Replicating Multi-quality Web Applications Using ACO and Bipartite Graphs*

Christopher B. Mayer[1], Judson Dressler[1], Felicia Harlow[1], Gregory Brault[1], and K. Selçuk Candan[2]

[1] Department of Electrical and Computer Engineering
Air Force Institute of Technology**
Wright-Patterson AFB, OH 45433
[2] Computer Science and Engineering Department
Arizona State University
Tempe, AZ 85287

Abstract. Since its introduction the Ant Colony Optimization (ACO) meta-heuristic has been successfully applied to a wide range of combinatorial problems. This paper presents the adaptation of ACO to a new NP-hard problem involving the replication of multi-quality database-driven web applications (DAs) by a large application service provider (ASP). The ASP must assign DA replicas to its network of heterogeneous servers so that user demand is satisfied at the desired quality level and replica update loads are minimized. Our ACO algorithm, AntDA, for solving the ASP's replication problem is novel in several respects: ants traverse a bipartite graph in both directions as they construct solutions, pheromone is used for traversing from one side of the bipartite graph to the other and back again, heuristic edge values change as ants construct solutions, and ants may sometimes produce infeasible solutions. Testing shows that the best results are achieved by using pheromone and heuristics to traverse the bipartite graph in both directions. Additionally, experiments show that AntDA outperforms several other solution methods.

1 Introduction

Ant Colony Optimization (ACO) is a meta-heuristic based on the natural behavior of real ants and their ability to efficiently solve minimization problems using just a few simple rules. In ACO, ant agents traverse a graph's edges (can be directed or undirected) as they link the graph's vertices together to construct a solution. After finding a solution, some subset of the ants deposits pheromone on the edges used in its solution in proportion to the solution's quality. That is, edges critical to the best solutions receive more pheromone than other edges. Pheromone evaporation makes edges less attractive over time and, hence, acts to weed out undesirable solutions. To move from one vertex to another when

* Research funded by NSF grant 998404-0010819000.

** The views expressed herein are those of the authors and do not reflect the official policy or position of the U.S. Air Force, Dept. of Defense, or the U.S. Government.

solving, each ant computes a value for outgoing edges by combining the edge's pheromone concentration and heuristic desirability. The ant randomly selects an outgoing edge to traverse with higher-valued edges selected more often. Making random decisions using both pheromone (representing past good solutions) and heuristics (which guides ants in the absence of pheromone) encourages the exploration of solutions in the neighborhood of known good solutions and allows for enough variability that distant solutions are unlikely to go unnoticed. The book [2] contains an excellent overview of the ACO meta-heuristic.

Structurally, ACO algorithms usually consist of a doubly-nested loop. The outer loop controls the number of trials executed. A trial represents a separate solving of the problem. The inner loop governs the time steps in each trial. During a time step each ant traverses the graph and constructs a solution, ants deposit pheromone on the graph's edges, and pheromone evaporates. The updated graph is then used as the basis for the next time step. Being a stochastic process, ACO requires multiple trials each consisting of many time steps.

ACO algorithms have been implemented for many NP-complete combinatorial problems such as traveling salesman [10, 11], job-shop scheduling [5], graph coloring [6, 8], sequential ordering [13], and multidimensional knapsack [1, 14]. Many of the ACO algorithms cited above produce solutions that compare favorably with the best known algorithms for solving the aforementioned problems.

In this paper, ACO is adapted to solve a new NP-hard assignment problem involving the replication of multi-quality database-driven web applications (DAs) by a large application service provider (ASP). In the NP-hard DA Replication Problem (DA*rep*) the ASP must assign DA replicas to its network of heterogeneous servers so that user demand is satisfied at the desired quality level and replica update loads are minimized [17]. DA*rep* is similar to several multi-commodity facility location problems [18, 19] and multidimensional knapsack problems [9, 20]. Though algorithms for solving these similar problems have been proposed, they cannot be directly applied to DA*rep*. Inspired by the success of multidimensional knapsack ACO algorithms [1, 14] and other ACO algorithms for solving problems similar to DA*rep* [7, 12, 16, 21], we have implemented an ACO-style algorithm called AntDA that has several novel or rarely seen features.

