Ant Colony Optimization for the Ship Berthing Problem

Chia Jim Tong, Hoong Chuin Lau, and Andrew Lim

School of Computing
National University of Singapore
Lower Kent Ridge Road
Singapore 119260
chia@acm.org, lauhc@comp.nus.edu.sg, alim@comp.nus.edu.sg

Abstract. Ant Colony Optimization (ACO) is a paradigm that employs a set of cooperating agents to solve functions or obtain good solutions for combinatorial optimization problems. It has previously been applied to the TSP and QAP with encouraging results that demonstrate its potential. In this paper, we present FF-AS-SBP, an algorithm that applies ACO to the ship berthing problem (SBP), a generalization of the dynamic storage allocation problem (DSA), which is NP-complete. FF-AS-SBP is compared against a randomized first-fit algorithm. Experimental results suggest that ACO can be applied effectively to find good solutions for SBPs, with mean costs of solutions obtained in the experiment on difficult (compact) cases ranging from 0% to 17% of optimum. By distributing the agents over multiple processors, applying local search methods, optimizing numerical parameters and varying the basic algorithm, performance could be further improved.

1 Ant Colony Optimization

The Ant Colony Optimization (ACO) paradigm was introduced in [1], [2] and [3] by Dorigo, Maniezzo and Colorni. ACO has been applied effectively to the traveling salesman problem (TSP) [4] and the quadratic assignment problem (QAP) [5], among several other problems. The basic idea of ACO is inspired by the way ants explore their environment in search of a food source, wherein the basic action of each ant is: to deposit a trail of pheromone (a kind of chemical) on the ground as it moves, and to probabilistically prefer moving in directions with high concentrations of pheromone deposit.

As an ant moves, the pheromone it leaves on the ground marks the path that it takes. Another ant that passes by later can detect the pheromone and decide to follow the trail with high probability. If it does follow the trail, it leaves its own pheromone on it, thus reinforcing the existing pheromone deposit. By this mechanism, the movement of ants along a path between the nest and the food reinforces the pheromone deposit on it, and this in turn encourages further traffic along the path. This behavior characterized by positive feedback is described as autocatalytic.

P.S. Thiagarajan, R. Yap (Eds.): ASIAN'99, LNCS 1742, pp. 359–370, 1999.

On the other hand, ants may take a direction other than the one with the highest pheromone concentration. In this way, an ant does not always have to travel on the path most traveled. If an ant takes a path less traveled that deviates slightly from a popular path, and also happens to be better (shorter) than other popular paths, the pheromone it deposits encourages other ants to also take this new path. Since this path is shorter, the rate of pheromone deposit per ant that travels on it is higher, as an ant traverses a shorter distance in one trip. In this way, positive feedback can occur on this path and it can start to attract ants from other paths.

By the interplay of these two mechanisms, better and better paths emerge as the exploration proceeds. For the purpose of designing an algorithm based on this idea drawn from nature, an analogy can be made of: 1) real ants vs. artificial agents, 2) ants' spatial environment vs. space of feasible solutions, 3) goodness of a given path vs. objective function of a given solution, 4) desirability of taking a particular direction vs. desirability of making a certain decision in constructing the solution, 5) real pheromone at different parts of the environment vs. artificial pheromone for different solution choices. One of the main ideas behind ACO algorithms is how relatively simple agents can, without explicit communication, cooperate to solve a problem by indirect communication through distributed memory implemented as pheromone.

In this paper, we study how ACO can be applied effectively to the ship berthing problem (SBP), through the FF-AS-SBP algorithm, an application of ACO to the SBP. The focus of this study is not on the SBP itself or on fine-tuning our algorithm for maximum performance. Rather, it is on demonstrating that ACO can be applied effectively to the SBP. In Section 2, we formally describe the SBP. In Section 3, we describe a candidate solution representation, from which we adapt an indirect, first-fit (FF), solution approach in Section 4 so that it becomes more suitable for the complex nature of the SBP. In this section, we also describe a randomized FF algorithm and the basis of FF-AS-SBP. FF-AS-SBP is described in Section 5. By naming the algorithm FF-AS-SBP, we acknowledge that there could be many other ACO algorithms for the SBP. In Section 6, we describe the experiment and report and interpret the results, comparing FF-AS-SBP against the randomized FF algorithm. In this section, we also discuss how results could be further improved, how the algorithm lends itself to parallelization, and possible future work. Finally, Section 7 is the conclusion.

