The Yard Allocation Problem

Ping Chen and **Zhaohui Fu** and **Andrew Lim**
Department of Computer Science
National University of Singapore
3 Science Drive 2, Singapore 117543
{chenp,fuzh,alim}@comp.nus.edu.sg

Abstract

The Yard Allocation Problem (YAP) is a real-life resource allocation problem faced by the Port of Singapore Authority (PSA). We first show that YAP is NP-Hard. As the problem is NP-Hard, we propose several heuristics, including Tabu Search methods with short and long term memory, a "Squeaky Wheel" Optimization (SWO) method, and a new hybrid which combines SWO with TS to solve the problem. Extensive experiments show very favorable results for our new hybrid method.

Introduction

Singapore has one of the world's busiest ports in terms of shipping tonnage with more than one hundred thousand ship arrivals every year. One of the major logistical problems encountered is to use the minimum container yard necessary to accommodate all different requests. Each request consists of a single time interval and a series of yard space requirements during that interval. An interesting constraint applying to every request is that the length of the required space can either increase or remain unchanged as time progresses, and once yard space is allocated to a certain request, that portion of the yard space cannot be freed until the completion of the request. The major reason for such a constraint is that once a container is placed in the yard, it will not be removed until the ship for which it is bound arrives. As a result, the yard requirement can only increase. The current allocation is made manually, hence it requires a considerable amount of manpower.

Problem Definition

The objective of the Yard Allocation Problem (YAP) can be expressed in two ways:

1. Minimise the yard used to accomodate all the space requirements;
2. Maximise the number of requirements allocated on a fixed yard.

These two objectives are in fact similar to each other. We use the first one as our objective in this paper because the solution to the second problem requires the solution to the first one together with a partition routine. The formal definition of our problem can be described as follows:

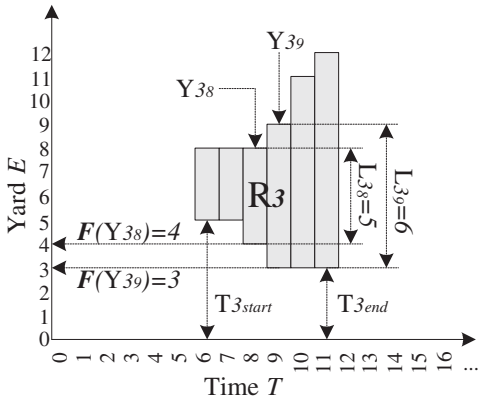

Figure 1: A *valid* request R_3.

Instance: A set R of n yard space requests and an infinite container yard E. $\forall R_i \in R, R_i$ has a series of (continuous) space requirements Y_{i_j} with length $L_{i_j}, j \in [T_{i_{start}}, T_{i_{end}}]$.
Output: A mapping function F, such that $F(Y_{i_j}) = e_k$, where $e_k \in E$ is some position on E.
Constraint: $\forall p, q \in [T_{i_{start}}, T_{i_{end}}]$ such that $p = q - 1$, $F(Y_{i_p}) \geq F(Y_{i_q})$ and $F(Y_{i_p}) + L_{i_p} \leq F(Y_{i_q}) + L_{i_q}$.
Objective: To minimize:

$$\max_{\forall R_i \in R, \forall Y_{i_j} \in R_i} (F(Y_{i_j}) + L_{i_j})$$

We use an example to illustrate the definition. Figure 1 shows a layout with only one *valid* requests R_3. The yard E is treated as an infinite straight line. Time T becomes a discrete variable with a minimum unit of 1. R_3 has six space requirements within interval $[6, 11]$ ($T_{3_{start}} = 6, T_{3_{end}} = 11$). The final position for Y_{3_8} and Y_{3_9} are $F(Y_{3_8}) = 4$ and $F(Y_{3_9}) = 3$ respectively. The corresponding output for R_3 will then be $(5, 5, 4, 3, 3, 3)$. Note all our pre-defined constraints hold as $F(Y_{3_8}) \geq F(Y_{3_9})$ and $F(Y_{3_8}) + L_{3_8} \leq F(Y_{3_9}) + L_{3_9}$. The max comes from $Y_{3_{11}}$ with the value of $F(Y_{3_{11}}) + L_{3_{11}} = 12$.

