A Critical-Shaking Neighborhood Search for the Yard Allocation Problem
(extended abstract)

Andrew Lim and Zhou Xu

1 Introduction

Singapore is one of the world’s busiest ports having more than one hundred thousand ship arrivals every year. Competing pressures for limited land use and competition from other regional and international ports impel port planner to use as minimum space of the container yard as possible to reconcile all different requests. The current yard allocation is made manually. To save the manpower, an automated way needs to be designed.

The yard allocation problem (YAP) was first introduced and defined in [3], by extending the berth allocation problem [5] and other port operations studies [1]. We adopt their definition of the YAP as follows. Let E denote an infinite container yard. Give a set R of n yard space requests. Every request Ri ∈ R spans from time TSi to time TEi and has a series of continuous space requirements Yi,t, each of which has a length Li,t for every time period t, where t ∈ Ti and Ti = {TSi, TSi + 1, ..., TEi}. To allocate spaces to requests, an allocation mapping F needs to be chosen to assign a starting position F(Yi,t) ∈ E to every Yi,t, where i ∈ R and t ∈ Ti.

Any space cannot be allocated to different requests during the same time. Furthermore, because spaces allocated to a request will keep occupied until the request ends, the mapping F must satisfy that for every request Ri ∈ R and t ∈ Ti, but t > TSi,

\[ F(Y_i,t) \leq F(Y_{i,t-1}) \quad \text{and} \quad F(Y_i,t) + L_{i,t} \geq F(Y_{i,t-1}) + L_{i,t-1}. \]  

The objective is then to minimize:

\[ \max_{i \in R, t \in T_i} (F(Y_i,t) + L_{i,t}), \]  

that is the space needed. For example, a valid scheduling is shown in Figure 1 cited from [3].

The YAP was proved to be NP-hard but could be transformed to the following two-stage decision [3]. In the first stage, a unique priority \( \sigma(i) \) will be determined for each request \( R_i \), where \( \sigma(i) \in \{1, ..., n\} \) and \( \sigma(i) \neq \sigma(j) \) for any \( j \neq i \). The bigger the \( \sigma(i) \) is, the higher the priority of \( R_i \) is. Then in the second stage, the optimum allocation mapping \( F \) can be obtained for the given \( \sigma \) efficiently, to minimize the space needed through the following greedy method. It allocates requests from that with the highest priority to that with the lowest. Let \( H_i,t \) denote the maximum occupied place in time \( t \in N_i \), after all requests with higher priorities than \( R_i \) have been allocated. Through a recursive algorithm proposed in [3], we determine \( F(Y_i,t) \) for request \( R_i \) to be the minimum possible place that is larger than \( H_i,t \) and that satisfies (1). Its time complexity is \( O(\Delta_T) \) where \( \Delta_T = \max_{\sigma R_i \in R}(TE_i - TS_i) \). [3] proves such an allocation mapping minimizes the length of yard for the particular \( \sigma \) given. Therefore, every \( \sigma \) of priorities for the \( n \) requests represents a feasible solution to the YAP problem. Let \( f(\sigma) \) denote the minimum yard space to accommodate requests based on \( \sigma \). Since \( f(\sigma) \) can be evaluated efficiently in the above way, to solve YAP, we only need to find the best \( \sigma \) of the request priorities so that \( f(\sigma) \) is minimized.

For instance shown in Figure 1, the priorities of the five requests from \( R_1 \) to \( R_5 \) can be \( \sigma(R_1) = 3, \sigma(R_2) = 5, \sigma(R_3) = 2, \sigma(R_4) = 1, \sigma(R_5) = 4 \) respectively. According to the priorities, the \( R_2 \) and the \( R_5 \) are firstly arranged. Afterwards, \( R_1 \) are placed just above \( R_2 \), while \( R_3 \) and \( R_4 \) are allocated above \( R_1, R_2, R_3 \) lastly. This schedule leads \( f(\sigma) \) to be 12.

In literature, several meta-heuristics has been attempted to seek near-optimum solution to the YAP, including simulated annealing (SA), tabu search (TS), squeaky-wheel optimization (SWO) [3], and genetic algorithm (GA) [2]. All of them are limited successful for their inefficiency either in solution quality or time performance.

This paper proposes a more effective heuristic procedure, named critical-shaking neighborhood search (CSNS), to solve the problem.

\[ \text{Figure 1. Five valid requests on yard, cited from [3]} \]
Algorithm 1: A General Framework of the CSNS

1: Generate an initial $s_0$ for the $n$ requests randomly;
2: Let $\sigma \leftarrow s_0$ and $\sigma_{opt} \leftarrow s_0$ and $L \leftarrow L_{max}$;
3: while $L > 0$ do
4:  Analyze the current $\sigma$ to pick up a set $S$ of $k$ critical requests from it;
5:  Shake priorities of requests in $S$ to generate a new $\sigma'$;
6:  Call a local neighborhood search to improve $\sigma'$ to $\sigma''$;
7:  if $\sigma''$ is better than $\sigma_{opt}$ then
8:    $\sigma_{opt} \leftarrow \sigma''$ and $L \leftarrow L_{max}$ and $\sigma \leftarrow \sigma''$;
9:  else
10:    if $\sigma''$ is not worse than $\sigma$ then
11:      $\sigma \leftarrow \sigma''$;
12:    end if
13:  $L \leftarrow L - 1$;
14: end if
15: end while

