

An Effective Ship Berthing Algorithm

Andrew Lim

Department of Computer Science
National University of Singapore
Lower Kent Ridge Road
Singapore 119260

Abstract

Singapore has one of the busiest ports in the world. Ship berthing is one of the problems faced by the planners at the port. In this paper, we study the ship berthing problem. We first provide the problem formulation and study the complexity of the problem with different restrictions. In general, the ship berthing problem is NP-complete, although, some of its variants may be solved quickly. While a geometrical model is intuitive, the model cannot be easily extended to handle clearance constraints and berth restriction. Rather than solving the problem geometrically, we transform the problem into the problem of fixing directions of edges in graph to form directed acyclic graph with minimal longest path. Since the problem is NP-complete, solving the problem exactly in polynomial time is highly unlikely. As a result, we devise a fast and effective greedy algorithm to can generate good solutions. The greedy method together with a tabu search like post optimization algorithm is able to return optimal or near optimal solutions.

1 Introduction

Situated at the crossroads of the world, the Port of Singapore is one of the world's busiest port. Every few minutes a ship arrives or departs the port. Every month the port handles more than one million transshipment containers.

When a ship arrives at the port, the planners must first decide where to berth the ship for the unloading and loading of containers. For the containers that are to be unloaded, the planners must decide where to place these containers in the yard. The wharf line of the port is divided into sections, and no ship can be berthed across any two sections. Which section to assign a ship to and exactly where to berth a ship within a section depend on factors like the locations of containers to be loaded and unloaded, the physical (i.e. depth of the berth) and resource limitations (i.e. suitability of quay crane) of each berth. A sketch of a port

is given in Figure 1. The allocation of ships to sections and placement of containers in the yard is studied by Lim [Lim, 1998]. The approach used is a variant of graph partitioning problem. Allocation of vessels to sections were also studied by Brown [Brown *et al.*, 1994; 1997].

One of the subproblems in [Lim, 1998] is the ship berthing problem. The ship berthing problem was studied by Heng [Heng, Khoong, and Lim, 1996] using a mixed integer linear programming model. Their model assumed constant inter-ship clearance distance and constant end-berth clearance distance. While their model worked reasonably well on historical test data, it did quite badly on fully packed test cases. Their approach is also computationally intensive. As the ship berthing problem is only a subproblem which is called many times in the berth yard planning system, it needs to be computationally efficient. Heng's version of the problem is also very closely related to the offline version of the dynamic storage allocation problem [Wilson *et al.*, 1995]. Because the dynamic storage allocation problem is a special case of the ship berthing problem, the ship berthing problem is NP-Complete [Garey and Johnson, 1979]

2 Problem Formulation

Ships come in different lengths and they arrive at the port at different times to be berthed. Every ship has an expected duration of stay which may be different from another ship. To berth a ship is to place the ship along the wharf line of a section. Once a ship is berthed, it will not be moved until its departure time. When two ships are berthed side by side, a certain minimum inter-ship clearance distance must be observed. Each ship has an inter-ship clearance distance which is dependent on the ship's length. The minimum inter-ship clearance distance of two ships berthed side by side is the larger of the two ships' inter-ship clearance distances. If a ship is berthed at the end of a section, a certain end-berth clearance distance must be observed. This end-berth clearance distance is not fixed and is dependent on the ship. A ship can also be given a fixed berthing location within a section. A ship may also be prohibited from

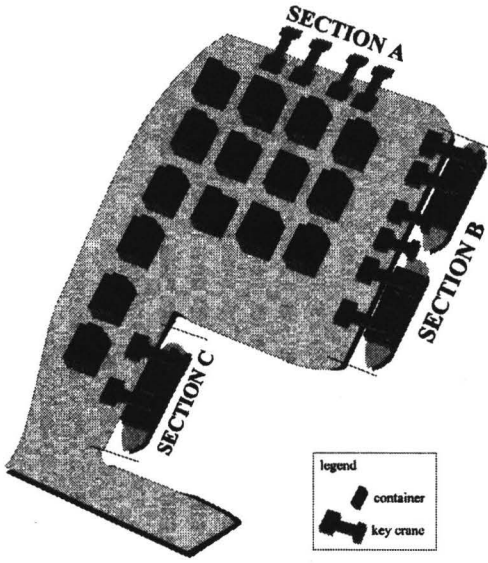


Figure 1: Sketch of a port

berthing at certain parts of a section. A berth plan for a set of ships in a section is the exact locations of ships within the section.