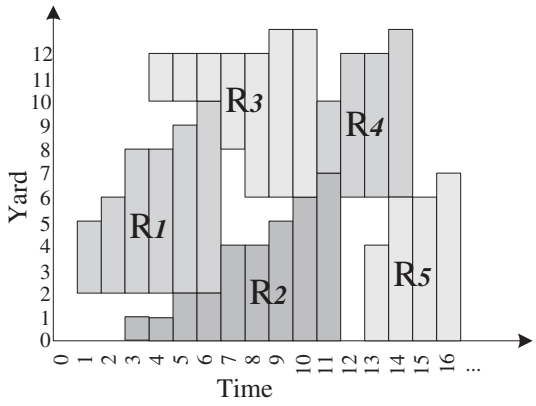

Figure 2: Five *valid* requests on yard

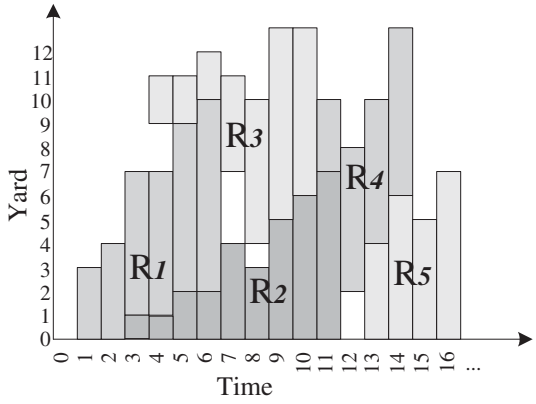

Figure 3: Five *invalid* requests on yard

We simply call each request a Stair Like Shapes (SLS) throughout this paper. Figure 2 shows five *valid* requests with the minimum yard required of 13. Though the packing in Figure 3 looks more compact, in fact, all allocations are *invalid* as the containment constraint is violated.

Theorem 1 *The Yard Allocation Problem (YAP) is NP-Hard.*

The Ship Berthing Problem (SBP) was first introduced in (Lim 1998). The SBP has a similar configuration except all the requests are of rectangular shape instead of SLS. (Lim 1999) has provided a NP-Hard proof for SBP by reducing the Set Partitioning Problem to SBP. As SBP is special case of YAP and YAP is in the class NP, YAP is NP-Hard.

Graph Transformation

Figure 2 illustrates the problem geometrically. However, the direct model may not be efficiently manipulated. We first transform the geometrical layout into a graph. Figure 4 is the corresponding graph transformation of the configuration in Figure 2. Each request R_i is represented by a vertex and there exists an edge E_{ij} connecting R_i and R_j iff R_i and

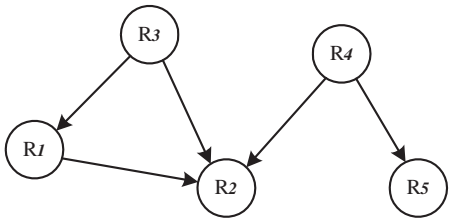

Figure 4: Graph Transformation of Figure 2

R_j have an overlap at some time. The direction of the edge determines the relative position of the two requests in the physical yard. Take Figure 2 again as an example, both R_1 and R_2 require some space at time 3,4,5 and 6, therefore in Figure 4 there is an edge between R_1 and R_2. Since R_1 is located above R_2, the direction of the edge is from R_1 to R_2. We name this edge E_{12}. Clearly, the transformed graph is a Direct Acyclic Graph (DAG). In a DAG, each vertex R_i can be assigned an Acyclic Label (AL) L_i and the edge E_{ij} implies $AL(R_i) < AL(R_j)$. Note that each $AL(R_i)(1 \leq i \leq n)$ is unique.

Lemma 1 *For each feasible layout of the yard, there exists at least one corresponding AL assignment of the vertices in the graph representation.*

A simple constructive proof can be obtained by the well-known Topological sorting algorithm. An AL assignment can also be interpreted as a permutation of $1, 2, \ldots, n$.

A "free" SLS is one with no other SLS above it, i.e. there is no obstacle blocking it from being popped out from the top of the layout. Again, use Figure 2 as an example. At the first iteration of the loop, R_3 and R_4 are the only two "free" SLSs. If we assign $AL(R_3) = 0$, in the second iteration, R_1 will become a new "free" SLS. The process continues until no more SLS are left in L.

