The General Yard Allocation Problem

Ping Chen[1], Zhaohui Fu[1], Andrew Lim[2], and Brian Rodrigues[3]

[1] Dept of Computer Science, National University of Singapore
3 Science Drive 2, Singapore 117543
{chenp,fuzh}@comp.nus.edu.sg

[2] Dept of IEEM, Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
iealim@ust.hk

[3] School of Business, Singapore Management University
469 Bukit Timah Road, Singapore 259759
br@smu.edu.sg

Abstract. The General Yard Allocation Problem (GYAP) is a resource allocation problem faced by the Port of Singapore Authority. Here, space allocation for cargo is minimized for all incoming requests for space required in the yard within time intervals. The GYAP is NP-hard for which we propose several heuristic algorithms, including Tabu Search, Simulated Annealing, Genetic Algorithms and the recently emerged "Squeaky Wheel" Optimization (SWO). Extensive experiments give solutions to the problem while comparisons among approaches developed show that the Genetic Algorithm method gives best results.

1 Introduction

The Port of Singapore Authority (PSA) operates this world's largest integrated container port and transshipment hub in Singapore. In year 2000, PSA handled 19.77 million Twenty-foot Equivalent Units (TEUs) of containers worldwide, including 17.04 million TEUs in Singapore. This represents 7.4% of the global container throughput and 25% of the world's total container transshipment throughput.

Typically, storage is a constraining component in port logistics management. Factors that impact on terminal storage capacity include stacking heights, available net storage area, storage density (containers per acre) and dwell times for empty containers and breakbulk cargo. The port in Singapore, faces such storage constraints in heightened way due to scarcity of land available for port activities. As such, the optimization of storage of cargo in its available yards is crucial to its operations. In studying its operations with the view of finding better ways to utilize storage space within the dynamic environment of the port, we have focused on a central allocation process in storage operations which allows for improved usage of space. In this process, requests are made from operations which coordinate ship berthing and ship-to-apron loading as well as apron-to-yard transportation. Each request consists of a set of yard spaces required for a single time interval. If space is allocated to the request, this space cannot be freed until completion of the request, i.e, until the end time. 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 to reduce labor cost. The purpose is to pack all such requirements into a minimum space. The current allocation is made manually, hence it requires a considerable amount of manpower and the yard arrangement generated is not efficient.

Sabria and Daganzo [1] give a bibliography on port operations with the focus on berthing and cargo-handling systems. On the other hand, traffic and vehicle-flow scheduling on landside upto yard points have been studied well (see for example, Bish et al.[2]). Other than studies such as Gambardella et al. [3], which address spatial allocation of containers on a terminal yard using simulation techniques, there has been little direct study on yard space allocation as described in this paper. We propose a basic model to address this port storage optimization problem. The model is applicable elsewhere as it is generic in form. We call it General Yard Allocation Problem (GYAP), which is an extended study of The Yard Allocation Problem (YAP) (see [4] [5]). The yard is treated as a two dimensional space instead of one dimensional in YAP.

This paper is organised as follows: problem definition is given in Sect. 2, followed by a discussion on two-dimensional packing, which is highly related to our problem, in Sect. 3. Different heuristic approaches including Tabu Search, "Squeaky Wheel" Optimization, Simulated Annealing and Genetic Algorithms, as discussed in the next four sections, are applied. Before we conclude in Sect. 9, experimental results are presented and analyzed in Sect. 8.

2 Problem Description and Formulation

The main objective of the GYAP problem is to minimize the container yard space used while satisfying all the space request. The GYAP problem can be described as follows:

A set R of n yard space requests, as described above, and a two-dimensional infinite container yard E are given. We can think of E as the being first quadrant in $\Re^2$. Each request $R_i \in R$ $(i = 1, ..., n)$ has a series of (continuous) space requirements Y_{i_j} with length L_{i_j} and width W_{i_j}, where $j \in [T_{i_{start}}, T_{i_{end}}]$, where the latter time interval is defined by the request R_i.
A mapping, F, such that $F(Y_{i_j}) = (x, y)$, where $(x, y) \in E$ gives the coordinate of the bottom left corner of Y_{i_j} as it is aligned in E with its sides parallel to the X-Y axes. Each map, F, must also satisfy the condition that for all $p, q \in [T_{i_{start}}, T_{i_{end}}]$ such that $p = q - 1$, and for $F(Y_{i_p}) = (x_{i_p}, y_{i_p})$ and $F(Y_{i_q}) = (x_{i_q}, y_{i_q})$, we must have $x_{i_p} \geq x_{i_q}, y_{i_p} \geq y_{i_q}, x_{i_p} + L_{i_p} \leq x_{i_q} + L_{i_a}$ and $y_{i_p} + W_{i_p} \leq y_{i_q} + W_{i_q}$.This constraint provides the fact that the total space requests increase in time, as would be expected in a realistic situation. Our objective is then to minimize, over all possible mappings F, :

$$\max_{i,j,Y_{i_j} \in R_i} [(Proj_X F(Y_{i_j}) + L_{i_j})] \times \max_{i,j,Y_{i_j} \in R_i} [(Proj_Y F(Y_{i_j}) + W_{i_j})]$$

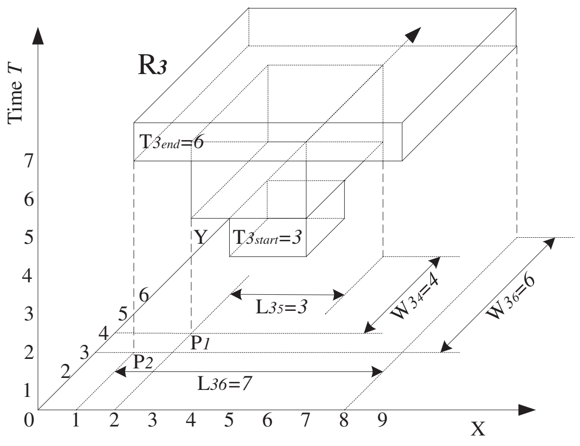

Fig. 1. A *valid* request R_3 in GYAP. The coordinates for P_1 and P_2 are $(2, 4)$ and $(1, 3)$ respectively. Same amount of space is required at time 4 and 5.

where $\mathrm{Proj}_X(\mathrm{Proj}_Y)$ denotes the orthogonal projections from $\Re^2$ onto the X (Y) axis. In other words, we would like to minimize the total area used in the yard while satisfying all the space requests.

Figure 1 shows a layout with only one *valid* request, R_3. Time T is taken as a discrete variable with a minimum unit of 1. R_3 has four space requirements, at times 3, 4, 5, and 6, two of which are the same at time 4 and 5, within the time interval $[T3_{start} = 3, T3_{end} = 6]$. The final positions for $Y3_5$ and $Y3_6$ are $F(Y3_5) = (2, 4)$ and $F(Y3_6) = (1, 3)$, respectively, as shown. The corresponding mappings for R_3 will then be $\{(3, 4), (2, 4), (2, 4), (1, 3)\}$, where all the constraints imposed can be seen to hold. The maximum value in this case, which is the product of the maximum X−coordinate and the maximum Y−coordinate, is 72. Note that the space requests is look like a inverted pyramid when together. We will call the each space requirement at each time slot a *layer.*

It is clear that GYAP is NP-hard, since (see [4]) we have found YAP is NP-hard, which is a special case of GYAP in the case when each request has the same length (width) which equals to the length(width) of the yard.

3 Two-Dimensional Rectangular Packing Problem

In finding solutions for the GYAP, we confront, at each time slot, a Two-Dimensional Rectangular Packing Problem (2-DRPP) with certain boundary constraints.

The 2-DRPP has been proven to be NP-hard. Various heuristic methods have been proposed. However, most of these are *meta-heuristics* because the complex representation of the 2DRPP makes it difficult to apply other heuristics. Such meta-heuristics usually use a lower-level packing algorithm, for example, greedy packing, and higher-level algorithms, such as Tabu Search.