2 The Ship Berthing Problem

This problem, which has been studied in [6] and [7], can be defined as follows: ships ($\mathcal{S} = \{S_i: i = 1, 2, \ldots, n\}$) are specified to have lengths l_i, arrive at a port at specified times t_i and stay at the port for specified durations d_i. Each ship that arrives is to be berthed along a wharf line of length L, i. e., it is placed at the interval $(b_i, b_i + l_i)$ along the wharf line. Once berthed, its location is fixed for the entire duration of its stay. Also, each ship has a minimum inter-

ship clearance distance c_i^s and a minimum end-berth clearance distance c_i^b . Four types of constraints apply:

- Ships can only be berthed within the extend of the wharf line. No part of any ship can extend beyond the beginning or the end of the wharf line. More strongly, the distance from either end of a ship to either end of the wharf line cannot be less than the minimum end-berth clearance distance.

$$\forall i \in \{1, 2, \ldots, n\} \quad c_i^b \leq b_i \leq L - l_i - c_i^b$$

- No two ships can share the same space along the wharf line if the time intervals in which they are berthed intersect. More strongly, the end-to-end distance between them cannot be less than the minimum inter-ship clearance of either one of them.

$$\begin{aligned} &\forall i, j \in \{1, 2, \ldots, n\} \\ &\quad (t_i, t_i + d_i) \cap (t_j, t_j + d_j) \neq \emptyset \\ &\rightarrow \left(b_i - \max\left\{c_i^s, c_j^s\right\}, b_i + l_i + \max\left\{c_i^s, c_j^s\right\}\right) \cap (b_j, b_j + l_j) = \emptyset \end{aligned}$$

- A ship may be given a fixed berthing location (b_i is fixed for some values of i).
- A ship may be prohibited from berthing in certain intervals of the wharf line. More precisely, the interval bounded by the location of the two ends of a ship after it has been berthed cannot intersect with any of the prohibited interval.

$$(b_i, b_i + l_i) \cap (p, q) = \emptyset \qquad \begin{array}{l}\text{if constraint applies to } S_i, \text{ where} \\ (p, q) \text{ is some forbidden interval}\end{array}$$

The minimization version of the problem is to determine L_o the minimum length of the wharf line needed to berth all the ships subject to the given constraints.

The decision version of the problem is to determine whether berthing is possible, given a fixed value of L.

The density D is defined as the maximum total length of berthed ships at any one time [1]:

$$D = \max_{t \in (-\infty, +\infty)} \left(\sum_{i \in \{i : t_i \leq t < t_i + d_i\}} l_i \right)$$

It is easy to see that D is a tight lower bound on L.

In this paper, we also define a measure F, which we call the fragmentation, defined as:

$$F = 1 - \frac{\sum d_i l_i}{(\max(t_i + d_i) - \min(t_i))\, D}$$

The berthing scenario can be visualized as a 2-D plane where the x-axis represents time, the y-axis represents space (along the wharf line), and each ship

[1] For convienience, this definition ignores minimum end-berth and inter-ship clearance