- Ants traverse a bipartite graph consisting of a set vertices representing the DAs (things to be assigned) and a set of vertices representing the servers (where things can be assigned). Ants move back-and-forth between the two sets of vertices along directed edges as they construct solutions. Other ACO algorithms for solving assignment problems of a bipartite nature use either a bipartite graph in which ants travel in only one direction (e.g., [12, 14, 16] or transform the problem so that vertices represent potential assignments (fully enumerate the solutions space) and are all of the same type (e.g., [8, 21]).
- Pheromone on DA-to-server edges and server-to-DA edges is used for traversing the graph in both directions. Although not the first to use pheromone for dual purposes [2, 8], AntDA is one of the few that do. Moreover, experiments show that pheromone in both directions produces the best solutions in terms of both update load (cost) and convergence rates.

– In DA*rep* ants may produce infeasible solutions. Since infeasible solutions may be examined on the way to feasible solutions, ants with infeasible solutions can deposit pheromone. Unlike [16] where infeasible solutions pay a constant penalty in terms of pheromone deposited, AntDA ants pay a penalty proportional to their infeasibility.
– Heuristic values of the edge selection equations change as the ants construct their solutions. Other ACO algorithms with non-constant heuristic values include [1, 14].

The rest of this paper is organized as follows. Section 2 introduces and formalizes DA*rep*. Section 3 presents the AntDA algorithm and the Server-Filling local optimization heuristic. Experimental results of AntDA's performance and the impact of bi-directional bipartite pheromone and the Server-Filling heuristic are given in Section 4.

2 The DA Replication Problem

An Application Service Provider (ASP) (e.g., Akamai or ASP-One) is a company that specializes in hosting web applications on behalf of clients. Database-driven web applications (DAs) are a particular kind of web application hosted by an ASP in which responses to user requests are built, in part at least, by querying a database. The database allows the DA to customize responses based on user input and enables the DA's content to change dynamically. Some DAs can provide multiple freshness/quality levels of service in order to meet the needs of different types of users [3, 4]. For example, consider an on-line brokerage where users have been grouped into two categories: high-quality and low-quality. High-quality users expect very fresh (timely) quotes. Low-quality users, on the other hand, are pleased with the default or moderately-fresh content. Note that low-quality users can be satisfied by high-quality results, but the reverse is not true.

In order to furnish each DA with enough processing capacity to meet demand, an ASP replicates the DAs on its network of servers. For a DA replica to function, the replica's local database needs to be updated with fresh content and/or kept synchronized. The load on a server hosting a DA replica has two components: (1) the **request load** incurred by responding to user requests and (2) the **update load** stemming from database changes. To simplify real DA behavior, request complexity is assumed to be independent of quality level. Thus the request load for each quality offered by a DA is solely dependent on the number of requests received. However, since database freshness determines a replica's quality level, higher quality levels require more frequent database updates. Thus, for a given DA, update loads increase with quality level.

The ASP must establish replicas so that demand for each quality of each DA is met and update burdens (costs) are minimized. Obviously, the ASP cannot allow any server to become overloaded. That is, the sum of the request loads and update loads of the replicas hosted by a server cannot exceed the server's capacity. We have chosen the sum of all replica update loads, **system update**

The Database-driven Application Replication Problem

Definitions: The ASP has m servers, $S = \{1, \ldots, m\}$. Each server $s \in S$ has a processing capacity denoted by C_s. The ASP has n customer-provided DAs to be hosted, $D = \{1, \ldots, n\}$, on its m servers. Each $d \in D$ operates at one or more service quality levels, $Q_d = \{1, \ldots, q, \ldots, qmax(d)\}$, where $qmax(d)$ is the highest level offered by DA d. The request load for DA d, RL_d, is the sum of the request loads, denoted by rl_d, for each of its quality levels: $RL_d = \sum_{q \in Q_d} rl_{d,q}$. For each service quality of a DA d there is a certain update load required to maintain that service quality: $UL_d = \{ul_{d,1}, \ldots, ul_{d,q}, \ldots, ul_{d,qmax(d)}\}$. Let $x_{s,d,q} \in \{0,1\}$ be a binary variable that indicates that server s is hosting a replica of a certain $\langle d, q \rangle$ pair where "$\langle d, q \rangle$ pair" is shorthand for "quality q of DA d." Let $\lambda_{s,d,q} \in [0,1]$ denote the fraction of the request load of a $\langle d, q \rangle$ pair, $rl_{d,q}$, assigned to server s. The update load experienced by server s depends on the quality level of the DA replicas it hosts:

$$ul_s = \sum_{d \in D} \sum_{q \in Q_d} x_{s,d,q} \cdot ul_{d,q}. \tag{1}$$

Server s's request load is the fraction of each $\langle d, q \rangle$ pair's request load sent to it:

$$rl_s = \sum_{d \in D} \sum_{q \in Q_d} \lambda_{s,d,q} \cdot rl_{d,q}. \tag{2}$$

