operations research letters

Operations Research Letters 22 (1998) 105–110

The berth planning problem[1]

Andrew Lim

Department of Information Systems and Computer Science, National University of Singapore, Singapore 119260

Received 1 April 1996; revised 1 November 1997

Abstract

Singapore has one of the busiest ports in the world. Berth planning is one of the problems faced by planners. This paper studies the berth planning problem. We first formulate the problem and show that it is NP-Complete. We transform the berthing problem to a restricted form of the two-dimensional packing problem, use a graph theoretical representation to capture the problem succinctly, and propose an effective heuristic for the problem. Experimental results show that our heuristic performs well on historical test data. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Berth planning; NP-Complete; Graphs; Heuristic

1. Introduction

Singapore has one of the busiest ports in the world. In 1994, Singapore maintained its position as the world's busiest port in terms of shipping tonnage of ship arrivals. In that year, there were 101,107 ship arrivals with a shipping tonnage of 678.6 million gross tonnes [2]. One of the planning problems encountered at the port is to decide if a given set of ships can be berthed in a section of the port given certain berthing constraints.

In this paper, we shall discuss and formulate the ship berthing problem in Section 2. In Section 3, we transform the berthing problem into a restricted form of the two-dimensional packing problem and show that the berthing problem is NP-Complete. In Section 4, we propose a graph representation that captures the problem succinctly. In Section 5, we propose an effective heuristic for our graph representation. Section 6 reports our experimental results. Finally, in the last section, we will have our conclusion.

2. Problem formulation

The port is divided into several sections and no ship can be berthed across a section. 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 a inter-ship clearance distance which is dependent on the ship's length. The minimum inter-ship clearance distance of

[1] This research is part of a project the author has with the Port of Singapore Authority. The author was a senior technical consultant in that project. This research was supported in part by the NUS Research Front RP3972679.

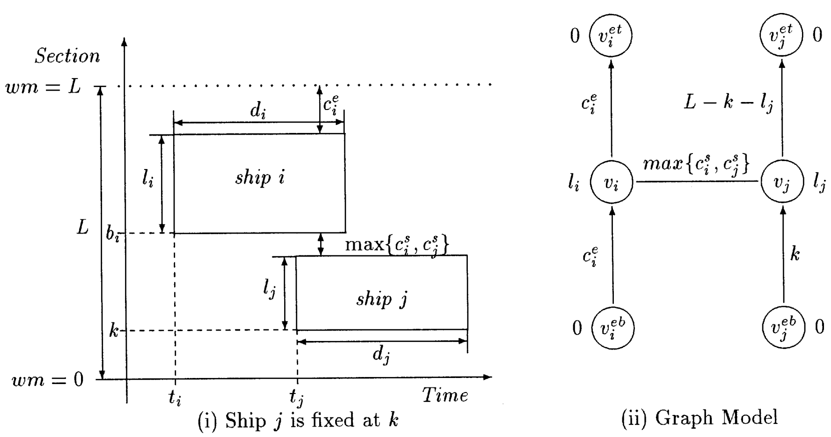

Fig. 1. Geometric representation to graph transformation.

two ships berthed side by side is the larger of the two ships' inter-ship clearance distances. If a ship is berth 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 berth plan for a set of ships in a section is the exact locations of ships within the section.

The decision version of the berth planning problem (BPP) is given below. Given a set of ships (where each ship i may have a fixed berth location, b_i, and has a length (l_i), time of arrival (t_i), duration of stay (d_i), inter ship clearance (c_i^s), end berth clearance (c_i^b)) is there a feasible berth plan for the ships for a section of length L.

The optimization version of the Berth Planning Problem (OBPP) is similar to the BPP problem except that the exact location of each ship in the section must be found and the maximum amount of space used in the section at any time is minimized.

3. Transformation and complexity

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 (which includes time clearance for departure). 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 fixed berth location. The berth planning problem (BPP) 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. Fig. 1(i) clarifies the transformation.

The berth planning problem is NP-Complete [1]. This can be established by reducing the set partition problem [1] to the berth planning problem.