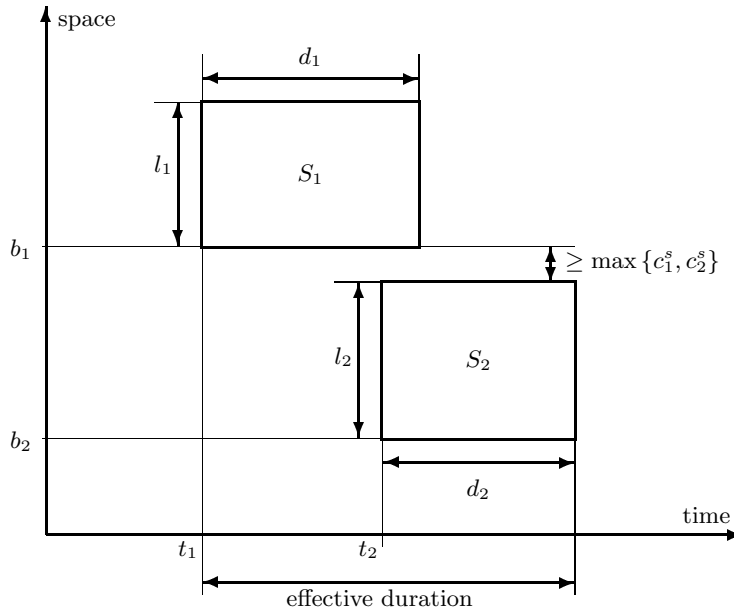

Fig. 1. A Sample Berthing Scenario

corresponds to a rectangular block whose x-extent is the time extent and the y-extent is the space extent of the ship. Figure 1 provides an example of this setup. Given an optimum solution of cost D, F is the percentage total area of regions not covered by a block, within the effective duration of the problem.

By having c_i^s and c_i^b set to zero and removing the last two constraints, the SBP becomes the dynamic storage allocation problem (DSA), which is known to be NP-complete and for which there is an 5-approximation algorithm (with a cost upper-bound of five times optimum)[8]. Since SBP is a generalization of DSA, it is also NP-complete. [7] provides a reduction of the NP-complete partition problem to the SBP, which is another way to show that SBP is NP-complete.

While simple to understand, a direct, geometric, representation for the SBP does not lend itself to an effective solution strategy. For this reason, a graph representation for the SBP was proposed in [7], together with an algorithm that uses this representation for finding good solutions.

In the new representation, each vertex v_i corresponds to the ship S_i and has weight l_i. Two distinct vertices v_i and v_j are connected by an (undirected) edge of weight $\max\left\{c_i^s, c_j^s\right\}$ iff $(t_i, t_i + d_i)$ and $(t_j, t_j + d_j)$ intersect. There are also zero-weight vertices v_l and v_r, and *arcs* $\langle v_l, v_i \rangle$ and $\langle v_i, v_r \rangle$ of weights c_i^b for each vertex v_i that corresponds to a ship S_i. The constraints related to prohibited and fixed berthing positions can be represented using a combination of auxilliary vertices

$\{v_{n+1}, v_{n+2}, \ldots\}$ acting as imaginary ships, auxilliary edges and auxilliary arcs. Details of this representation can be found in the original paper [7].

A feasible solution is any DAG, G, resulting from setting the direction of all edges, i. e., converting each edge to an arc of either direction. The cost of the solution, cost(G), is equal to both the length of the longest path in it and the value of L corresponding to that solution. The objective, therefore, is to find a DAG with as short a longest path as possible.

3 Solution as a Vertex Permutation

Not all edge direction assignments lead to a feasible solution as some lead to circuits, which do not exist in a DAG. Rather than searching all $2^{|E|}$ possible digraphs, many of which may not be DAGs, we can map each possible DAG to a vertex permutation and perform the search over the space of possible permutations. Each permutation $\pi = \begin{pmatrix} 1 & 2 & \cdots & n \\ \pi_1 & \pi_2 & & \pi_n \end{pmatrix} = (\pi_1 \pi_2 \ldots \pi_n)$ maps each vertex i to an integer label $\pi_i \in \{1, 2, ..., n\}$, where $n = |V|$. An edge $\langle u, v \rangle$ is set to arc $\langle u, v \rangle$ iff $\pi_u < \pi_v$ and $\langle v, u \rangle$ iff $\pi_v < \pi_u$. In this way, the digraph induced by a permutation is always a DAG, and each DAG has at least one corresponding permutation. A given permutation can always be normalized w. r. t. a given graph by first computing the DAG it induces and then performing a deterministic DAG labeling to obtain the normalized permutation.