We can represent a *ship* geometrically by a rectangle such that the height of the rectangle is the length of the ship and the length of the rectangle is the duration of its stay. The left edge of the rectangle represents the arrival time of the ship. The right edge represents the departure time of the ship. A *section* can be represented geometrically by an infinitely long rectangle where its height represents the length of the section and the length represents the time axis. We can associate a coordinate with the left bottom corner of the geometric representation of each ship. The x -coordinate is fixed as it represents time of arrival, but the y -coordinate is not known unless the ship has a preassigned berth location. The Ship Berthing Problem is to decide the y -coordinates of the set of boxes in the long rectangle of the section such that all rectangles representing ships are non-overlapping and are within the rectangle of the section with all clearance distances are satisfied. Figure 2 clarifies the transformation.

Let us define the problem formally. Let $\mathcal{S}$ be the set of ships $\{S_1, S_2, \dots, S_n\}$. Let $b_i, l_i, t_i, d_i, c_i^a, c_i^b$, and F_i be the start wharf mark of the berthing location, length, arrival time, duration of stay, inter-ship clearance distance, end-berth clearance distance, and the set of forbidden berth positions respectively of Ship S_i . When S_i is berthed at b_i , S_i will occupy wharf mark from b_i to $b_i + l_i$ from time t_i to time $t_i + d_i$. F_i is a set of intervals in the section in which ship i cannot be berthed. If S_i is berthed at b_i , the interval $[b_i, b_i + l_i]$ should not intersect with any interval in F_i . For a clearer picture of the above definitions, please refer to Figure 2.

The optimization version of the Ship Berthing Problem

(SBP) is defined Figure 2, where L is the wharf length:

The decision version of the Ship Berthing Problem is similar to the SBP problem. All we need to do is to remove the objective function to Minimize L and assume that L is given.

2.1 Complexity

Lemma 1 If $\forall S_i \in \mathcal{S}, l_i = C_1, c_i^a = C_2, c_i^b = C_3$, where C_1, C_2 and C_3 are constants, the ship berthing problem can be solved in $O(n \log n)$ time.

Proof: The above problem can be transformed to the problem of coloring of interval graphs by first partitioning the section into $\lfloor \frac{L-2C_2+C_2}{C_1+C_2} \rfloor$ fixed berths and sorting the ships, in non-decreasing order based on the arrival times, followed by assigning the ships in the sorted order to the fixed berths using the criteria of most recently used available berth. This algorithm takes $O(n \log n)$ time. $\square$

Biro[Biro et al., 1992] showed that 1-precoloring of interval graph is polynomial time solvable and 2-precoloring is NP-Complete. Using his results, if $\forall S_i \in \mathcal{S}, l_i = C_1, c_i^a = C_2, c_i^b = C_3$, where C_1, C_2 and C_3 are constants, and if some of the ships are preassigned to a single berth, then the problem remains polynomial time solvable. If some of the ships are preassigned to 2 berths, then the problem becomes NP-complete even if the ships have the same length and clearance distances.

In the next section, we will show that the berth planning problem is NP-complete. Our reduction uses the PARTITION problem which is known to be NP-Complete. The definition of the partition problem is given below:

PARTITION Given a finite set $A = \{a_1, a_2, \dots, a_x\}$ and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$, is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A-A'} s(a)$?

Lemma 2 The berth planning problem is NP-Complete.