A lower-level greedy packing routine allocates the objects according to certain given ordering, which is determined by the higher-level heuristics. A greedy packing routine is the Bottom Left (BL) heuristic [6] [7] [8]. Starting from the top right corner each item is slid as far as possible to the bottom and then as far as possible to the left of the object. These successive vertical and horizontal operations are repeated until the item locks in a stable position.

The major advantage of this routine is its simplicity. Its time complexity is only $O(N^2)$, where N is the total number of items to be packed. Due to its low complexity this simple heuristic is favorable in a hybrid combination with a meta-heuristic, since the decoding routine has to be executed every time the quality of a solution is evaluated and hence contributes significantly to the running time of any hybrid algorithm [8].

The BL packing routine uses a deterministic algorithm, hence the input ordering of the objects fully determines the final layout. Heuristics can be applied to optimize the ordering. In fact, such orderings can also be considered as permutations, and hence, in this work, we use the word *permutation* for a solution representation.

As the objective of GYAP is to minimize the total yard space while satisfying all the space requests, A procedure, called *EVALUATE_SOLUTION*, is required. It takes a solution (i.e permutation) and computes the minimum space required by applying BL heuristic on packing those pyramids into $\Re^3_+$ (with two dimensions representing the yard and one dimension for the time axis).

```
EVALUATE_SOLUTION (P)
1 for each $R \in P$
2   while the position of $R$ changes
3     $SHIFT(R, R_{end}, 0, 0, bottom)$
4     $SHIFT(R, R_{end}, 0, 0, left)$
5 return Minimum area used
```

```
SHIFT ($R, t, l, b, o$)
01 $B$ :=bottom most position to shift all layers (time $t'$)
02 if $B < b$
03   $B = b$
04 $L$ :=left most position to shift all layers (time $t''$)
05 if $L < l$
06   $L = l$
07 if $o = bottom$
08   for all layers $r$ after $t' - 1$
09     shift $r$ bottomwards to $B$
10     $SHIFT(R, t' - 1, L, B, o)$
11 else
12   for all layer $r$ after $t'' - 1$
13     shift $r$ leftwards to $L$
14     $SHIFT(R, t'' - 1, L, B, o)$
```

4 Tabu Search

Tabu Search (TS) is a local search meta-heuristic that uses the best neighborhood move that is not "tabu" active to move out from local optimum by incorporating *adaptive memory* and *responsive exploration* [9]. 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) [10] [11].

4.1 Tabu Search with Short Term Memory

The usage of memory of TSSTM is via the Tabu List. Such an adaptation is also known as *recency*-based Tabu Search. The neighborhood solution can be obtained by swapping any two numbers in the permutation. 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. Neighborhood solutions that are identical to the original solution after normalization are excluded for efficiency reasons.

4.2 Tabu Search with Long Term Memory

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 attempts to identify certain potentially "good" patterns, which will be used to guide the search process towards possibly better solutions [12]. Two kinds of diversification techniques are used. One is random re-start. The other is randomly picking a sub-sequence and inserting it to 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 inverted order (or original, if random) is inserted back in the position in ahead of 1. Intensification is similar to TSSTM. TSLTM uses a *frequency*-based memory by recording both *residence*-frequency and *transition*-frequency of the visited solutions.

5 "Squeaky Wheel" Optimization

"Squeaky Wheel" Optimization (SWO) is a new heuristic approach proposed in [13]. Until now, this concept can only be found in a few papers: [14], [15] and [16]. In 1996, a "doubleback" approach was proposed to solve the Resource Constrained Project Scheduling (RCPS) problem [17] , which motivated the development of SWO in 1998. The YAP is similar to the RCPS except that it 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 a "drop" routine similar to Left-Shift. We continue using this technique in GYAP.

The ideas in SWO mimic how human beings solve problems by identifying the "trouble-spot" or the "trouble-maker" and trying to resolve problems caused by the latter. 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 [18]. 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.