Objective and Constraints: The ASP seeks an assignment of DA replicas to servers that minimizes the system-wide update burden, UB, and is subject to four constraints.

$$\min\ UB = \min \sum_{s \in S} \sum_{d \in D} \sum_{q \in Q_d} ul_{d,q} \cdot x_{s,d,q} \tag{3}$$

1. Request load for each quality of each DA is satisfied: $\sum_{q \in Q_d} \sum_{s \in S} \lambda_{s,d,q} = 1$.
2. Only one quality level of a DA is hosted by a server: $\sum_{q \in Q_d} x_{s,d,q} \leq 1$.
3. A server's processing capacity cannot be exceeded: $ul_s + rl_s \leq C_s$.
4. Requests processed by a replica must meet or exceed the request's quality expectation, q_r: $\sum_{q_s | q_s, q_r \in Q_d \wedge q_s \geq q_s} x_{s,d,q_s} \geq \lambda_{s,d,q_r}$.

Fig. 1. DA*rep* formalized as an integer linear program.

burden, as the cost for an ASP to minimize as it also indirectly minimizes number of replicas and network bandwidth. The ASP's replication problem is called the **DA Replication Problem** (DA*rep*). DA*rep* is formalized as an integer linear program (ILP) in Fig. 1.

3 AntDA: An ACO Algorithm for DA*rep*

In AntDA, ants operate on a bipartite graph representing an instance of DA*rep* (Fig. 2). The graph, $G = (V, E)$, consists of a set of vertices, V, and edges connecting vertices, E. The vertices are divided into two groups, DQ and S, such that $V = DQ \cup S$ and $DQ \cap S = \emptyset$. Each vertex in DQ represents a $\langle d, q \rangle$

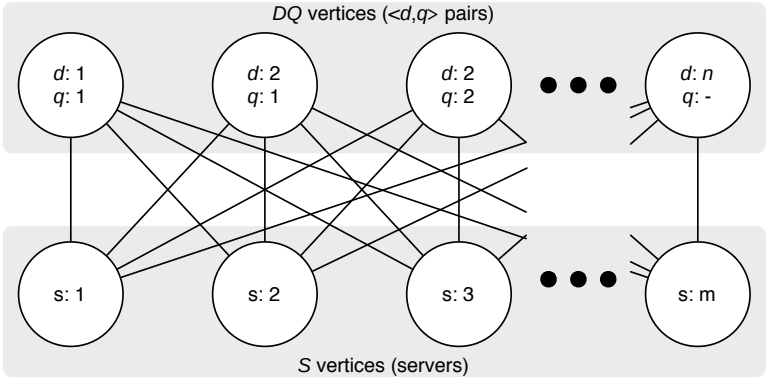

Fig. 2. The bipartite problem graph used by AntDA. Although all $\langle d, q\rangle$ pairs are connected to servers via directional edges and vice-versa, single non-directional edges are shown here for simplicity.

pair. The vertices in S represent the servers. Each $dq \in DQ$ is connected to every $s \in S$ by a directed edge (dq, s). Similarly each $s \in S$ is connected to every $dq \in DQ$ by a directed edge (s, dq). Even though each edge (dq, s) has a reverse edge (s, dq), undirected edges are not used since pheromone is interpreted differently on edges of type (dq, s) versus edges of type (s, dq).

Ants construct solutions by moving back and forth between vertices in DQ ($\langle d, q\rangle$ pairs) and vertices in S (servers) assigning dq request load and creating a replica or adjusting the service quality of an existing replica on the server chosen for assignment. Ants work independently (maintain their own solution spaces) but share the same graph. An ant works on a solution until either server capacity is exhausted or all request load has been assigned to the servers. Once the ants have solved, they deposit pheromone on the shared graph, pheromone evaporation takes place, and then the next time step begins. Ants are placed at a random server vertex at the beginning of each time step.