Proof: If we set all inter-ship clearances and end-berth clearances to be zero the berth planning problem is exactly the same as the dynamic storage allocation problem [Garey and Johnson, 1979]. The dynamic storage allocation problem has been shown to be NP-complete by Stockmeyer using the 3-PARTITION problem. However, this result is not published [Garey and Johnson, 1979]. In this section, we shall sketch a simple reduction using the PARTITION problem. Let $T = \sum_{a \in A} s(a)$. We use the triple (l_i, t_i, d_i) to represent the ship S_i , where l_i, t_i, d_i are the length, arrival time and the duration of stay of the ship respectively. Let $\mathcal{S} = \mathcal{S}_A \cup \mathcal{S}_B$. $\mathcal{S}_A = \{S_1, S_2, S_3, \dots, S_9\}$, where

$$\begin{aligned} S_1 &= (T + 2, 0, 1) & S_2 &= (1, 0, 3) \\ S_3 &= (\frac{T}{2} + 1, 1, 1) & S_4 &= (1, 1, 3) \\ S_5 &= (\frac{T}{2}, 1, 1) & S_6 &= (1, 2, 3) \\ S_7 &= (\frac{T}{2}, 3, 1) & S_8 &= (\frac{T}{2} + 1, 3, 1) \end{aligned}$$

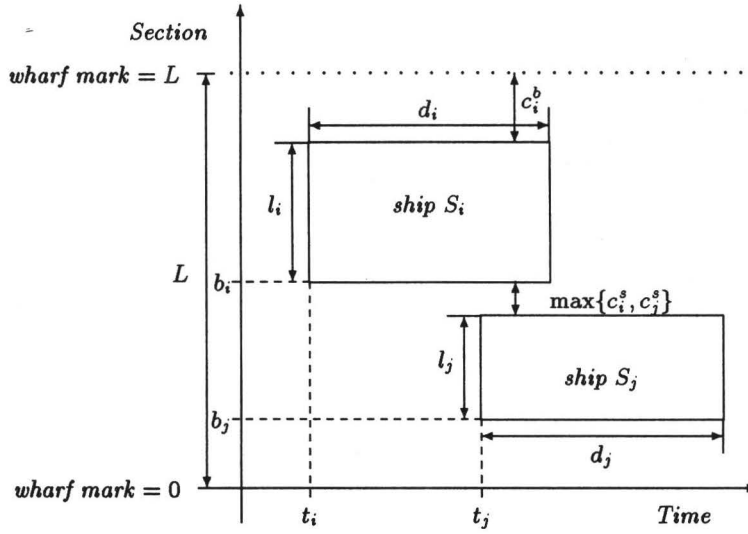


Figure 2: Geometric Representation to Graph Transformation

Minimize L
Subject to :

$$\begin{aligned}
 c_{ij} &\geq c_i^s \quad \forall S_i, S_j \in \mathcal{S} \quad \text{s.t.} \quad t_i \leq t_j \leq t_i + d_i \quad \text{or} \quad t_j \leq t_i \leq t_j + d_j \\
 c_{ij} &\geq c_j^s \quad \forall S_i, S_j \in \mathcal{S} \quad \text{s.t.} \quad t_i \leq t_j \leq t_i + d_i \quad \text{or} \quad t_j \leq t_i \leq t_j + d_j \\
 b_i + l_i + c_{ij} &\leq b_j + (1 - x_{ij})M \quad \forall S_i, S_j \in \mathcal{S} \quad \text{s.t.} \quad t_i \leq t_j \leq t_i + d_i \quad \text{or} \quad t_j \leq t_i \leq t_j + d_j \\
 b_j + l_j + c_{ij} &\leq b_i + x_{ij}M \quad \forall S_i, S_j \in \mathcal{S} \quad \text{s.t.} \quad t_i \leq t_j \leq t_i + d_i \quad \text{or} \quad t_j \leq t_i \leq t_j + d_j \\
 x_{ij} &\in \{0, 1\} \quad \forall S_i, S_j \in \mathcal{S} \quad \text{s.t.} \quad t_i \leq t_j \leq t_i + d_i \quad \text{or} \quad t_j \leq t_i \leq t_j + d_j \\
 c_i^b &\leq b_i \leq L - l_i - c_i^b \quad \forall S_i \in \mathcal{S} \\
 b_i + l_i &\leq p + (1 - x_{if_j})M \quad \forall S_i \in \mathcal{S} \quad \forall f_j = [p, q] \in F_i \\
 q &\leq b_i + x_{if_j}M \quad \forall S_i \in \mathcal{S} \quad \forall f_j = [p, q] \in F_i \\
 x_{if_j} &\subseteq \{0, 1\} \quad \forall S_i \in \mathcal{S} \quad \forall f_j \in F_i \\
 M &\text{ is a constant } > \sum_{\forall i} l_i + \max\{c_i^b, c_i^s\}
 \end{aligned}$$

