solutions to the cases tested. In case 1 and 9 the optimal solution was found in all 10 runs, while in case 4 the average optimality gap is less than 0.1 %. We have compared the average optimality gaps from the two methods since both methods contain stochastic elements. The *learning_multi* solutions was on average better than the solutions from the *random_multi* heuristic in 4 of the cases, while in 4 of the cases the average optimality gaps were equal. The *random_multi* heuristic was quicker in 8 of the 9 cases, and the smallest optimality gap from 10 runs was the same or better in 8 of the cases.

The local search setting we have used so far is aimed at finding one near optimal solution quickly. This method is well suited in a practical situation where the decision maker quickly wants to have a good picture of the feasibility and economy of the set of cargoes at hand, for instance when a new spot cargo shall be evaluated. Another setting occurs when the decision maker is actually planning the schedule for his fleet of ships. Due to different practical constraints that cannot be modeled in the system, the solutions might need to be adjusted to the real world, and hence the value of the solution can change. In such a situation it might be very useful to have a set of diverse high quality solutions to choose from. Therefore we have also tested a different local search setting for the *learning_multi* heuristic where the number of local searches is extended. We still generate 100 initial solutions, and 6 solutions are subjected to the quick local search. Now, 3 of these are subjected to the extended local search. Otherwise the parameters are unchanged. The results can be seen in Table 4 where we report the average optimality gap of the first and third extended local search solution. The average distance is given as the average over the 10 runs over the average of the three possible distances between the three best solutions. Finally, the average CPU time is reported.

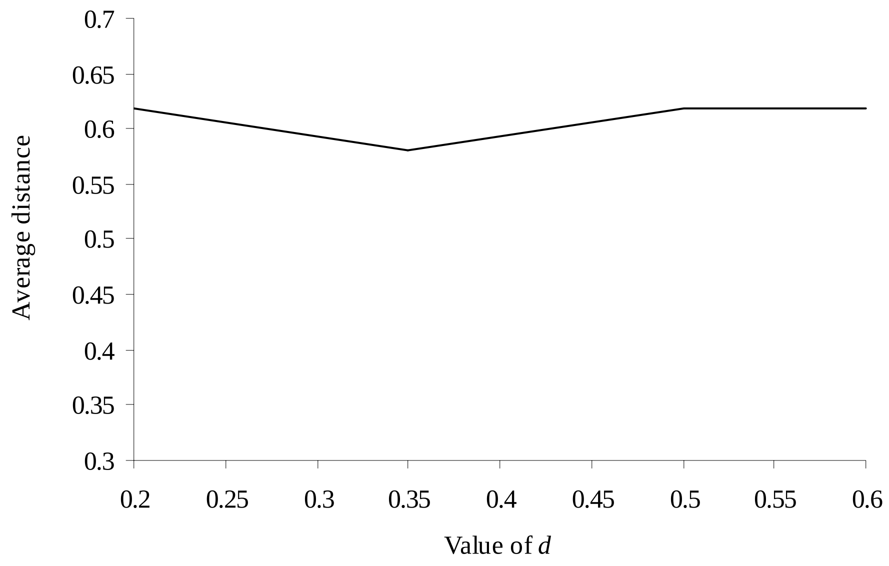

Figure 1 The effect of the value of *d* for case 7

Table 3. Computational results

	Case								
	1	2	3	4	5	6	7	8	9
learning_multi									
Smallest optimality gap (%)	0	0	0.3	<0.1	0	0.9	0.9	0.3	0
Average optimality gap (%)	0	0.3	0.3	<0.1	0.6	2.7	1.7	0.7	0
Average CPU (s)	<1	4	18	12	6	31	12	62	68
random_multi									
Smallest optimality gap (%)	0	0	0	0	0	0	0.9	0.6	0
Average optimality gap (%)	0	0.1	0.3	<0.l	0.5	2.1	1.3	0.9	0
Average CPU (s)	<1	2	4	2	5	9	7	16	3

Table 4 shows that the new setting gives a set of diverse high quality solutions with a somewhat increased computational effort. The objective difference between the first and the third solution is very small for most of the cases. The *random_multi* heuristic gives slightly higher quality in many of the cases, while the *learning_multi* heuristic gives more diverse solutions in all cases except case 1, where the average distance is equal.

