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

Andrew Lim and Zhou Xu[1][2]

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 $R_i \in R$ spans from time TS_i to time TE_i and has a series of continuous space requirements $Y_{i,t}$, each of which has a length $L_{i,t}$ for every time period t, where $t \in T_i$ and $T_i = \{TS_i, TS_i + 1, ..., TE_i\}$. To allocate spaces to requests, an allocation mapping F needs to be chosen to assign a starting position $F(Y_{i,t}) \in E$ to every $Y_{i,t}$, where $i \in R$ and $t \in T_i$. Any space can not 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 $R_i \in R$ and $t \in T_i$ but $t > TS_i$,

$$F(Y_{i,t}) \le F(Y_{i,t-1}) \text{ and } F(Y_{i,t}) + L_{i,t} \ge F(Y_{i,t-1}) + L_{i,t-1}. \quad (1)$$

The objective is then to minimize:

$$\max_{i \in R, t \in T_i} (F(Y_{i,t}) + L_{i,t}), \quad (2)$$

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 $R_i \in R$, where $\sigma(i) \in \{1, ..., n\}$ and $\sigma(i) \ne \sigma(j)$ for any $j \ne i$. 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 σ 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 $\Delta_T = \max_{\forall R_i \in R}(TE_i - TS_i)$. [3] proves such an allocation mapping minimizes the length of yard for the particular σ given. Therefore, every σ 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 σ. Since $f(\sigma)$ can be evaluated efficiently in the above way, to solve YAP, we only need to find the best σ of the request priorities so that $f(\sigma)$ is minimized.

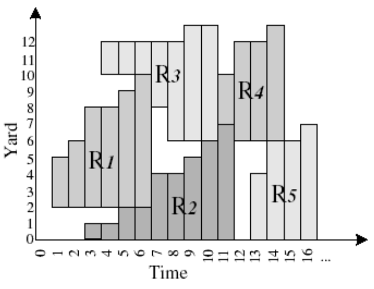

Figure 1. Five valid requests on yard, cited from [3]

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_5 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 simulate 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.

[1] Dept Industrial Engineering and Engineering Management, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong
[2] Corresponding Author: Zhou Xu; Email: xuzhou@ust.hk

2 General Framework

Algorithm 1 A General Framework of the CSNS

```
1: Generate an initial $\sigma_0$ for the $n$ requests randomly;
2: Let $\sigma \leftarrow \sigma_0$ and $\sigma_{opt} \leftarrow \sigma_0$ 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 σ_0 of priorities generated randomly. Afterwards, an iterative improvement is made as follows. Let σ denote the current priorities. First, we analyze the structure of σ to pick a set S of k critical requests. Changing the priorities of those critical requests may lead an improvement of 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 σ' 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_i according to a given σ. Based on that, the score function for request R_i is $c(R_i) = \sum_{t\in T_i} c(Y_{i,t})$. Both $c(R_i)$ and $c(Y_{i,t})$ can be computed recursively as follows.

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

$$c(R_i) = \sum_{t\in N_i} c(Y_{i,t}). \tag{3}$$

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 ρc_{max} to construct S.

After shaking priorities of the picked critical requests, we obtain a new solution σ'. To improve σ' 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 σ'' is the improved solution from σ' after the local search. We update the current best solution σ_{opt} by σ'' if σ'' is better than σ_{opt}. And move the current solution σ to σ'' if the σ'' is not worse than σ. The iterations will continue until no improvement of σ_{opt} has been appeared for L_{max} times. To reduce the number of parameters, we set the L_{max} 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 1. Experimental Results

INST	LB	BEST	GA		CSNS	
			OBJ[1]	CPU[2]	OBJ[1]	CPU[2]
R126	21	22	24	100.67	**22**	2.2
R117	34	34	**34**	147.33	**34**	0.5
R145	39	39	**39**	261.33	**39**	1.56
R178	50	53	55	341.00	**53**	104.5
R188	74	77	79	440.33	**77**	153.44
R173	77	77	79	558.33	**77**	14.77
R250	83	85	89	1210.67	**85**	60.77
R236	97	98	101	817.67	**98**	405.2
R213	164	177	187	1440.67	**177**	310.83
AVG	71.00	73.56	76.33	590.89	73.56	117.09

[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

[1] E.K. Bish, T.Y. Leong, C.L. Li, J.W.C. NG, and D. Simichi-Levi. Analysis of a new vehicle scheduling and location problem. *Naval Research Logistics*, 48:363–385, 2001.

[2] Ping Chen, Zhaohui Fu, and Andrew Lim. Using genetic algorithms to solve the yard allocation problem. In *Proceedings of Genetic and Evolutionary Computation Conference 2002, New York City, USA.*

[3] Ping Chen, Zhaohui Fu, and Andrew Lim. The yard allocation problem. In *Proceedings of the Eighteenth American Association for AI National Conference (AAAI), 2002, Edmonton, Alberta, Canada.*

[4] D. Joslin and D. Clements. Squeaky wheel optimization. In *Proceedings of the Fifteenth American Association for AI National Conference (AAAI), 1998, Madison, WI.*

[5] A. Lim. On the ship berthing problem. *Operation Research Letters*, 22:105–110, 1998.