Figure 3: Mathematical Model of the Problem

$$S_9 = (T + 2, 4, 1)$$

For the set $\mathcal{S}_A$, if $L \leq T + 3$, there are only 2 berth plans (in fact one is a reflection of the other). In both of these plans, there are two unoccupied regions of equal size. One is from wharf mark 1 to $\frac{T}{2} + 1$, from time unit 2 to time unit 3. The other is from wharf mark $\frac{T}{2} + 2$ to $T + 2$ from time unit 2 to unit 3. Let $\mathcal{S}_B = \{S_{10}, S_{11}, \dots, S_{9+|A|}\}$ where $S_i = (s(a_i - 9), 2, 1)$. Clearly, if there is a berth plan where $L \leq T + 3$ then the set $\mathcal{S}_B$ must be partitioned into two disjoint subsets such that the sum of lengths of ships in the two subsets are the same. It is clear that the transformation will only take polynomial time. It is also clear that the problem is in NP. Hence, the berth planning problem is NP-complete. $\square$

Since the berth planning problem is NP-Complete, it is highly unlikely that a fast algorithm can be devised to solve the problem optimally.

3 Graph Model

While the geometrical model is visually attractive, it cannot be extended to handle clearance constraints and berth restriction constraints easily. In this section, we shall transform our geometrical representation of the Ship Berthing Problem to a graph, $\mathcal{G}$. For each ship $S_i \in \mathcal{S}$, we have a vertex v_i in $\mathcal{G}$. The weight of vertex v_i is set to l_i which is the length of the ship S_i . If two ships $S_i, S_j \in \mathcal{S}$ have time overlap, then there is an edge (v_i, v_j) linking the 2 vertices v_i and v_j . The weight of the edge (v_i, v_j) is the larger of the two ships inter-ship clearance distances, i.e. $\max\{c_i^s, c_j^s\}$. For each ship S_i , there are 2 additional vertices v_i^{et} and v_i^{eb} . These vertices have weights 0. There is a directed edge $\langle v_i^{et}, v_i \rangle$ and another directed edge $\langle v_i, v_i^{eb} \rangle$. The weight of these directed edges is the end-berth clearance c_i^e .

If a ship S_j is required to be fixed at a particular berth location k (i.e. $b_j = k$ and S_j will occupy wharf mark from k to $k + l_j$) within the section, we will add two vertices u_1 and u_2 and two edges $\langle u_1, v_j \rangle$ and $\langle v_j, u_2 \rangle$. u_1 and u_2 will have vertex weight of 0. $\langle u_1, v_j \rangle$ and $\langle v_j, u_2 \rangle$ will have edge weight of k and $L - (k + l_j)$. Similarly, a ship S_j may not be permitted to berth from wharf mark p to wharf mark q . We can handle this situation by creating a fictitious ship which is fixed at location p with length $q - p - c_j^s$ which has a hypothetical time overlap with only ship j . Using such a transformation, berth restrictions for ships can be handled consistently.

At the end of the transformation, we have a graph consisting of directed and undirected edges. Let us pick an undirected edge $e_{ij} = (v_i, v_j)$. e_{ij} exists because ship S_i and ship S_j have a time overlap. This implies that both ship S_i and ship S_j cannot share any part of the section. If ship S_i is berthed at a lower wharf mark, (a wharf mark is a particular position in a section) than ship S_j , we will set the edge (v_i, v_j) to become $\langle v_i, v_j \rangle$. Similarly, if ship S_j is berthed at a lower wharf mark than

ship S_i , we will set the edge e_{ij} to go from vertex v_j to vertex v_i , i.e. $\langle v_j, v_i \rangle$. The ship berthing problem has been transformed to a problem of setting the directions of undirected edges in the graph such that the graph becomes directed acyclic and the longest path in the graph is minimized. The length of a path in our graph is the sum of all vertex weights and edge weights of all vertices and edges in the path.