In our problem, we start with a random solution, which is a random permutation. The Analyzer evaluates the solution by applying the packing routine. If the best known result (yard space) 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 this threshold T, i.e. the total area of the pyramid above the cutting line T. All 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 into the yard. A tie, i.e. more than one element 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.

We also found that the performance of the SWO can be further improved if a "quick" Tabu Search technique TSSTM is embedded in the SWO. We call this modified algorithm SWO+TS, where TS denotes Tabu Search. The Constructor passes its solution to a TSSTM engine, which performs a quick local search and passes 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 [14], where solutions are partitioned and SWO is applied to each partition.

6 Simulated Annealing

Simulated annealing [19] stochastically simulates the slow cooling of a physical system. We used the following Simulated Annealing algorithm on our problem:

Step 1. Choose some initial temperature T_0 and a random initial starting configuration θ_0. Set $T = T_0$. Define the Objective function (Energy function) to be $En()$ and the cooling schedule σ.

Step 2. Propose a new configuration, θ', of the parameter space, within a neighborhood of the current state θ, by setting $\theta' = \theta + \phi$ for some random vector ϕ.

Step 3. Let $\delta = En(\theta') - En(\theta)$. Accept the move to θ' with probability

$$\alpha(\theta, \theta') = \begin{cases} 1 & if \delta < 0 \\ exp(-\frac{\delta}{T}) & otherwise \end{cases}$$

Step 4. Repeat Step 2 and 3 for K iterations, until it is deemed to have reached the equilibrium.
Step 5. Lower the temperature by $T = T \times \sigma$ and repeat Steps 2-4 until certain stopping criterion, for our case $T < \epsilon$ (for some small ϵ) is met.

Due to the logarithmic decrement of T, we set $T_0 = 1000$. The Energy function is simply defined as the length of the yard required. The probability $exp(-\frac{\delta}{T})$ is a Boltzmann factor. The number of iterations K is proportional to the input size n. A neighborhood is defined similarly as the one in TS by swapping of any two permutations and re-positioning a random permutation.

7 Genetic Algorithms

Genetic Algorithms [20] (GA) are search procedures based notions from natural selection.It is clear that the classical binary representation is not a suitable in GYAP, in which a permutation $(0, 1, \ldots, n-1)$ is used as the solution representation. The solution space is a permutation of $(0, 1, \ldots, n-1)$. The binary codes of these permutations do not provide any advantage. At times, the situation is even worse: the change of a single bit may not lead to a valid solution. Here, we adopt a vector representation, i.e. by using a permutation directly as the chromosome in the genetic process. We will illustrate the two major genetic operators used in our approach, crossover and mutation.

7.1 Crossover Operator

Using permutations as chromosome, we have implemented three crossover operators: *Classical crossover with repair*, *Partially-mapped crossover* and *Cycle crossover*. All these operators are be tailored to suit our problem domain. A small change in the crossover operator may cause totally different results.

Classical Crossover with Repair. The Classical Crossover operator builds the offspring by appending the head from one parent with the tail from the other parent, where the head and tail come from a random cut of the parents' chromosomes. A repair procedure may be necessary after the crossover [21]. For example, the two parents (with random cut point marked by '|'):

$$p_1 = (0\ 1\ 2\ 3\ 4\ 5 \mid 6\ 7\ 8\ 9) \ \ and \ \ p_2 = (3\ 1\ 2\ 5\ 7\ 4 \mid 0\ 9\ 6\ 8)$$

will produce the following two offsprings:

$$o_1 = (0\ 1\ 2\ 3\ 4\ 5 \mid 0\ 9\ 6\ 8) \ \ and \ \ o_2 = (3\ 1\ 2\ 5\ 7\ 4 \mid 6\ 7\ 8\ 9)$$

However, the two offsprings are not valid permutations after the crossover. A repair routine replaces the repeated numbers with the missing ones randomly. The repaired offsprings will be:

$$o_1 = (7\ 1\ 2\ 3\ 4\ 5 \mid 0\ 9\ 6\ 8) \ \ and \ \ o_2 = (3\ 1\ 2\ 5\ 7\ 4 \mid 6\ 0\ 8\ 9)$$

The classical crossover operator tries to maintain the absolute positions in the parents.