In this paper, we use the terms 'labeling' and 'position' both to mean 'permutation'. 'Position' carries the meaning that vertices can be ordered in a sequence s. t. the first vertex in the sequence has label 1, the second vertex in the sequence has label 2 and so on, and the position of a vertex, which is equivalent to its labeling, is its position in that sequence.

The permutation representation or solutions seems to lend itself to a direct solution strategy similar to that used in [5], where the solution is also represented as a permutation. However, there is one important difference that makes the permutation representation problematic for the SBP. In the QAP, individual labels contribute piecewise additively to the final objective function. In the SBP, individual labels alone do not determine the cost of the longest path. This value is a function of the collective interaction between the vertex positions within the graph and there is no known straightforward relationship between the cost and the labeling over a subset of the vertices— over all possible DAGs, the cost of the longest path is not predicted or determined by individual vertex labels, or even arc directions, as the following two examples illustrate: The mere flip of a single arc can drastically change the cost; the reverse of a DAG G (all arcs are flipped; π_i is swapped with π_{n-i+1}) has the same cost as G.

Therefore, both the graph representation and the permutation representation seem unlikely to support ACO algorithms. For this reason, we adapted from the permutation representation a more indirect solution approach, which is explained in the next section.

4 First-Fit Approaches

In the standard first-fit algorithm (FF), ships are berthed (packed) one at a time in some predetermined order. When a ship S_i is to be packed, it is position as near to the front as possible without overlap with other ships S_j which have been berthed and whose time intervals $(t_j, t_j + d_j)$ intersect with the time interval of S_i, $(t_i, t_i + d_i)$. Of course, the packing of each ship is also done so that none of the other problem constraints are violated.

This can visualized on the geometric representation as positioning each block (ship), one at a time, with the x-coordinate fixed, minimizing the y-coordinate while not letting the current block overlap (or come too close) with over blocks which have been packed, or any other constraint to be violated.

When all the ships have been packed, the cost of the solution can be easily determined from the y-coordinate, length and minimum end-berth clearance of each ship.

The input the the FF algorithm, therefore, are the SBP problem itself and the order in which to pack the ships, represented as a permutation π, where ship i is the ith ship to be packed. It should be clear that for a given SBP problem, some permutations yield better solutions than others. In fact, it can be shown that there always exists a permutation that yields an optimum solution. However, for a given problem, it is difficult to perform a quick analysis to obtain an optimum, or even a good permutation. A straightforward approach, therefore, is to actually try different permutations.

4.1 Randomized First-Fit Algorithm

This algorithm simply generates a random permutation and calls FF with it. This is repeated for as many times as required to obtain better and better solutions. Improvements become increasing rare as the cost the the best discovered solution approaches the optimum.

4.2 ACO-First-Fit Algorithm

Using an ACO approach, we would like to obtain good permutations for doing FF.

Each permutation can be assigned a cost, which is the cost of the solution obtained by calling FF with it.

Unlike in DAG labeling, permutations could be related to cost in a straightforward way. In other words, for a given problem, there could be some easily represented, hidden, rules by which FF packing could be ordered so as to obtain a reasonable solution. For example, the complex nature of SBP seems not as strong in this representation, as can be seen in that in general, a permutation does not have the same cost as its reverse, and that displacing one ship in the packing sequence does not drastically change the cost, as are the case in DAG labeling.

Intuitively, it is the relative order in which packing is done that affects the cost, rather than the absolute position of each ship in the packing sequence—we are concerned with whether "S_i is packed before S_j" rather than whether "S_i is the jth ship to be packed". To support this mode of representation, given a permutation π of length n, we define a $n \times n$ boolean, lower-triangular matrix (only entries strictly below the diagonal are defined):

$$\mathbf{R} = \left(r_{ij} = \begin{cases} \text{true if } \pi_i < \pi_j \\ \text{false if } \pi_i > \pi_j \end{cases} \right)$$

It can be easily seen that there is a one-to-one function that maps π to $\mathbf{R}$, and that the reverse is not true, i. e., some values of $\mathbf{R}$ are invalid. A permutation π can be constructed from constraints encoded in (a valid) $\mathbf{R}$ as follows:

1. Set $\pi_1 = 1$
2. Determine (from the 2nd row of $\mathbf{R}$) whether $\pi_2 < \pi_1$ (then $\pi_2 = 1$ and π_1 should be shifted right, i. e., set to 2) or $\pi_2 > \pi_1$ (then $\pi_2 = 2$).
3. Determine (from the 3rd row of $\mathbf{R}$) whether $\pi_3 = 1, 2$ or 3 and set it to that value. $\forall i \neq 3$ s. t. $\pi_i \geq \pi_3$, increment π_i by 1.
4. In the same fashion determine and update π_i for $i \in \{4, \ldots, n\}$.