The AL assignment only has the partial order property. Each physical layout may correspond to more than one AL assignments due to the lack of total order property. $[R_1 : 2, R_2 : 3, R_3 : 0, R_4 : 1, R_5 : 4]$ and $[R_1 : 2, R_2 : 4, R_3 : 1, R_4 : 0, R_5 : 3]$ are two possible AL assignments.

This one-to-many relationship between physical layout and AL assignments in the graph representation will incur a huge amount of confusion in heuristic searches, including Genetic Algorithms, etc. Heuristic methods tend to identify certain *good* patterns which may potentially lead to a better solution while exploring the search space. Two very different looking solutions, which may actually correspond to the same physical layout, will make it very difficult for the heuristic to identify the correct patterns.

We can avoid such confusion by normalizing the AL assignment. When there is more than one SLS to be popped out, we break the tie by selecting the SLS with the smallest label. Each un-normalized AL assignment is used to construct the corresponding DAG. Then a Topological Sort with above-mentioned tie-breaker will give the *unique* AL

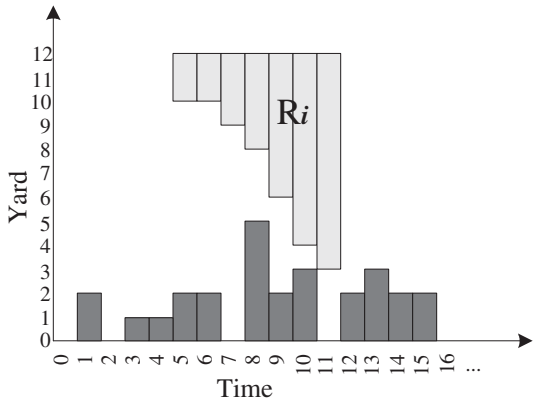

Figure 5: Before dropping: R_i is ceiling aligned

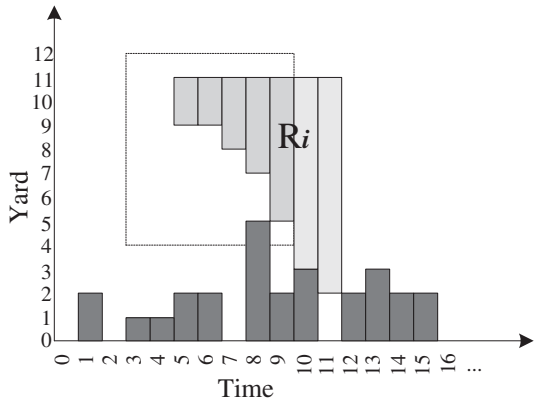

Figure 6: Each stair of R_i drop by 1. Stairs at times 10 and 11 are in their final positions. Those stairs which can drop further are in dark color surrounded by a rectangle

assignment. From this point onwards, all our solutions are represented by their normalized unique AL assignments.

Each physical layout now has a unique AL assignment. Naturally, the optimal layout has an optimal AL assignment. Our goal is to find out such an optimal AL assignment. One of the major operations, the evaluation of a given AL assignment, turns out to be non-trivial. In SBP (Lim 1998) (Fu & Lim 2000), a longest path algorithm on a DAG was used to find the minimum berth length needed. However, YAP deals with SLS, whose relative position and distance cannot be calculated in a straight-forward way, unlike rectangles. We have to use a recursive procedure to find the minimum yard needed for a given AL assignment A.

Evaluate-Solution (A)
1 **while** exists unallocated SLS
2 pick SLS S with largest AL
3 Drop($S, S_{end}, 0$)
4 **foreach** time T_i
5 **if** $T_i > L$
6 $L := T_i$
7 **return** L

Drop (S, t, l)
1 $L :=$lowest position to drop all stairs (time t')
2 **if** $L < l$
3 $L = l$
4 **forall** stair s after $t' - 1$
5 drop s to position L
6 Drop($S, t' - 1, L$)

The recursive function $Drop$ uses a greedy approach to drop a given SLS to a position as low as possible. We illustrate the details through Figures 5, 6 and 7: R_i has seven space requirements starting from time 5 to 11. R_i is first aligned to the ceiling before the process starts (Figure 5). Then from time 5 to 11, we find the maximum distance that each "stair" can drop, without exceeding a lower bound of 0. The minimum amongst all the maximum possible drop is used. In this case, the minimum distance of 1 is given at time 10 and hence every stair is shifted down by 1 (Figure 6).