The directed acyclic condition of the graph is important as the directions of the edges represent relative berth locations, therefore it is impossible to have a situation such that ship S_i is berthed at a lower location than ship S_j , and ship S_j is berthed at a lower location than ship S_k , and ship S_k is berthed at a lower location than ship S_i . Therefore, when we set the direction of the undirected edges we must not create any cycle. The length of the longest path in the directed acyclic graph created is the minimum length required in the section to berth the set of ships $\mathcal{S}$.

Lemma 3 *The Ship Berthing Problem can be transformed to a problem of fixing the directions of some edges in a graph such that the graph becomes directed acyclic and the longest path in the graph is no more than the section length.*

Proof: Obvious from the above discussion. $\square$

It is clear that the optimization version of the Ship Berthing Problem can be transformed to a problem of fixing the directions of edges in a graph such that the graph becomes directed acyclic and the longest path in the graph is minimized. In the worst case, only $O(n)$ of the decision versions of the problem need to be solved through the use of binary search on L .

4 An Effective Greedy Algorithm

In this section, we shall discuss our heuristic for fixing the edge directions in our graph representation to create a DAG with the minimum longest path. Before describing the algorithm, let us define the following. Let e_{ij} be the edge between vertices v_i and v_j . If e_{ij} is undirected, it can be set in two directions, namely from v_i to v_j or from v_j to v_i .

Let β_{ij} and β_{ji} be defined as follows :

$$\beta_{ij} = L_{In}(v_i) + w(v_i) + w(e_{ij}) + w(v_j) + L_{Out}(v_j)$$

$$\beta_{ji} = L_{In}(v_j) + w(v_j) + w(e_{ij}) + w(v_i) + L_{Out}(v_i)$$

$L_{In}(v_i)$ is the longest incoming path of vertex v_i . $L_{Out}(v_i)$ is the longest outgoing path from vertex v_i . $w(v_i)$ and $w(e_{ij})$ are the weights of vertex v_i and edge (i, j) respectively.

For every undirected edge, e_{ij} , the potential of edge, $\Phi(e_{ij})$ is given by $\max\{\beta_{ij}, \beta_{ji}\}$. The algorithm of our heuristic is given in Figure 4. The algorithm first computes the potential of every undirected edge. Next the algorithm selects the undirected edge with the highest potential. In the event of a tie, a second criterion is used. This criterion is $|\beta_{ij} - \beta_{ji}|$, the larger the better. Once the edge is selected, the direction of the edge is set

such that the longest path through that edge is as small as possible. In the event of a tie, the edge is set from the vertex with lower degree to the vertex with higher degree. Next, the algorithm updates the longest incoming paths and outgoing paths, and potentials of all affected edges. This may be computed by doing 2 topological sweeps (2 passes, computing the longest incoming path for each vertex in one pass and computing the longest outgoing path for each vertex in the second pass). A topological sweep takes $O(|V(\mathcal{G})| + |\mathcal{E}(\mathcal{G})|)$ time. Updating will take significantly less time in practice even though the worst case time complexity stays the same. Repeat the process of finding the edge with the highest potential, fixing its direction and updating the potential of the rest of the undirected edges until the directions of all edges are set. The longest path in the graph defines the minimum section length needed. The longest incoming path of a vertex v_i defines the wharf mark whereby the ship i is to be berthed.

We shall use a simple example to illustrate our heuristic. In our example, we assume that all clearance distances are zero, i.e. no clearance needed. Note that at the end of the iterations, the longest incoming path for each vertex, v_i will determine the wharf mark of the ship S_i .

It can be easily seen that our algorithm takes at most $O(|E(\mathcal{G})|(|\mathcal{E}(\mathcal{G})| + |V(\mathcal{G})|))$. In practice, our algorithm runs in $O(|V(\mathcal{G})|^2)$ time.