Partially Mapped Crossover. Partially Mapped Crossover (PMX) was first used in [22] to solve the Traveling Salesman Problem. We have made several adjustments to accommodate our permutation representation. The modified PMX builds an offspring by choosing a subsequence of a permutation from one parent and preserving the order and position of as many numbers as possible from the other parent. The subsequence is determined by choosing two random cut points. For example, the two parents:

$$p_1 = (0\ 1\ 2 \mid 3\ 4\ 5\ 6 \mid 7\ 8\ 9) \ \ \textit{and} \ \ p_2 = (3\ 1\ 2 \mid 5\ 7\ 4\ 0 \mid 9\ 6\ 8)$$

would produce offspring as follows. First, two segments between cutting points are swapped (symbol 'u' represents 'unknown' for this moment):

$$o_1 = (u\ u\ u \mid 5\ 7\ 4\ 0 \mid u\ u\ u) \ \ \textit{and} \ \ o_2 = (u\ u\ u \mid 3\ 4\ 5\ 6 \mid u\ u\ u)$$

The swap defines a series of mappings implicitly at the same time:

$$3 \leftrightarrow 5, 4 \leftrightarrow 7, 5 \leftrightarrow 4 \ \textit{and} \ 6 \leftrightarrow 0.$$

The 'unknown's are then filled in with numbers from original parents, for which there is no conflict:

$$o_1 = (u\ 1\ 2 \mid 5\ 7\ 4\ 0 \mid u\ 8\ 9) \ \ \textit{and} \ \ o_2 = (u\ 1\ 2 \mid 3\ 4\ 5\ 6 \mid 9\ u\ 8)$$

Finally, the first u in o_1 (which should be 0 will cause a conflict) is replaced by 6 because of the mapping $0 \leftrightarrow 6$. Note such replacement is transitive, for example, the second u in o_1 should follow the mapping $7 \leftrightarrow 4, 4 \leftrightarrow 5, 5 \leftrightarrow 3$ and is hence replaced by 3. The final offspring are:

$$o_1 = (6\ 1\ 2 \mid 5\ 7\ 4\ 0 \mid 3\ 8\ 9) \ \ \textit{and} \ \ o_2 = (7\ 1\ 2 \mid 3\ 4\ 5\ 6 \mid 9\ 0\ 8)$$

The PMX crossover exploits important similarities in the value and ordering simultaneously when used with an appropriate reproductive plan [22].

Cycle Crossover. Original Cycle Crossover (CX) was proposed in [23], again for the TSP problem. Our CX builds offspring in such a way that each number (and its position) comes from one of the parents. We explain the mechanism of the CX with following example. Two parents:

$$p_1 = (0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9) \ \ \textit{and} \ \ p_2 = (3\ 1\ 2\ 5\ 0\ 4\ 7\ 9\ 6\ 8)$$

will produce the first offspring by taking the first number from the first parent:

$$o_1 = (0\ u\ u\ u\ u\ u\ u\ u\ u\ u)$$

Since every number in the offspring should come from one of its parents (for the same position), the only choice we have at this moment is to pick number 3, as the number from parent p_2 just "below" the selected 0. In p_1, it is in position 3, hence:

$$o_1 = (0\ u\ u\ 3\ u\ u\ u\ u\ u\ u)$$

which, in turn, implies number 5, as the number from p_2 "below" the selected 3:

$$o_1 = (0\ u\ u\ 3\ u\ 5\ u\ u\ u\ u)$$

Following the rule, the next number to be inserted is 4. However, selection of 4 requires the selection of 0, which is already in the list. Hence the cycle is formed as expected.

$$o_1 = (0\ u\ u\ 3\ 4\ 5\ u\ u\ u\ u)$$

The remaining 'u's are filled from p_2:

$$o_1 = (0\ 1\ 2\ 3\ 4\ 5\ 7\ 9\ 6\ 8).$$

Similarly,

$$o_2 = (3\ 1\ 2\ 5\ 0\ 4\ 6\ 7\ 8\ 9).$$

CX preserves the absolute position of the elements in the parent sequence [21].