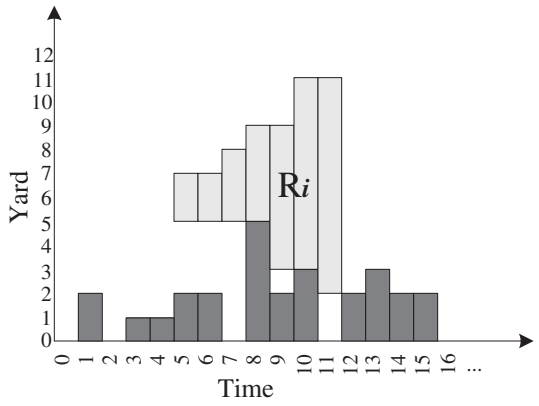

Figure 7: Final layout

Because of the initial ceiling alignment, no further shifting down is needed for all stairs at time 10 (inclusive) onwards. Note that the surface was touched at time 10.

Stairs from time 5 to 9, which are surrounded by a rectangle in Figure 6, can still be dropped further but this time with a lower bound of 3, which is the height of the previous touching surface at time 10. The dropping process completes after a few more recursions at times 8,7,6 and 5. The final layout is shown by Figure 7. Note the worst case time complexity for $Drop$ is $n \times T$, where n is the number of requests and T is the average time span for all requests.

Lemma 2 *For a given AL assignment, the greedy dropping approach always returns the layout with minimum yard used.*

Proof. The proof of the correctness of a greedy algorithm consists of two parts: First, the greedy choice always leads to an optimal solution, or any optimal solution can be transformed into a solution obtained by the greedy choice. Second, the problem has an optimal sub-structure, i.e. the

global optimum implies a local optimum. The optimal substructure property is obvious for YAP. To show the greedy choice property, we compare the solution G obtained by the greedy dropping approach with any arbitrary optimal solution O. Consider the following algorithm:

Compact (A, G, O)
1 **let** $L :=$ set of SLSs;
2 **while** L is not empty
3 pick SLS S with largest AL
4 **for** $(i = S_{begin}; i \leq S_{end}; i++)$
5 **let** $G_{s_i} :=$ position of S_i in G
6 **let** $O_{s_i} :=$ position of S_i in O
7 **if** $O_{s_i} > G_{s_i}$
8 $O_{s_i} := G_{s_i}$

The algorithm $Compact$ will transform any optimal solution into a corresponding solution that can be obtained by the greedy approach without increasing the amount of yard used. Note line 7 is based on the fact that no optimal solution can allocate S_i in a lower position than greedy approach.

Up to now, we have built a one-to-one relationship between physical layout and the AL assignment $(0, 1, \ldots, n-1)$. The problem is to find the optimal AL assignment.

Tabu Search

Tabu Search is a local search meta-heuristic that uses the best neighborhood move that is not "tabu" active to move out from a local optimum by incorporating *adaptive memory* and *responsive exploration* (Hammer 1993). According to the different usage of memory, conventionally, Tabu Search has been classified into two categories: Tabu Search with Short Term Memory (TSSTM) and Tabu Search with Long Term Memory (TSLTM) (Glover & Laguna 1997) (Sait & Youssef 1999).

TSSTM is the simpler method. Its usage of memory is via the Tabu List. Such an adaptation is also known as *recency* based Tabu Search. TSLTM uses more advanced Tabu Search techniques including intensification and diversification strategies. It archives total or partial information from all the solutions it has visited. This is also known as *frequency* based Tabu Search. It tries to identify certain potentially "good" patterns, which will be used to guide the search process towards possibly better solutions (Pham & Karaboga 2000).

Tabu Search with Short Term Memory

Our TSSTM implementation consists of two major components: a neighborhood search and a tabu list. The neighborhood solution can be obtained by swapping any two ALs in the AL assignment. For example: $[2, 3, 0, 1, 4]$ is a neighborhood solution of $[1, 3, 0, 2, 4]$ by interchanging the positions of 1 and 2. Neighbourhood solutions that are identical to the original solution after normalization are excluded for efficiency reasons.