3.1 Transitioning From $\langle d, q\rangle$ pairs to Servers.

When at vertex dq, ant k must find a server to which the $\langle d, q\rangle$ pair represented by dq will create a replica and assign load. Let S^k_{dq} be the set of servers (vertices) upon which request load of the $\langle d, q\rangle$ pair represented by vertex dq can be assigned. Let $rem(C_s)$ represent the unused (remaining) capacity of server s. Server s is available for assignment if the net change in update load on s caused by its hosting DA d at quality q is less than $rem(C_s)$ (i.e., s will be able to handle request load for the $\langle d, q\rangle$ pair). This is the **server hosting condition**.

The probability that ant k selects edge (dq, s) is

$$p^k_{dq,s}(t) = \begin{cases} \frac{[\tau_{dq,s}(t)]^\alpha \cdot [\eta_{dq,s}]^\beta}{\sum_{s \in S^k_{dq}} [\tau_{dq,s}(t)]^\alpha \cdot [\eta_{dq,s}]^\beta}, & \text{when } s \in S^k_{dq} \\ 0, & \text{when } s \notin S^k_{dq}. \end{cases} \tag{4}$$

$\tau_{dq,s}(t)$ is the pheromone concentration on (dq, s) at time step t. α and β are constants governing the relative importance of pheromone to the heuristic desirability, $\eta_{dq,s}$, of traveling along edge (dq, s):

$$\eta_{dq,s} = \frac{rem(C_s)}{\sum_{s \in S^k_{dq}} rem(C_s)} \tag{5}$$

where $rem(rl_{d,q})$ is the amount of request load for $\langle d, q \rangle$ pair yet to be assigned to a server. The heuristic is based on the idea that greedily selecting the largest server should reduce the number of replicas created and, thus, update burden produced. Note that $\eta_{dq,s}$ values change as servers are assigned.

After selecting edge (dq, s) the ant moves from vertex dq to vertex s. Once at s the ant creates a replica for the $\langle d, q \rangle$ pair and assigns as much remaining request load of the $\langle d, q \rangle$ pair, $rem(rl_{d,q})$, to s as possible. If a replica of DA d already exists on s, then the ant adjusts the quality level of the replica if needed (increases the update load of the replica). The server's remaining capacity, $rem(C_s)$, is decreased based on the amount of update load and request load assigned.

After creating a replica of DA d at quality level q on server s, the ant invokes following two-step **Server-Filling (SF)** heuristic.

1. SF first tries to avoid the creation of extra replicas of d by finding other qualities of d that will completely fit on s. More specifically, SF looks for another quality $r \in Q_d$ such that all of $rem(rl_{d,r})$ can be assigned to s. Note that update load differences have to be accounted for since it may be that $r > q$ and hence $ul_{d,r} > ul_{d,q}$. SF assigns the highest r found, repeating with additional qualities of d if possible. Let y be the highest quality of d assigned to s at the end of this step.
2. If s still has spare capacity after step 1, SF looks for the highest quality y of d such that $u < y$ and assigns as much request load of quality u as possible to the replica.

The SF heuristic is an optional, but beneficial, part of AntDA; in experiments SF reduced update burden by over 4% on average (Section 4).

3.2 Moving From Servers to $\langle d, q \rangle$ pairs

An ant at vertex s must decide which $\langle d, q \rangle$ pair should be assigned next. Let DQ^k_s be the set of dq vertices which are still capable of being assigned to servers. A $\langle d, q \rangle$ pair can be assigned if:

1. it has some amount of unassigned request load ($rem(rl_{d,q}) > 0$), and
2. there is at least one server $s \in S$ for which the server hosting condition (see previous page) is satisfied with respect to the $\langle d, q \rangle$ pair.