Table 4. Computational results, several solutions

	Case								
	1	2	3	4	5	6	7	8	9
learning_multi									
Average best solution	0	0.3	0.3	<0.1	0.3	1.6	1.6	0.7	0
Average third best solution	-	1.9	3.7	1.9	1.7	3.8	2.7	1.7	7.6
Average distance	0.57	0.43	0.44	0.53	0.64	0.57	0.51	0.62	0.33
Average CPU	<1	4	20	13	7	33	15	67	68
random_multi									
Average best solution	0	0.1	0.1	0.2	0.3	0.8	1.4	1.1	0
Average third best solution	-	1.1	1.3	1.7	1.8	3.8	2.5	2.4	5.5
Average distance	0.57	0.30	0.35	0.32	0.53	0.50	0.46	0.60	0.27
Average CPU	<1	3	5	2	5	10	9	17	4

5. Concluding remarks

In a DSS, the DM is often better served with a set of good, diverse, solutions, to a model than with the best solution produced by that model. This is due to all the factors that cannot easily be modeled, due to modeling, solving or other considerations.

We therefore have presented a self-learning multi-start local search heuristic that is designed for solving ship scheduling problems where serving all cargoes is difficult. The heuristic is based on a set partitioning formulation of the ship scheduling problem where columns are generated a priori by an enumeration tree and an insertion heuristic. The resulting set partitioning problem is then solved repeatedly by a stochastic self-learning constructive procedure. Learning weights are defined for both cargoes and columns, and the aim is to guide the solution process towards diverse high quality solutions. Finally, a selection of the solutions from the construction process is subjected to the local search procedure presented by Brønmo et al. (2005).

In the computational study the method is tested on a set of test cases based on real world data. The cases are highly constrained, so that feasibility is difficult. The computational results show that the heuristic produces high quality solutions to the cases tested, which indicates that the heuristic is well suited for constrained ship scheduling cases. Finally, the results show that the heuristic is able to give several high quality solutions for the planner to choose from. This can be particularly useful in a practical planning situation where the model solved is no longer correct due to some real-world considerations that cannot be easily modeled in a computer system.

References

Bausch, D. O., G. G. Brown and D. Ronen. (1998). “Scheduling short-term marine transport of bulk products”. Maritime Policy and Management 25(4), 335-348

Brønmo, G., M. Christiansen, K. Fagerholt and B. Nygreen. (2005). “A multi-start local search heuristic for Ship Scheduling – A Computational Study”. Computers and Operations Research, available online June 28th 2005, 17p.

Cho S-C. and A. N. Perakis. (2001). “An improved formulation for bulk cargo ship scheduling with a single loading port”. Maritime Policy and Management 28(4), 339-345.

Cordeau J-F, Laporte G, Mercier A. (2001). “A unified tabu search heuristic for vehicle routing problems with time windows”. Journal of the Operational Research Society 52, 928–936.

Christiansen M. and K. Fagerholt,(2002). ”Robust ship scheduling with multiple time windows,” Naval Research Logistics 49(6), 611-625.

Christiansen, M., K. Fagerholt and D. Ronen, (2004). “Ship Routing and Scheduling: Status and Perspectives”. Transportation Science 38, 1-18.

Glover, F. (1977). “Heuristics for integer programming using surrogate constraints,” Decision Sciences 8, 156 – 166.

Fagerholt, K. (2002). ”A computer-based decision support system for vessel fleet scheduling - Experience and future research”. Decision Support Systems 1041, 1-13.

Fagerholt K. and M. Christiansen. (2000). “A combined ship scheduling and allocation problem”. Journal of the Operational Research Society 51(7), 834-842.

Kim S-H. and K-K. Lee. (1997) ”An optimization-based decision support system for ship scheduling”. Computers and Industrial Engineering 33(3-4), 689-692.

Løkketangen, A. and F. Glover. (1996). “Probabilistic Move Selection in Tabu Search for 0/1 Mixed Integer Programming Problems”. In “Metaheuristics: Theory and Applications”, Kluwer, 467 - 488.

Løkketangen, A. and F. Glover. (1997). “Surrogate Constraint Analysis - New Heuristics and Learning Schemes for Satisfiability Problems”. In “Satisfiability Problem: Theory and Applications”, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 35.

Løkketangen, A. and D.L. Woodruff. (2005). “A Distance Function to Support Optimized Selection Decisions”. Decision Support Systems 39(3), 309 - 332.

Marti, R. (2003). “Multi-Start Methods” In: Glover FW, Kochenberger GA. (Eds.) “Handbook of Metaheuristics”, Springer, 355-368.

Scott, J. L. (1995). ”A transportation model, its development and application to a ship scheduling problem”. Asia-Pacific Journal of Operational Research 12, 111-128.

Sørensen, K. (2006). “Distance measures based on the edit distance for permutation type representations”. Forthcoming in Journal of Heuristics.