As shown in Algorithm 1, the critical-shaking neighborhood search begins with an initial $s_0$ of priorities generated randomly. Afterwards, an iterative improvement is made as follows. Let $\sigma$ denote the current priorities. First, we analyze the structure of $\sigma$ to pick a set $S$ of $k$ critical requests. Changing the priorities of those critical requests may lead to pick the solution in a high probability. Because it is difficult to determine an optimum value of the priorities for the $k$ critical requests, we shake their priorities randomly to generate a new $\sigma'$ of priorities. The idea of picking and shaking is similar to that of squeeze-wheel optimization (SWO) which was first introduced by Joslin and Clements (see [4]) and applied to solve the YAP by [3]. However, the SWO in [3] constructs a new solution by always moving the critical request to the front, instead of shaking randomly as we do. Experiments show that the random shaking is effective to lead a fast convergence to solutions with high qualities. Moreover, since the SWO in [3] picks requests that occupies extra space against an expect limit, there is always a bias to a few requests. To overcome this bias, we adopt a different policy to pick critical requests in the following way.

Firstly, let us define the score function $c(Y_{i,t})$ for each space requirement of each request $R_t$ according to a given $\sigma$. Based on that, the score function for request $R_i$ is $c(R_i) = \sum_{t \in T_{c}} c(Y_{i,t})$. Both $c(R_i)$ and $c(Y_{i,t})$ can be computed recursively as follows.

$$
c(Y_{i,t}) = \begin{cases} 
1 & \text{if } F(Y_{i,t}) + L_{i,t} = B, \\
c(R_j)/n(R_j) & \text{if } j \text{ and } i \text{ is consecutive}, \\
0 & \text{otherwise}.
\end{cases}
$$

$$
c(R_i) = \sum_{t \in N_{i}} c(Y_{i,t}).
$$

Hence, the higher the $c(R_i)$ is, the more critical the request $R_i$ is considered to be.

According to the score function $c(R_i)$ of every request $R_i$, a set $S$ of $k$ critical requests is picked in the following way. Suppose $c_{max}$ is the maximum score among $c(R_i)$ for all $R_i \in R$. We randomly choose $k$ requests from those with scores higher than $\rho c_{max}$ to construct $S$.

After shaking priorities of the picked critical requests, we obtain a new solution $\sigma'$. To improve $\sigma'$ further, a local search will be called. Various neighborhoods can be defined, including the 1-swap and the 1-insert neighborhoods. To enhance the efficiency, the size of neighborhoods is reduced by introducing particular constraints that will have little side-effect on the solution quality but lessen the time consuming a lot.

Suppose $\sigma''$ is the improved solution from $\sigma'$ after the local search. We update the current best solution $\sigma_{opt}$ by $\sigma''$ if $\sigma''$ is better than $\sigma_{opt}$. And move the current solution $\sigma$ to $\sigma''$ if the $\sigma''$ is not worse than $\sigma$. The iterations will continue until no improvement of $\sigma_{opt}$ has been appeared for $L_{max}$ times. To reduce the number of parameters, we set the $L_{max}$ times to be 1000 in this paper.

### 3 Experimental Results

Extensive experiments are conducted on a Pentium IV 2.40GHZ personal computer with program coded in C++. For testing, we adopt the nine benchmark instances, which were randomly generated and shared in http://www.comp.nus.edu.sg/fuzh/YAP by [3].

<table>
<thead>
<tr>
<th>INST</th>
<th>LB</th>
<th>BEST</th>
<th>GA</th>
<th>CSNS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OBJ</td>
<td>CPU</td>
<td>OBJ</td>
<td>CPU</td>
</tr>
<tr>
<td>R126</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>100.61</td>
</tr>
<tr>
<td>R117</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>147.33</td>
</tr>
<tr>
<td>R145</td>
<td>39</td>
<td>39</td>
<td>39</td>
<td>261.33</td>
</tr>
<tr>
<td>R178</td>
<td>50</td>
<td>53</td>
<td>55</td>
<td>341.00</td>
</tr>
<tr>
<td>R188</td>
<td>74</td>
<td>77</td>
<td>79</td>
<td>440.33</td>
</tr>
<tr>
<td>R173</td>
<td>77</td>
<td>77</td>
<td>79</td>
<td>558.33</td>
</tr>
<tr>
<td>R250</td>
<td>83</td>
<td>85</td>
<td>89</td>
<td>1210.67</td>
</tr>
<tr>
<td>R236</td>
<td>97</td>
<td>98</td>
<td>101</td>
<td>817.67</td>
</tr>
<tr>
<td>R213</td>
<td>164</td>
<td>177</td>
<td>187</td>
<td>1440.67</td>
</tr>
</tbody>
</table>

**AVG**

1 indicates the length of the yard used;

2 indicates the time (in seconds) to achieve the solution of OBJ;

Table 1 summarizes the experimental results. It shows that the CSNS over-performs the genetic algorithm (GA), which is the best approach in literature, in terms of both higher solution qualities and less time consumed.

### REFERENCES