If $DQ_s^k = \emptyset$, the algorithm terminates. Otherwise, the probability that ant k selects edge (s, dq) is given by:

$$p_{s,dq}^k(t) = \begin{cases} \frac{[\tau_{s,dq}(t)]^\alpha \cdot [\eta_{s,dq}]^\beta}{\sum_{s \in DQ_s^k} [\tau_{s,dq}(t)]^\alpha \cdot [\eta_{s,dq}]^\beta}, & \text{when } dq \in DQ_s^k \\ 0, & \text{when } dq \notin DQ_s^k. \end{cases} \quad (6)$$

$\tau_{s,dq}(t)$ is the pheromone concentration on (s, dq). α and β again control the relative importance of pheromone and heuristic desirability, in this case $\eta_{s,dq}$:

$$\eta_{s,dq} = \frac{ul_{d,q} \cdot rem(rl_{d,q})}{\sum_{dq \in DQ_s^k} ul_{dq} \cdot rem(rl_{d,q})}. \quad (7)$$

Dividing $ul_{d,q} \cdot rem(rl_{d,q})$ by a server's remaining capacity, $rem(C_s)$, estimates the update burden incurred by creating replicas on servers of size $rem(C_s)$. Eq. (7) is an appropriate heuristic since it prefers $\langle d, q \rangle$ pairs most likely to produce high update burdens (no matter which servers are used). Note that $\eta_{s,dq}$ values change as the ant constructs its solution.

After making its selection, the ant traverses the edge to the selected $\langle d, q \rangle$ pair and then transitions back to a server node (Section 3.1).

3.3 Pheromone Update Rule

When each ant has constructed a solution to DA*rep* it is time to deposit pheromone on the shared graph. By finding a solution, an ant has essentially assigned values for the $x_{s,d,q}$ and $\lambda_{s,d,q}$ variables described in the formal version of the problem shown in Fig. 1.

Since better solutions have lower update burdens, the amount of pheromone deposited by ants should be inversely proportional to a solution's update burden. However, low update burdens are not always better – some ants' solutions may be infeasible (i.e., they do not assign all request load). Differentiating between feasible and infeasible solutions when deciding how much pheromone to deposit on the edges used in an ants solution is easily handled. Let $UB_k(t)$ be the update burden of ant k's solution after time step t as computed by (3). We adjust $UB_k(t)$ to account for infeasible assignments as follows:

$$UB'_k(t) = \frac{UB_k(t)}{\left(\frac{RL_k(t)}{RL}\right)^\omega} \quad (8)$$

where $RL_k(t)$ is the amount of request load assigned by ant k in time step t and ω is a constant that determines the magnitude of the penalty paid for not assigning all request load. Eq. (8) increases the update load of an infeasible assignment based on how much request load was satisfied raised by ω.

Once $UB'_k(t)$ has been determined, it is used to calculate the amount of new pheromone ant k will deposit. We allow the ants with the m best solutions to

Table 1. Parameter and condition values for AntDA experiments.

$\alpha = 1$	$\beta = 8$
$\rho = 0.8$	$\omega = 4$
$\gamma = 1$	# Ants $= \lvert DQ \rvert + \lvert S \rvert$
Initial Edge Pheromone $= 0.1$	$m = \lfloor$# Ants $\cdot\, 0.1 \rfloor$

deposit pheromone after each time step. More specifically, if edge e was used in the ith best solution and $i \leq m$, then the amount of pheromone deposited on e by the ant that produced the ith best solution is

$$\Delta_e^i(t) = \frac{\gamma}{UL_i'(t)} \tag{9}$$

where γ is a constant. We set $\gamma = 1$ during our experiments as we found it had little, if any, impact on performance. If an edge e was not used by ant i, then $\Delta_e^i(t) = 0$.

Let $\Delta\tau_e(t) = \sum_{i=1}^{m} \Delta_e^i(t)$ be the amount of new pheromone to be deposited on edge e because of the m solutions chosen. The amount of pheromone on the edges in graph G is then updated as is typically done in ACO [2]:

$$\tau_e(t+1) \leftarrow (1-\rho) \cdot \tau_e(t) + \rho \cdot \Delta\tau_e(t) \tag{10}$$

4 AntDA Validation

This section presents the results of experiments that compare AntDA with other solution methods, reveal the importance of the Server-Filling (SF) optimization heuristic, and the importance of pheromone and heuristics on ants traversing the bipartite graph.