This above definition provides the basis for the pheromone design of FF-AS-SBP.

5 Ant Colony Optimization for the SBP

A vital factor in the effectiveness of any ACO algorithm is the design of the pheromone trail data structure—the information contained therein and how it is used. The FF-AS-SBP algorithm presented below uses the design we have experimentally found to give the best performance.

The algorithms in this section are not presented in a computationally optimized form, or the form in which we implemented them, but only to describe their computational effects.

The FF-AS-SBP algorithm takes as input the problem in the graph representation and the density D of the problem, which is used as a lower bound. It stores solutions as FF permutations and returns the cost of the best permutation found. The cost, L_π of a permutation π is defined as the cost of the packing obtained from performing FF with π. The parameters to the algorithm are α, N, $K$and γ. α and γ are real numbers in the interval $[0, 1)$. N and K are positive integers.

The non-scalar variables used are $\mathcal{B}$ and $\mathbf{T}$. $\mathcal{B}$ is a set of permutations — the set of the best K solutions discovered. $\mathbf{T} = (\tau_{ij})$ is a $n \times n$ matrix of real values each representing the intensity of a pheromone trail. τ_{ij} represents the desirability of setting $\pi_i < \pi_j$, in the spirit of matrix $\mathbf{R}$ of Section 4.2 (unlike in $\mathbf{R}$, the matrix is not lower-triangular but diagonal elements are irrelevant).

FF-AS-SBP Algorithm

1. Let $\tau_0 = (nD)^{-1}$. Let $L_{\min}$ denote $\min_{\pi \in \mathcal{B}} L_\pi$.
2. $\mathcal{B} \leftarrow \emptyset$; $\mathbf{T} \leftarrow \tau_0$
3. Populate $\mathcal{B}$ with a random permutation for $2K$ times
4. Delete from $\mathcal{B}$ all but the best K permutations
5. $\mathbf{T} \leftarrow (1 - \alpha)\mathbf{T}$
6. For each $\pi \in \mathcal{B}$, update the pheromone matrix $\mathbf{T}$ according to the Global Update Algorithm using a reinforcement value of L_π^{-1}
7. For each ant $a = 1, 2, \ldots, N$, build a solution permutation according to the Ant Solution Algorithm and add the solution to $\mathcal{B}$ while keeping $|\mathcal{B}| \leq K$ by deleting worst permutations as necessary
8. If $L_{\min} > D$ and neither the iteration limit nor time limit has been exceeded then goto 5
9. Return $L_{\min}$

Steps 1–4 perform initialization. Steps 5–6 perform global pheromone update using the K best solutions. Step 7 performs ant exploration and local pheromone update.

We now explain the role of each input parameter. α represents the rate of pheromone evaporation and determines conversely, the persistence of memory from earlier iterations. N is the number of ants used in step 7 for constructing ant solutions. K is the number of globally best solutions to remember for the purpose of the global update in step 6. γ represents the preference for exploitation over exploration when ants construct their solutions in the Ant Solution Algorithm.

We now describe the sub-algorithms in detail.

Global Update Algorithm

This simple algorithm reinforces pheromone trails associated with a given solution π by a given reinforcement value Δ.