4. Graph representation

We transform our geometrical representation of the berth planning problem to a graph, $\mathscr{G}$. Let the ships to be berthed in section $\mathscr{S}$ be numbered from 1 to n. n is the total number of ships to be berthed in $\mathscr{S}$. For each ship i, $1 \leqslant i \leqslant n$, we have a vertex v_i in $\mathscr{G}$. There is a weight in each vertex v_i, this weight is given by the length of the ship l_i. Let s_i and s_j be 2 ships i and j, $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant n$. Their corresponding vertices in $\mathscr{G}$ are v_i and v_j, respectively. If both ships have a time overlap, then there is an edge (v_i, v_j) linking the 2 vertices v_i and v_j. For each ship i, $1 \leqslant i \leqslant n$, there are two additional vertices v_i^{et} and v_i^{eb}. These vertices have weights 0. There is a directed edge $\langle v_i^{eb}, v_i \rangle$ and another directed edge $\langle v_i, v_i^{\mathrm{ct}}\rangle$ for each $i\in\{1,\ldots,n\}$. The weight of these directed edges is the end berth clearance c_i^e for each ship i. A particular ship j may be required to be fixed at a particular berth location within in the section. This is due to the fact that the ship may need certain type of quay crane to unload its cargo. Let us assume that the ship j is required to be fixed at the location starting at k, i.e. $b_j = k$, then the weight of the directed edge $\langle v_j^{\mathrm{eb}}, v_j\rangle$ is changed from c_j^e to k and the weight of the directed edge $\langle v_j, v_j^{\mathrm{et}}\rangle$ is changed from c_j^e to $L-(k+l_j)$. The graph model can be found in Fig. 1(ii).

At the end of the transformation, we have a graph which consists of some directed edges and some undirected edges. Let us pick an undirected edge $e_{ij} = (v_i, v_j)$. e_{ij} exists because ship i and ship j have a time overlap. This implies that both ship i and ship j cannot share any part of the section. If ship i is berthed at a lower wharf mark, wm (a wharf mark is a particular position in a section) than ship j, we will set the edge (v_i, v_j) to become $\langle v_i, v_j\rangle$. Similarly, if ship j is berthed at a lower wharf mark than ship 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 direction 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 is important as the direction of the edge signifies the berth locations. It is impossible to have a situation such that ship i is berthed at a lower location than ship j, and ship j is berthed at a lower location than ship k, and ship k is berthed at a lower location than ship 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 1 to n assigned to the section.

If the length of the section is bigger than the longest path in the directed acyclic graph than the section can accommodate the set of ships, else the section will not be able to accommodate the set of ships.

Lemma 1. *The optimization version of the berth planning 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 minimized.*

Proof. Obvious from the above discussion. □

Corollary 1. *The decision version of the berth planning 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.*

5. Heuristic

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:

$$\begin{aligned}\beta_{ij} &= \mathrm{LongestIn}(v_i) + \mathrm{weight}(v_i) + \mathrm{weight}(e_{ij}) \\ &\quad + \mathrm{weight}(v_j) + \mathrm{LongestOut}(v_j),\\ \beta_{ji} &= \mathrm{LongestIn}(v_j) + \mathrm{weight}(v_j) + \mathrm{weight}(e_{ij}) \\ &\quad + \mathrm{weight}(v_i) + \mathrm{LongestOut}(v_i).\end{aligned}$$

$\mathrm{LongestIn}(v_i)$ is the longest incoming path of vertex v_i. $\mathrm{LongestOut}(v_i)$ is the longest outgoing path from vertex v_i.

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 Fig. 2. 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 two topological sweeps (two passes, computing the longest incoming path for each vertex in one

```
Algorithm BerthPlanner
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
```

Fig. 2. Berth planning heuristic.

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. The geometric representation and its equivalent graph representation can be found in Fig 3 . Fig. 4 shows how our heuristic work uses the example. 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 i.

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

Lemma 2. *Algorithm berthplanner does not create a cycle from a path.*

Proof. Let us consider a cycle $C = v_1v_2\ldots v_{k-1}v_kv_1$. 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 $p_{1k} = v_1v_2\ldots v_k$. 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 the algorithm). This implies

$$\text{LongestIn}(v_k) + \text{LongestOut}(v_1) \leqslant \text{LongestIn}(v_1) + \text{LongestOut}(v_k).$$

But,

$$\text{LongestIn}(v_k) \geqslant \text{LongestIn}(v_1) + |p_{1k}| \text{ and } \text{LongestOut}(v_1) \geqslant |p_{1k}| + \text{LongestOut}(v_k).$$

This implies that

$$\text{LongestIn}(v_k) + \text{LongestOut}(v_1) \geqslant \text{LongestIn}(v_1) + \text{LongestOut}(v_k) + 2|p_{1k}|.$$

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

$$\text{LongestIn}(v_k) + \text{LongestOut}(v_1) > \text{LongestIn}(v_1) + \text{LongestOut}(v_k).$$

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

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.