Our experiments shows Classical crossover and CX have a stable but slow improvement rates, while PMX demonstrates oscillating but fast convergence. In our later experiments, the majority of the crossover is done by PMX. Classical crossover and CX are applied at a much lower probability.

7.2 Mutation Operator

Mutation is another classical genetic operator, which alters one or more genes (part of a chromosome) with a probability equal to the mutation rate. There are several known mutation algorithms which work well on different problems:

– Inversion: invert a subsequence.
– Insertion: select an number and insert it back in a random position.
– Displacement: select a subsequence and insert it back in a random position.
– Reciprocal Exchange: swap two numbers.

In fact the Inversion, Displacement and Reciprocal Exchange are quite similar to our neighborhood solution and diversification techniques used in Tabu Search and Simulated Annealing in previous sessions. We adopt a relatively low mutation rate of 1%.

We use population size $P = 1000$ for most cases. The evolution process starts with a random population. The population is sorted according to the objective function, the better the quality, the higher the probability it will be selected for reproduction. At each iteration, a new generation with population size $2P$ is produced and the better half, which is of size P, survive for the next iteration. The evolution process continues until certain stop criterion are met.

8 Experimental Results

We conducted extensive experiments on randomly generated data [1]. The graph for each test case contains one components, so that the cases cannot be partitioned into more than one independent sub-case.

All the programs implementing various heuristic methods are coded in GNU C++, with an extensive use of Standard Template Library (STL) for efficiency data structures like priority queue, set and map.

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 purpose.

Table 1. Experimental results for GYAP (Entries in the table show the minimum length of the yard required. Names of Data Sets show the number of pyramids in the file; LB:Lower Bound)

Data Set	Lower Bound	TSSTM	TSLTM	SWO	SWO+TS	SA	GA
P35	47	84	80	112	84	76	76
P86	218	616	583	748	550	572	550
P108	78	205	200	270	200	210	175
P127	221	660	649	891	671	671	550
P137	239	680	670	950	690	600	560
P142	40	135	125	190	140	120	115
P160	249	828	828	996	852	840	780
P167	243	814	792	1012	825	748	649
P187	302	948	912	1032	912	816	768

Table 1 illustrates the results and Table 2 shows the running time for each of the test performed in Table 1. GA is the most cost-effective approach while TSSTM has the simplest implementation for which it is not surprising to see that it achieves the poorest results. Long term memory certainly improves the performance of TS, though the improvement is not very obvious and stable sometimes. We believe one of the major difficulties with long term memory is the fine tuning of parameters, including the assignment of relative weights to yard length, residence frequency and transition frequency in the objective function.

The performance of SWO is poor in our experiments because of two reasons. Firstly, the BL packing routine makes it difficult to identify the bottleneck, or the "trouble makers". The objects are manipulated in a two-dimensional space, and assigning blame factor according to only one dimension may not well reflect the structure of the solution. Secondly, there is the loss of correspondence between physical layout and actual solutions. SWO is more sensitive to the problem domain as it needs to know the exact structure of the solution in order to assign

[1] All test data are available on the web with URL:
`http://www.comp.nus.edu.sg/~fuzh/GYAP`

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

Data Set	TSSTM	TSLTM	SWO	SWO+TS	SA	GA
P35	783	4521	1033	5462	4512	923
P86	948	6529	2312	5978	2351	1423
P108	2321	8392	4528	8432	8934	3452
P127	2783	9837	4678	10262	11023	6621
P137	2796	9640	4780	9892	9857	7048
P142	893	3428	1532	4582	3275	1094
P160	4781	15327	3085	18539	6793	3583
P167	6063	14294	3769	19852	6832	3781
P187	7806	17327	4085	20542	6673	6849

proper blame values. SA is the second best approach. We believe it is because the cooling schedule is not much affected by the loss of correspondence.