Experimental Configuration. Unless otherwise noted, AntDA was run with the parameter values and conditions shown in Table 1. The values and conditions were determined through trial and error. For the most part, AntDA was fairly insensitive to the values shown in Table 1 (changing any particular value had only minor effects). The one parameter that had a major impact was the number, m, of ants that deposit pheromone at the end of each time step (Section 3.3). We found that setting m to be the top 10% of the number of ants cut the convergence rate by as much as six times without affecting update burdens. Test cases were subjected to fifty trials of 400 time steps each.

Results for performance comparisons were obtained by using a random assignment algorithm, Random, a greedy algorithm, Greedy, and the LINGO ILP solver [15]. Random randomly picks a $\langle d, q \rangle$ pair with non-zero remaining request load and assigns it to a random server capable of hosting it. Random reports the best solution found out of 1000 trials. The Greedy algorithm makes assignments by choosing the $\langle d, q \rangle$ pair with the highest predicted update burden. This is the same as computing the numerator of Eq. 7 for each $\langle d, q \rangle$ pair still capable of

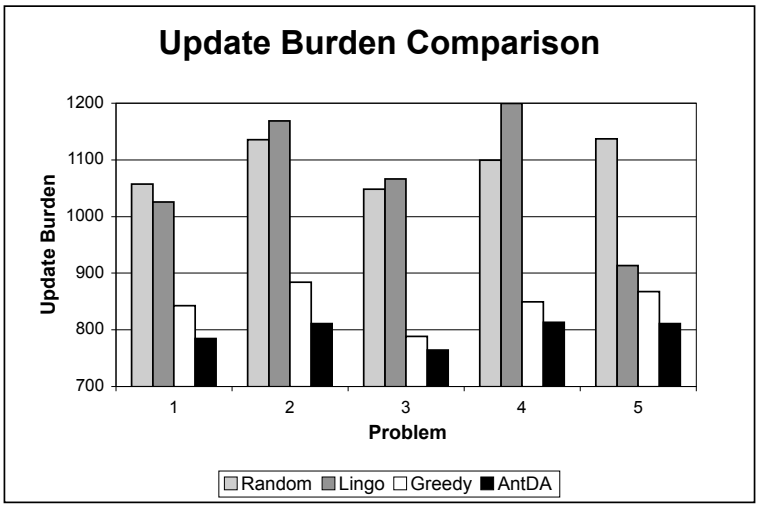

Fig. 3. AntDA performance compared to other solution methods.

being assigned to some server. The $\langle d, q \rangle$ pair selected is then assigned to the server with the most remaining capacity. The Greedy algorithm also executes the Server-Filling optimization heuristic. Both Random and Greedy algorithms stop once all request load has been assigned or when no more request load can be assigned to any server. LINGO solves DA*rep* using the ILP formulation of Fig. 1. Although ILP solvers such as LINGO are the only known method besides complete enumeration that can find guaranteed optimal solutions, execution times can be prohibitive. Therefore, we allotted four hours for LINGO to work on problems. This was sufficient time for LINGO to produce feasible, but not optimal, solutions and provides a notion of DA*rep*'s complexity.

Due to space limitations, experimental results are shown for just five test cases (hypothetical DA*rep* instances) involving five DAs of three quality levels each.[3] The update load for quality levels 1, 2, and 3 were randomly generated within the bounds of $[5, 14]$, $[16, 25]$, and $[27, 36]$, respectively. The update loads for quality three make it impossible to assign request load to the smallest of the servers. The number of nodes in the test cases were either 70 or 75. Request load increased in conjunction with quality level and was randomly generated in the ranges of $[34, 100]$, $[133, 200]$, $[233, 300]$, respectively. Once the DAs were generated, servers were added to complete each test case. Servers were added in sets of five servers having sizes $\{25, 50, 75, 100, 125\}$ until the Greedy algorithm produced feasible solutions.

[3] Although we have a suite of over 70 test cases with equal or fewer quality levels per DA and varying update loads and request loads per quality level, we display the test cases that proved to be the most difficult to solve. Results for the other test cases are similar to those shown here.