Lemma 4 *Algorithm Greedy Berth-Planner does not create a cycle from a path.*

Proof: Let us consider a cycle $C = v_1 v_2 \dots v_{k-1} v_k v_1$. Without loss of generality, let us assume that the edge e_{k1} is the last edge to fix its direction in C , which is from v_k to v_1 , to create the cycle C . From our graph construction, it is clear that the length of the path $|p_{1k}| > 0$ as $p_{1k} = v_1 v_2 \dots v_k$. To set the edge e_{k1} to go from v_k to v_1 , β_{k1} must be less than or equal to β_{1k} (see algorithm). This implies

$$L_{In}(v_k) + L_{Out}(v_1) \leq L_{In}(v_1) + L_{Out}(v_k)$$

But,

$$L_{In}(v_k) \geq L_{In}(v_1) + |p_{1k}|$$

$$L_{Out}(v_1) \geq |p_{1k}| + L_{Out}(v_k)$$

This implies that

$$L_{In}(v_k) + L_{Out}(v_1) \geq L_{In}(v_1) + L_{Out}(v_k) + 2|p_{1k}|$$

Since $|p_{1k}| > 0$,

$$L_{In}(v_k) + L_{Out}(v_1) > L_{In}(v_1) + L_{Out}(v_k)$$

From the above, it is clear that our algorithm will never turn a path to a cycle. $\square$

Since the algorithm does not create any cycle, there will not be a situation where the algorithm is stuck, i.e. no matter which direction you fix an edge, you get a cycle. To be in such a situation, you must first have a cycle. From the above lemma, cycle detection is not needed in our heuristic.

Algorithm Greedy Berth-Planner

```

Step 1:  $\forall v_i \in V(\mathcal{G})$  set  $LongestIn(v_i) = 0$ 
        and  $LongestOut(v_i) = 0$ 
Step 2:  $\forall e_{ij} \in E(\mathcal{G})$  compute  $\Phi(e_{ij})$ 
Step 3: Find the undirected edge,  $e_{ij}$ , with
        the highest potential  $\Phi(e_{ij})$ .
        If there is a tie in the highest potential,
        pick the edge with the largest  $|\beta_{ij} - \beta_{ji}|$ .
Step 4: If  $(\beta_{ij} < \beta_{ji})$ 
        then set edge to go from  $v_i$  to  $v_j$ 
        else if  $(\beta_{ij} > \beta_{ji})$ 
        then set edge to go from  $v_j$  to  $v_i$ 
        else if  $deg(v_i) < deg(v_j)$ 
        then set edge to go from  $v_i$  to  $v_j$ 
        else set edge to go from  $v_j$  to  $v_i$ 
Step 5: Update the affected longest incoming and
        outgoing paths of vertices
Step 6: Update the potential of affected undirected edges
Step 7: If there is an undirected edge, goto 3
Step 8: End

```

Figure 4: Ship Berthing Heuristic

5 Local Search Post Optimization

Using our graph model, every arrangement is represented by at least an acyclic labeling. Given an acyclic labeling, the longest path and the berth position of every ship can be found in $O(|V(\mathcal{G})| + |\mathcal{E}(\mathcal{G})|)$ time. Our local search improvement approach is based on the Tabu Search method introduced by Glover [Glover, 1989; 1990]. It is applied to the solution generated by the greedy algorithm.

Given an acyclic labeling, AL , of a graph, we define the neighborhood solution of AL to be the set of acyclic labelings reachable from AL by swapping the acyclic labels of two vertices. Among all the possible swaps, we pick the swap that improves best result seen so far. If that is not possible, we will pick the swap, that is not forbidden by the tabu list, that returns the best result. Our tabu list's length is 10. It stores the 10 most recent swaps. The iterative improvement will be performed until the stopping criterion is reached. The stopping criterion is satisfied if there is no improvement of the best solution after 10 iterations or the current solution equals the lower bound.