9 Conclusion

In this paper, the General Yard Allocation Problem is studied which involves two-dimensional rectangle packing as a related problem. We adopted a simple Bottom Left packing strategy as a first heuristic. Heuristics like Tabu Search, Simulated Annealing, Genetic Algorithms and "Squeaky Wheel" Optimization were then applied to problem in the extensive experiments and solutions obtained. Our approach sheds light on the use of these meta-heuristics on a set of problems, including those of packing where packing lists can change in time. In comparisons between results obtained, we found that the GA implementation achieves the best results to this problem.

References

1. F. Sabria and C. Daganzo: Queuing systems with scheduled arrivals and established service order. Transportation Research B, vol. 23. (1989) 159–175
2. E.K. Bish, T.-Y. Leong, C.-L. Li, J. W.C. Ng, D. Simchi-Levi: Analysis of a new vehicle scheduling and location problem. Naval Research Logistics, vol. 48. (2001) 363–385
3. L. Gambardella, A. Rizzoli, M. Zaffalon: Simulation and planning of an intermodal container terminal. Simulation, vol. 71. (1998) 107–116
4. P. Chen, Z. Fu, A. Lim: The yard allocation problem. In: Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02), Edmonton, Alberta, Canada (2002)
5. P. Chen, Z. Fu, A. Lim: Using genetic algorithms to solve the yard allocation problem. In: Proceedings of the Genetic and Evolutionary Computing Conference (GECCO-2002) New York City, USA (2002)
6. B. Baker, E.C. Jr., R. Rivest: Orthogonal packing in two dimensions. SIAM Journal of Computing, vol. 9. (1980) 846–855

7. S. Jacobs: On genetic algorithms for the packing of polygons. European Journal of Operational Research, vol. 88. (1996) 165–181
8. E. Hopper, B. Turton: An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. European Journal of Operational Research, vol. 128. (2001) 34–57
9. P. L. Hammer: Tabu Search. J.C. Baltzer, Basel, Switzerland (1993)
10. F. Glover, M. Laguna: Tabu Search. Kluwer Academic Publishers (1997)
11. S. Sait, H. Youssef: Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems. IEEE (1999)
12. D. Pham, D. Karaboga: Intelligent optimisation techniques: genetic algorithms, tabu search, simulated annealing and neural networks. London and New York, Springer (2000)
13. D. Clements, J. Crawford, D. Joslin, G. Nemhauser, M. Puttlitz, M. Savelsbergh: Heuristic optimization: A hybrid ai/or approach. In: Workshop on Industrial Constraint-Directed Scheduling (1997)
14. D.E. Joslin, D. P. Clements: "squeaky wheel" optimization. Journal of Artificial Intelligence Research, vol. 10. (1999) 353–373
15. D.E. Joslin, D. P. Clements: Squeaky wheel optimization. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98), Madison, USA (1998) 340–346
16. D. Draper, A. Jonsson, D. Clements, D. Joslin: Cyclic scheduling. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (1999)
17. J.M. Crawford: An approach to resource constrained project scheduling. In: Proceedings of the 1996 Artificial Intelligence and Manufacturing Research Planning Workshop (1996)
18. J.C. Culberson, F. Luo: Exploring the k-colorable landscape with iterated greedy. Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, David S. Johnson and Michael A. Trick (eds.). DIMACS Series in Discrete Mathematics and Theoretical Computer Science (1996) 245–284
19. S. Kirpatrick, C. Gelatt, Jr., M. Vecchi: Optimization by simulated annealing. Science, vol. 220. (1983) 671–680
20. J.H. Holland: Adaptation in natural artificial systems. Ann Arbor: University of Michigan Press (1975)
21. Z. Michalewicz: Genetic Algorithms + Data Structure = Evolution Programs. Springer-Verlag, Berlin Heidelberg New York (1996)
22. D. Goldberg, R. Lingle: Alleles, loci, and the traveling salesman problem (1985)
23. I. Oliver, D. Smith, J. Holland: A study of permutation crossover operators on the tsp (1987)