For each $u \in \{1, 2, \ldots, n-1\}$
  For each $v \in \{u+1, \ldots, n\}$
    If $\pi_u < \pi_v$
      $\tau_{uv} \leftarrow \tau_{uv} + \Delta$
    else
      $\tau_{vu} \leftarrow \tau_{vu} + \Delta$

Ant Solution Algorithm

This is the algorithm that each ant performs in building a solution π. The solution (permutation) is incrementally constructed by setting π_i in order, starting on π_1 and ending on π_n, similar to the method described in Section 4.2. The difference is that at each step, instead of being determined by a boolean matrix of constraints, each π_i is set to a value probabilistically chosen from all possible values based on their desirability.

At each step, the desirability measure of each candidate value, a function of different pheromone strengths, is evaluated and either exploitation or exploration is chosen probabilistically. Exploitation is chosen with probability γ. In

exploitation, the candidate value with the highest desirability is chosen. In exploration, these values are probabilistically chosen based on their desirability. We now define the desirability measure d_p of a given value p when choosing a value for π_c.

d_p is defined as

$$\prod_{v \in \{1,2,\ldots,c\}} \begin{cases} \tau_{vc} \text{ if } \pi_v < p \\ \tau_{cv} \text{ otherwise} \end{cases}$$

With d computed, exploitation chooses the value with highest d, while exploration chooses each candidate value p with probability proportionate to d_p. After a value p has been chosen and π has been updated accordingly, all pheromone trails τ_{ij} associated with the p, i. e., that appear as terms in the product in the above equation, are diminished:

$$\tau_{ij} \leftarrow (1 - \alpha)\tau_{ij} + \alpha\tau_0$$

The above is known as the local (pheromone) update.

6 Experiment

In this section we discuss the design, execution and results of our experiment to study the performance of the FF-AS-SBP algorithm, as well as our interpretation of the results and possible future work.

For simplicity in the design of the experiments, we considered only problem instances with zero edge weights. Also, the constraints related to clearance distances and fixed and forbidden positions are absent from the test cases. In other words, only problems that are also DSA problems are used. Nevertheless, the results should extend to the general case.

While we have not mapped out the characteristics of the parameter space as it is too vast, we have empirically found that setting $\alpha = 0.1$, $N = n$, $K = \lceil 0.1m \rceil$ ($m = |E|$) and $\gamma = 0.9$ seems to yield reasonably good solutions. This is the parameter setting adopted in all the test runs recorded in this section.

Each of the 6 test cases consist of one connected component–i. e., there is no way to cut the geometric representation into two parts with a vertical line without also cutting a block. This is because a problem of more than one component can be divided into these components to be solved individually.

The test cases were generated by a block cutting algorithm that generates blocks of varying height and width from an initial rectangular block, which is recursively divided into smaller and smaller blocks (some intermediate blocks are L-shaped blocks). Hence, all the problems generated are compact, i. e., have zero fragmentation, a property that is expected to make them difficult to solve optimally.

The randomized FF algorithm and FF-AS-SBP are run 10 times on each case, and each run is given $3|V|$ seconds to live. For each algorithm applied to each of the 6 test cases, the costs from each of the 10 runs are reported in ascending

Table 1. Randomized First-Fit Algorithm Results

Case	D	$\|V\|$	$\|E\|$	Runs									
1	80	20	119	80	80	80	80	80	80	80	80	80	80
2	120	30	338	120	120	120	121	121	121	125	125	126	126
3	160	40	457	175	186	187	187	188	189	190	190	190	193
4	200	50	633	245	246	247	248	249	251	251	252	254	254
5	240	60	783	298	300	300	301	301	305	306	307	308	312
6	320	80	1641	416	417	417	419	420	422	424	425	425	427

Table 2. FF-AS-SBP Results

Case	D	$\|V\|$	$\|E\|$	Runs									
1	80	20	119	80	80	80	80	80	80	80	80	80	80
2	120	30	338	120	120	120	120	120	120	120	120	121	121
3	160	40	457	173	173	175	176	176	177	177	182	182	184
4	200	50	633	205	225	231	232	233	235	238	239	244	247
5	240	60	783	254	254	255	258	258	258	258	258	258	258
6	320	80	1641	368	369	369	369	370	375	376	379	384	387

order. These results are shown in Table 1 and Table 2 and summarized in Table 3.