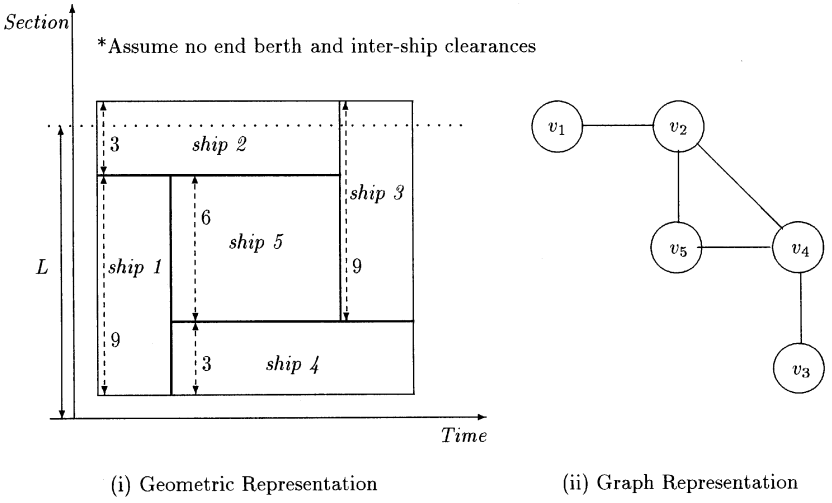
Section
*Assume no end berth and inter-ship clearances
ship 2
3
ship 3
9
ship 5
6
ship 1
9
ship 4
3
L
Time
v1
v2
v5
v4
v3
(i) Geometric Representation
(ii) Graph Representation

Fig. 3. A simple example.

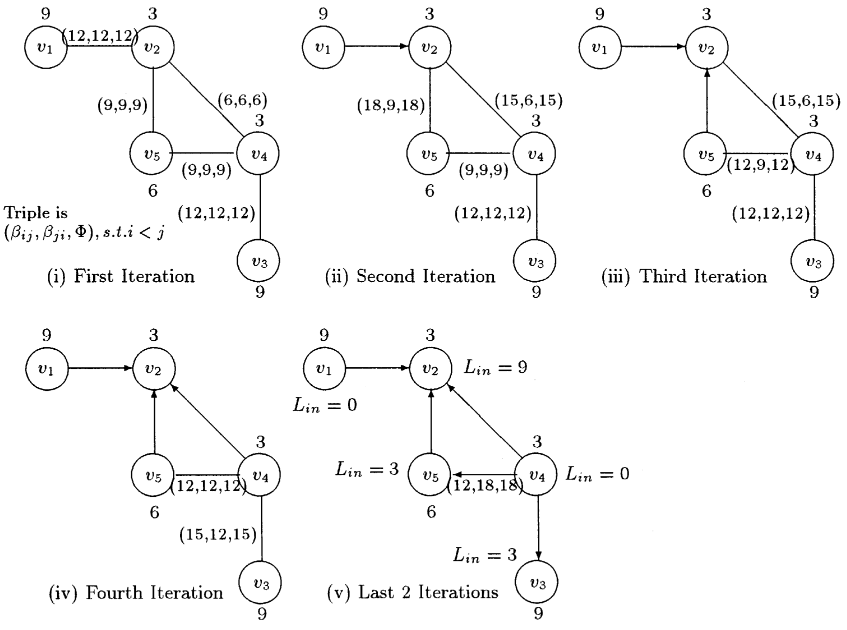

Fig. 4. Going through each iteration.

6. Experimental results

We have implemented our heuristic using the C++ programming language. We tested our heuristic using real historical data from the Port of Singapore Authority for six months. We tried to pack ships allocated to a section for each month. There are altogether 13 sections in the port. As a result, there are about 78 test sets. The size of the test sets range from 20 to 250 ships. Out of all the 78 test sets for the heuristic, our heuristic only failed to pack for two test sets. The result returned is about 5% above the section's length. In all test cases, our heuristic returned a solution in less than 5 s on a SunSparc 10 workstation.

7. Conclusions

In this paper, we studied the Berth planning problem. We formulated the problem and showed that it is NP-Complete. We transformed the problem to a restricted version of the two-dimensional packing problem and present a graph theoretic model that captured the problem succinctly. We then proposed a good heuristic that has performed well using historical test data.

References

[1] M. Garey, D. Johnson, Computers and Intractability, Freeman, 1979.

[2] Port of Singapore Authority, Annual Report 1994. PSA Corporate Communications Department, 1979.