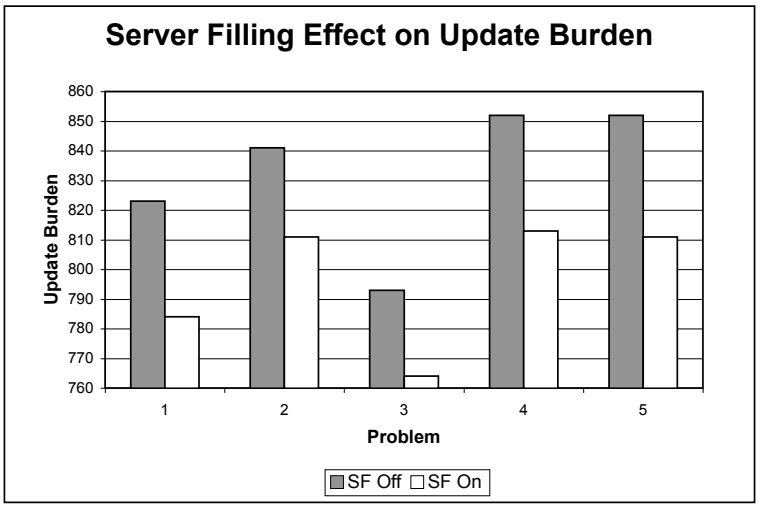

Fig. 4. Effect of Server-Filling heuristic on AntDA performance.

Results and Analyses. Figure 3 shows the performance of AntDA and the other solution methods for the five test cases. Clearly AntDA produces better solutions than the three other methods. Although it produces the lowest update burdens, AntDA has higher solution times than the other methods. The Greedy can produce a solution in millisecond, the Random algorithm needed about 1.5 minutes, ILP was cut off after 4 hours, and AntDAtook just under four hours to complete a test case. However, AntDAfound its best solution by the 18th time step on average with a standard deviation of 10.17. Using the rule-of-thumb that 95% of values fall within two standard deviations of their mean, this means that AntDA will find its best answer within 50 time steps $(18 + (2 \cdot 10.17) \leq 50)$ 95% of the time for the five test cases. Reducing the number of time steps brings AntDA down to a half hour per test case (about 30 seconds per trial).

The next set of results show the impact of the Server-Filling (SF) optimization heuristic. Fig. 4 shows the minimum update burdens produced by AntDA with and without SF. Using SF reduced the update burden by an average of 35.6 points which translates to a 4.28% reduction on average.

The use of a bipartite graph in which the ants follow pheromone and heuristics in order to traverse from one set of vertices to the other and back raises the question of how important pheromone and heuristics are to AntDA performance. Table 2 shows the impact of all possible cases of turning pheromone and heuristics off for the fifth of the five test cases. Similar results were obtained from the other four test cases. For example, line 1 of Table 2 shows performance results when pheromone is turned off when moving from an application vertex to a server (the pheromone components are removed from Eq. 4). Similarly, line 4 shows the impact of ignoring heuristics when moving from a server to a $\langle d, q \rangle$ pair(by removing the heuristic components of Eq. 6) .

Table 2. Importance of pheromone and heuristics on AntDA performance.

Line	Pheromone Off	Heuristic Off	Min UB	Max UB	Avg UB	StdDev UB	Conv. Avg.
1	DQ-to-S	-	838	866	864.78	9.13	198.55
2	S-to-DQ	-	812	836	824.48	4.83	4.84
3	-	DQ-to-S	830	887	854.16	14.36	44.22
4	-	S-to-DQ	815	864	841.68	9.55	221.22
5	both	-	815	826	822.24	2.60	148.30
6	-	both	835	896	863.40	11.23	121.56
7	DQ-to-S	S-to-DQ	874	874	858.20	9.25	204.04
8	S-to-DQ	DQ-to-S	833	973	911.24	23.08	17.30
9	-	-	811	822	816.26	2.90	25.28

Line 9 of Table 2 contains the statistics for normal AntDA operation. Note that lines 2, 4, and 5 come close to meeting the minimum update burden (UB) of line 9. However, the other lines all have higher maximum and average update burdens. Also, lines 4 and 5 converge on their best solution (rightmost column) after 221.22 and 148.30 time steps, respectively, which is much worse than the normal mode of operation (line 9). Line 2 fares much better with respect to convergence rate and minimum update burden, but still has a higher maximum and average update burden than the normal mode. Altogether, Table 2 indicates that using pheromone and heuristics to make edge selections in both directions (normal operating mode) gives the best overall performance.