Tabu Search with Long Term Memory

We implemented TSLTM in two phases: Diversification and Intensification. We used two kinds of diversification techniques, one used random re-starts and the other involved randomly picking a sub-sequence and inserting it in a random position. For example, $[0, 1, 2, 3, 4]$ may be changed to $[0, 3, 2, 1, 4]$ if $[2, 3]$ is chosen as the sub-sequence and its reverse (or original, if random) is inserted back in the position in front of 1. Intensification is similar to TSSTM. TSLTM uses a *frequency* based memory by recording both the *residence* frequency and the *transition* frequency of the visited solutions. In our implementation, residence frequency is taken as the number of times that the $AL(R_i) < AL(R_j), 1 \leq i, j \leq n$ in the selected solution in each iteration. The transition frequency is taken as the summation of the improvements when $AL(R_i)$ is swapped with $AL(R_j)$. The sum can be either positive or negative.

Diversification and Intensification are interleaved and during either phase, the residence frequency and transition frequency are updated according to the current selected solution. The objective function has three contributors. Besides the length of the yard space required, both the residence frequency and the transition frequency are used to evaluate the solution. Higher residence frequency indicates that an attribute is highly attractive and encourages the solution towards such direction.

"Squeaky Wheel" Optimization

"Squeaky Wheel" Optimization (SWO) is a new heuristic approach proposed in (Clements *et al.* 1997a). Until now, this concept can only be found in a few papers: (Clements *et al.* 1997b), (Joslin & Clements 1998), (Joslin & Clements 1999) and (Draper *et al.* 1999). However, we found this approach very attractive because of its fitness to our problem and very encouraging results. In 1996, a "doubleback" approach was proposed to solve the Resource Constrained Project Scheduling (RCPS) problem (Crawford 1996), which motivates the development of SWO in 1998. Our YAP is similar to the RCPS except that YAP has no precedence constraints and the tasks (requirements) are Stair Like Shapes (SLS). Instead of Left-Shift and Right-Shift in "doubleback", we only use one "drop" routine similar to Left-Shift.

The idea of SWO is also very similar to how human beings solve problems by identifying the "trouble-spot" or the "trouble-maker" and trying to resolve problems caused by them. In SWO, a greedy algorithm is used to construct a solution according to certain priorities (initially randomly generated) which is then analyzed to find the "trouble-makers", i.e. the elements whose improvements are likely to improve the objective function score. The results of the analysis are used to generate new priorities that determine the order in which the greedy algorithm constructs the next solution. This Construct/Analyze/Prioritize (C/A/P) cycle continues until a certain limit is reached or an acceptable solution is found. This is similar to the Iterative Greedy heuristic proposed in (Culberson & Luo 1996). Iterative Greedy is especially designed for Graph Coloring Problem and may not be directly applicable to other problems, whereas SWO is a more general optimization heuristic.

From another perspective, SWO can be viewed as operating on two search spaces: solutions and prioritizations.

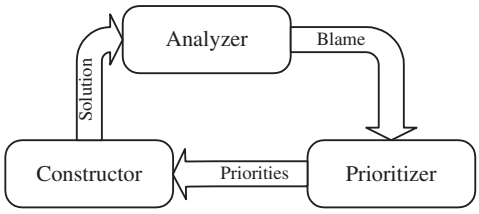

Figure 8: The CAP cycle in SWO

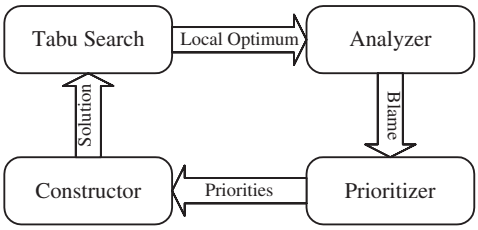

Figure 9: Modified SWO with Tabu Search

Successive solutions are only indirectly related via the re-prioritization that results from analyzing the prior solution. Similarly successive prioritizations are generated by constructing and analyzing solutions.