The randomized FF algorithm acts as a control for checking whether pheromone information does make a real difference in the quality of obtained solutions. For this reason, time-to-live of the runs is expressed in terms of actual time taken rather than number of iterations.

FF-AS-SBP performed consistently and significantly better than randomized FF, especially for larger cases, except in case 1, where both algorithms always returned an optimum solution.

Our interpretation of the experiment results is that pheromone information did help to improve the quality of the solution when applied to an FF-based approach to the SBP. We also note that it was necessary for us to adopt this indirect approach to avoid building an algorithm that would futilely explore the rough terrain of the more direct approach based on DAG labeling.

Table 3. Experiment Summary

				Randomized FF			FF-AS-SBP		
Case	D	$\|V\|$	$\|E\|$	Min	Mean	Max	Min	Mean	Max
1	80	20	119	80	80	80	80	80	80
2	120	30	338	120	122.5	126	120	120.2	121
3	160	40	457	175	187.5	193	173	177.5	184
4	200	50	633	245	249.7	254	205	232.9	247
5	240	60	783	298	303.8	312	254	256.9	258
6	320	80	1641	416	421.2	427	368	374.6	387

In view of the current results, the following areas merit further investigation:

- Heuristic measure for (partial) FF permutations while they are being constructed. An obvious choice is the cost of the partial solution.
- Alternative pheromone matrix design.
- The full characteristics of the parameter space in affecting the dynamics of the ant exploration and pheromone matrix. This could provide insight on how to optimize convergence and quality of solution, by setting parameters appropriately, or even dynamically varying them during the search through the use of appropriately designed meta-heuristics.
- Employing heterogeneous ants in the search. Ants could differ by quantitative parameters or qualitatively by the search strategy they use. The diversity introduced by having heterogeneous ants could contribute to overall algorithm performance. If investigation demonstrates that employing heterogeneous ants is indeed a profitable approach, the result could extend to other fields of distributed computing and lead to interesting investigation on the use of heterogeneous agents in those fields as well.
- How performance can be improved by increasing the number of ants. If increasing the number of ants can significantly improve performance, then there exists the possibility of distributing the work of many ants to multiple processors to achieve significant speedup, since individual ants operate independently of one another.
- How local search techniques can be combined with the basic ACO technique to improve performance. An example of such a technique is the famous tabu search method[9].

7 Conclusions

In this paper, we have formulated a pheromone information design and coupled it with the FF algorithm to produce an ACO algorithm that obtains reasonably good solutions for the SBP, thus demonstrating that the ACO paradigm can be applied effectively to the SBP. Morever, we have demonstrated that pheromone structure does not need to be directly related to the physical structure with which a target problem naturally is expressed.

The major ingredients of our algorithm, FF-AS-SBP, are the FF algorithm, permutation construction and the pheromone information that supports this construction. FF-AS-SBP compares favorably against the randomized FF algorithm.

We have also proposed a few key areas for further investigation to obtain even better performance and deeper insight. Of special interest is the possibility of using heterogeneous agents in FF-AS-SBP and in distributed computing in general.

8 Acknowledgement

This research was supported in part by the NUS Research Grant RP3972679.

References

[1] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: An autocatalytic optimizing process. Technical Report 91-016 Revised, Dipartimento Elettronica e Informazione, Politecnico di Milano, Italy, 1991.

[2] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics*, B(26):29–41, 1996.

[3] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992. In Italian.

[4] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learing approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[5] L. M. Gambardella, È. D. Taillard, and M. Dorigo. Ant colony for the QAP. Technical Report IDSIA 97-4, IDSIA, Lugano, Switzerland, 1997.

[6] K. Heng, C. Khoong, and A. Lim. A forward algorithm strategy for large scale resource allocation. In *First Asia-Pacific DSI Conference*, pages 109–117, 1996.

[7] Andrew Lim. An effective ship berthing algorithm. In *IJCAI '99*, volume 1, pages 594–605, 1999.

[8] Jordan Gergov. Approximation algorithms for dynamic storage allocation. In *ESA '96: Fourth Annual European Symposium*, pages 52–61, 1996.

[9] F. Glover. Tabu search - part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.