5 Conclusion

In this paper the Ant Colony Optimization (ACO) meta-heuristic was successfully applied to a new NP-hard problem, DA Replication Problem(DA*rep*), in which an application service provider must replicate multi-quality database-driven web applications on its network of servers at minimal cost. The ACO formulation for DA*rep*, AntDA, is the first to use a fully bipartite graph. Ants deposit and follow pheromone on directed edges connecting a set of application vertices with a set of server vertices and then back from servers to applications. Other novel aspects of AntDA include dynamically changing heuristic values and the possibility that ants may produce infeasible solutions. Experiments showed that AntDA outperforms several other solution methods. Moreover, the use of the Server-Filling optimization heuristic by AntDA decreased costs by over 4%. Tests in which pheromone and heuristic values were removed from the ants' decision-making process as they traveled the bipartite graph revealed that pheromone and heuristics in both traversal directions delivers the best performance.

References

[1] I. Alaya, C. Solnon, and K. Ghédira. Ant algorithm for the multidimensional knapsack problem. In *International Conference on Bioinspired Optimization Methods and their Applications*, pages 63–72, 2004.

[2] E. Bonabeau, M. Dorigo, and T. Théraulaz. *From Natural to Artificial Swarm Intelligence*. Oxford University Press, New York, 1999.
[3] L. Bright and L. Raschid. Using latency-recency profiles for data delivery on the web. In *VLDB*, pages 550–561, 2002.
[4] M. Cherniack, E. F. Galvez, M. J. Franklin, and S. Zdonik. Profile-driven cache management. In *ICDE*, pages 645–656, 2003.
[5] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant system for job-shop scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science*, 34(1):39–53, 1994.
[6] F. Comellas and J. Ozón. An ant algorithm for the graph colouring problem. In *First International Workshop on Ant Colony Optimization (ANTS '98)*, 1998.
[7] O. Cordón, I. Fernández de Viana, and F. Herrera. Analysis of the best-worst ant system and its variants on the QAP. *Lecture Notes in Computer Science*, 2463:228–234, 2002.
[8] D. Costa and A. Hertz. Ants can colour graphs. *Journal of the Operational Research Society*, 48:295–305, 1997.
[9] M. Dawande, J. Kalagnanam, P. Keskinocak, R. Ravi, and F. S. Salman. Approximation algorithms for the multiple knapsack problem with assignment restrictions. *Combinatorial Optimization*, 4(2):171–186, 2000.
[10] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics–Part B*, 26(1):29–41, 1996.
[11] C. J. Eyckelhof and M. Snoek. Ant systems for a dynamic tsp. *Lecture Notes In Computer Science*, 2463:88–99, 2002.
[12] W. K. Foong, H. R. Maier, and A. R. Simpson. Ant colony optimization for power plant maintenance scheduling optimization. In *Conference on Genetic and Evolutionary Computation (GECCO)*, pages 249–256, 2005.
[13] L. M. Gambardella and M. Dorigo. An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 12(3):237–255, 2000.
[14] G. Leguizamon and Z. Michalewicz. A new version of ant system for subset problems. In *Congress on Evolutionary Computation*, pages 1459–1464, 1999.
[15] LINDO Systems, Inc. http://www.lindo.com.
[16] H. R. Lourenço and D. Serra. Adaptive search heuristics for the generalized assignment problem. *Mathware and Soft Computing*, 9(2):209–234, 2002.
[17] C. B. Mayer. *Quality-based Replication of Freshness-Differentiated Web Applications and Their Back-end Databases*. PhD thesis, Arizona State University, December 2005.
[18] J. B. Mazzola and A. W. Neebe. Lagrangian-relaxation-based solution procedures for a multiproduct capacitated facility location problem with choice of facility type. *European Journal of Operational Research*, 115:285–299, 1999.
[19] H. Pirkul and V. Jayaraman. A multi-commodity, multi-plant, capacitated facility location problem: Formulation and efficient heuristic solution. *Computers and Operations Research*, 25(10):869–878, 1998.
[20] H. Shachnai and T. Tamir. Noah's bagels - some combinatorial aspects. In *International Conference on Fun with Algorithms*, 1998.
[21] T. Stützle and M. Dorigo. *New Ideas in Optimization*, chapter ACO algorithms for the quadratic assignment problem, pages 33–50. McGraw-Hill, 1999.