Figure 8 is the succinct illustration given in (Joslin & Clements 1999). Information such as solutions and priorities is passing around the C/A/P cycle. We implement the SWO system in our problem and it gives encouraging results. The system starts with a random solution, which is a random AL assignment. The normalization routine is applied before the solution is passed to the Analyzer, which evaluates the solution by applying the dropping routine. If the best known result (yard length) is B, then a threshold T is set to be $B - 1$. The blame factor for each request is the sum of the space requirements that exceed the threshold T, i.e. the total area of the SLS above the cutting line T. All the blame information is passed to the Prioritizer, which is a priority queue in our case. When the control has been handed over to the Constructor again, it continuously deletes the elements from the priority queue and immediately drops them in to the yard. A tie, i.e. more than one elements with the same priority, is broken by considering their relative positions in previous solution. This tie-breaker also helps avoid cycles in our search process. Experiments show that the normalization routine plays a very important role in SWO.

We also found that the performance of the SWO can be further improved if a "quick" Tabu Search technique, or TSSTM is embedded in the SWO. We call this modified algorithm SWO+TS, TS standing for Tabu Search. The flow chart now becomes Figure 9. The Constructor passes its solution to a TSSTM engine, which performs a quick local search and pass the local optimum to the Analyzer. Experiments shows a considerable improvement against the original SWO system. Similar ideas of SWO with "intensification" have been proposed in (Clements *et al.* 1997b), where the solution is partitioned and SWO is applied to each of the partitions.

Experimental Results

We conducted extensive experiments on randomly generated data [1]. The graph for each test case contains one connected component, in other words, the test cases cannot be partitioned into more than one independent sub-case. Due to the difficulties of finding any optimal solution in the experiments, a trivial *lower bound* is taken to be the sum of the space requirements at each time slot and used for benchmarking.

Data Set	LB	STM	LTM	SWO	SWO+TS
R126	21	28	26	25	24
R117	34	39	37	36	34
R145	39	50	45	45	40
R178	50	69	69	67	58
R188	74	105	98	98	82
R173	77	98	91	94	83
R250	83	141	119	113	94
R236	97	139	130	133	107
R213	164	245	246	246	190

Table 1: Experimental results (Entries in the table shows the minimum length of the yard required. Name of Data Set shows the number of SLSs in the file; LB:Lower Bound; STM:Tabu Search with Short Term Memory; LTM:Tabu Search Long Term Memory)

Table 1 illustrates the results. It is not surprising to see that SWO+TS outperforms all other heuristics in all test cases by a considerable margin. TSSTM gives the worst results though it has the simplest implementation. TSLTM has an obvious improvement from TSSTM, but the amount of improvement is not very stable. One of the major difficulties with Long Term Memory is the assignment of relative weights to yard length, residence frequency and transition frequency in the objective function. SWO is relatively easy to implement with comparable results to TSLTM. We believe that SWO has good *diversification* abilities and Tabu Search is good at *intensification*. Therefore combining the two methods gives the best results, which are within 10% of the trivial lower bound most of the time.

Data Set	STM	LTM	SWO	SWO+TS
R126	594	3423	1014	2719
R117	753	2521	956	2821
R145	1215	3693	1342	3375
R178	2568	5362	3474	5890
R188	2523	7822	2832	7832
R173	3432	6743	2764	7292
R250	4578	10239	3598	15632
R236	5027	11053	3486	17021
R213	4891	10476	3632	18983

Table 2: Experiment running time (seconds) for Table 1.

[1] All test data are available at the author's URL: http://www.comp.nus.edu.sg/~fuzh/YAP

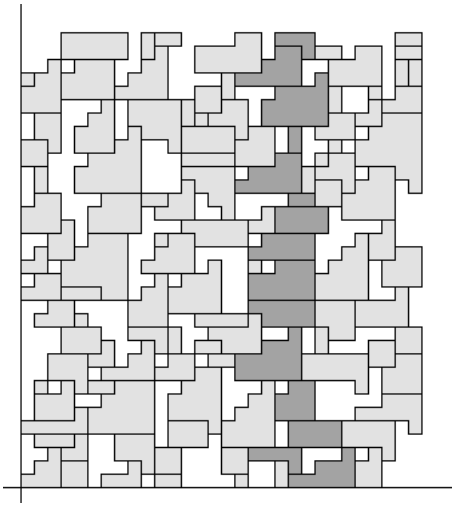

Figure 10: Physical layout of 117 SLSs (requests). Data Set: R117

Table 2 shows the running time for each of the test performed in Table 1 on a Dual-CPU (Pentium III 800MHz each) Linux machine. Although SWO and TSSTM are faster than the other two heuristics, SWO+TS is still the most cost-effective approach.