6 Experimental Results

We have implemented our greedy heuristic and the local search algorithm in C++ using Microsoft Visual C++ development environment. We ran all our experiments on a pentium II 300MHz machine with 128 MBytes of memory. We tested our algorithms on 2 data sets and compare our algorithm with the lower bound. The lower bound is given by the maximum density. Density is given by the largest of all the sums of lengths of vessels at all points on the x (time) axis in the geometric representation. The first data set is the historical data from the Port. The second is manually generated. Each test case

Historical Test Set				
Size	LB	Greedy Algorithm	Local Search	PostOpt
			Result	Time
150	772	903	772	9
29	161	161	—	—
165	726	887	803	8
57	144	144	—	—
72	308	308	—	—
231	774	858	810	23
185	720	829	743	12
173	813	898	856	11
118	632	682	682	7
59	511	511	—	—
149	767	898	787	9

Table 1: Historical Test Data

Compact Test Set				
Size	LB	Greedy Algorithm	Local Search	PostOpt
			Result	Time
15	20	20	—	—
20	26	26	—	—
30	25	32	25	2
26	24	29	24	1
40	25	29	27	6
50	20	25	22	10
100	20	26	23	19

Table 2: Fully Packed Manually Generated Test Data

in the second data set is generated by first defining a rectangle and cutting smaller rectangles from the original rectangle for each test set. In the experiments, all clearance distances are set to 0. This is to facilitate easy checking of the quality of the solutions generated. The results are summarized in Table 1 and 2.

The first column of the tables is the number of ships. The second is the density. The third is the result obtained by the greedy method. The greedy took less than 1 seconds for all the test cases in our experiment. The fourth and fifth columns are the result and time taken in seconds respectively by the local search algorithm.

A sample solution of a test case is given in Figure 5.

7 Conclusion

In this paper, we formulated the ship berthing problem, analyzed its complexity and transformed the geometric version of the problem into a graph problem. We also proposed a fast and effective greedy algorithm which when used with a tabu search like post optimization routine can return optimal or near optimal solutions.

8 Acknowledgment

This research was motivated by a project the first author had with the Port of Singapore Authority. This

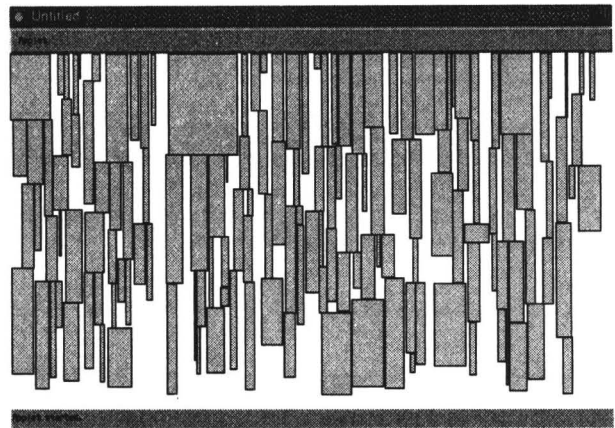


Figure 5: A Solution to a Real World Test Data with 149 ships

research was supported in part by the NUS Research Grant RP3972679.

References

- [Biro *et al.*, 1992] M. Biro, M. Hujter, and Zs. Tuza. Precoloring extension. I. Interval Graphs. *Discrete Mathematics*, 100, 1992.
- [Garey and Johnson, 1979] M. Garey and D. Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979.
- [Brown *et al.*, 1997] G.G. Brown, K.J. Cormican, S. Lawphongpanich, and D.B. Widdis. Optimizing Submarine Berthing with a Persistence Incentive. *Naval Research Logistics*, 44:301–318, 1997.
- [Brown *et al.*, 1994] G.G. Brown, S. Lawphongpanich, and K.P. Thurman. Optimizing Ship Berthing. *Naval Research Logistics*, 41:1–15, 1994.
- [Glover, 1989] F. Glover. Tabu Search - part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.
- [Glover, 1990] F. Glover. Tabu Search - part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.
- [Heng, Khoong, and Lim, 1996] K. Heng, C. Khoong, and A. Lim. A Forward Algorithm Strategy for Large Scale Resource Allocation. In *First Asia-Pacific DSI Conference*, pages 109–117, 1996.
- [Lim, 1998] A. Lim. Algorithms for Yard-Berth Scheduling. manuscript, 1998.
- [Wilson *et al.*, 1995] P.R. Wilson, M.S. Johnstone, M. Neely, and D. Boles. Dynamic Storage Allocation: A Survey and Critical Review. In *Lecture Notes in Computer Science 986*, pages 1–116, 1995.