After a careful analysis of the normalization routine, it turns out that the normalization does not affect the Tabu Search very much, particularly in TSSTM. This is because TS focuses on neighborhood searches and does not pay too much attention to capturing solution patterns. An AL assignment identical to the original solution will have exactly the same value in the objective function and hence can only be identified as a non-improving move and is most unlikely to be selected. However, normalization is important to SWO by reducing the search space. When the *confusing* factors are removed, the search process becomes more stable and focused.

Figure 10 provides the solution obtained by SWO+TS for input file R117. The heavily-shaded SLSs contain the region that is the densest (lower bound). Due to the stair-like shapes, the packing layout looks sub-optimal. A closer look reveals further improvements are very unlikely.

Conclusion

In this paper, we have shown the Yard Allocation Problem (YAP) is NP-Hard by reducing the Ship Berthing Problem (SBP) to it. The geometrical representation of YAP is then transformed into a Direct Acyclic Graph (DAG) for efficient manipulation. A normalization procedure is proposed to guarantee a one-to-one relationship between geometric layout and Acyclic Label Assignment of the DAG. Finding the optimal layout is transformed into a search for the optimal Acyclic Label Assignment. Two heuristic methods are applied: first, the traditional Tabu Search, with both short and long term memory; second, the "Squeaky Wheel" Optimization (SWO) approach. The results obtained were not very attractive.

Observing that the SWO has good *diversification* abilities while the Tabu Search has good *intensification* abilities, we combined the two approaches together and named the new approach "Squeaky Wheel" Optimization with Tabu Search (SWO+TS). Extensive experiments showed that our new approach, SWO+TS, outperformed both original Tabu Search and SWO by a margin of 10%.

References

Clements, D.; Crawford, J.; Joslin, D.; Nemhauser, G.; Puttlitz, M.; and Savelsbergh, M. 1997a. Heuristic optimization: A hybrid AI/OR approach. In *Workshop on Industrial Constraint-Directed Scheduling*.

Clements, D.; Crawford, J.; Joslin, D.; Nemhauser, G.; Puttlitz, M.; and Savelsbergh, M. 1997b. Heuristic optimization: A hybrid ai/or approach. In *Proceedings of the Workshop on Industrial Constraint-Directed Scheduling. In conjunction with the Third International Conference on Principles and Practice of Constraint Programming (CP97)*.

Crawford, J. M. 1996. An approach to resource constrained project scheduling. In *Proceedings of the 1996 Artificial Intelligence and Manufacturing Research Planning Workshop*.

Culberson, J. C., and Luo, F. 1996. Exploring the k-colorable landscape with iterated greedy. *Johnson &Trick* 245–284.

Draper, D.; Jonsson, A.; Clements, D.; and Joslin, D. 1999. Cyclic scheduling. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*.

Fu, Z., and Lim, A. 2000. A hybrid method for the ship berthing problem. In *Proceedings of the Sixth Artificial Intelligence and Soft Computing*.

Glover, F., and Laguna, M. 1997. *Tabu Search*. Kluwer Academic Publishers.

Hammer, P. L. 1993. *Tabu Search*. Basel, Switzerland: J.C. Baltzer.

Joslin, D. E., and Clements, D. P. 1998. Squeaky wheel optimization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98), Madison, WI*, 340–346.

Joslin, D. E., and Clements, D. P. 1999. "squeaky wheel" optimization. *Journal of Artificial Intelligence Research* 10:353–373.

Lim, A. 1998. On the ship berthing problem. *Operations Research Letters* 22(2-3):105–110.

Lim, A. 1999. An effective ship berthing algorithm. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 594–599.

Pham, D., and Karaboga, D. 2000. *Intelligent optimisation techniques: genetic algorithms, tabu search, simulated annealing and neural networks*. London; New York: Springer.

Sait, S., and Youssef, H. 1999. *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems*. IEEE.