Ant Algorithm Parallelization

Jake Krohn
Department of Computer Science
University of Minnesota, Morris
krohnk@mrs.umn.edu

Abstract

Using the social insect colony metaphor, researchers have been able to develop surprisingly fast and accurate algorithms to help solve combinatorial optimization problems such as the Traveling Salesman or Quadratic Assignment problems. Informally known as 'ant algorithms,' these approaches use a swarm of simple information-sharing agents, which self-organize into what is often the best solution to a given problem. A key to this self-organizing behavior is modification of the environment, which is done by each agent and which affects the later behavior of all of the other agents in the system. In natural ant colonies, these environmental modifications occur through the use of chemical secretions called pheromones; computer scientists have successfully put the pheromone idea to use in settings far removed from the biological world. Research in this area is promising, and is limited only by the computational power available to simulate the movement of multiple agents over a defined problem space. Faster convergence towards an optimal solution can be shown when a parallel implementation of an ant algorithm is used. This is due to the increased simultaneous searching of solution spaces over a parallel or distributed architecture. This paper discusses one such system, which is written in Java and uses the JPVM parallel frameworks as a message passing interface to help solve the Traveling Salesman Problem. The design of the parallel algorithm and the information sharing strategy are addressed, as the approach used in a parallel/distributed architecture differs from the approach used in a sequential computing environment. Some early results are given and a comparison to a similar sequential algorithm will be shown.

Keywords

Ant algorithm, ants, Ant System, distributed processing, emergence, Java, JPVM, parallel processing, stimergy, self-organization, Swarm Intelligence, Traveling Salesman Problem

INTRODUCTION

In an ironic turn of events, many researchers in the field of computer science have pushed aside their keyboards and CPU's in favor or a more archaic technology: the ant colony. And they do so with good reason. Ants, as well as other social insects, have an uncanny knack for collective problem solving.

Much work has been done by computer scientists and ethologists to model this behavior. Through these models, researchers have gained insight as to how answers to complex problems arise from the interactions of simple organisms. This knowledge has then been successfully transformed into a series of problem solving techniques – a form of Artificial Intelligence called Swarm Intelligence. In Swarm Intelligence, problem solving is not centrally managed by one controlling force; rather a distributed "swarm" of relatively simple components work in unison. From these basic interactions, a solution emerges.

The Swarm Intelligence model has been applied to a number of problems (see [1, 11] for an overview) and successes are many. However, this approach faces the same equalizing limit as all others: namely, the bound imposed by available processing power.

One way to circumnavigate this obstacle is by using simple ideas of parallel processing over a distributed network of processors. The resulting speedup should be significant enough to warrant the initial investment into parallel development.

In this paper, I will first explain the Swarm Intelligence model using the common Traveling Salesman Problem as an example of the model's applicability. An understanding of how the model is applied to this popular benchmark problem will lead to a general understanding of the model as a whole.

While the Swarm Intelligence approach lends itself naturally to parallelization, issues such as information sharing strategies exist and will be discussed. The choice of programming language and parallel framework is equally as important and will also be covered. The paper will also include some early results and a comparison to a similar sequential algorithm. The paper will conclude with comments on these results and possible directions for future research.

The Swarm Intelligence Model

In Swarm Intelligence, the guiding model of intelligence is the social insect colony. Characteristic of this kind of intelligence are the concepts of self-organization and stimergy.

Self-Organization

Self-organization is a set of dynamic mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components [1]. Having no

knowledge of the global state of the system, these components (called *agents*) make decisions using only local information. From these local decisions, a global pattern emerges.

Self-organization relies on four basic properties [1]:

1. *Positive feedback* (*amplification*), which is the reinforcement of behavior that leads to a desirable outcome. In Swarm Intelligence, positive feedback is modeled by a digital equivalent to the attractive chemical secretions released by ants and other organisms, which is called *pheromone*.
2. *Negative feedback*, which helps counterbalance positive feedback so that the reinforcement of one (possible sub-optimal) behavior will not constantly occur at the expense of other (possibly more favorable) solutions. In a system of ants, pheromone *evaporation* is one such example of negative feedback.
3. *Fluctuations* in the system, be they random actions on the part of the agents or unplanned changes in the environment itself. Fluctuations ensure the discovery of new (perhaps better) solutions and provide the seed from which emergent structures grow. As an example of this, ants in colonies move about the land with a somewhat unpredictable pattern, ensuring this property remains intact.
4. *Multiple interactions* of the agents, along with reuse of information provided by others, creates a system where the intelligence is stored not at the local level, but within the structure of the system itself. Ants in colonies encounter many fellow ants and leave pheromone trails while carrying out tasks; these meetings and trails provide a data store that is independent of the ants themselves.

Stimergy

Interactions within a society of social insects can take on one of two forms: direct or indirect. Direct interactions are obvious in nature: bodily contact, visual contact, food exchange, etc. Indirect interactions are subtler, but are no less important. In fact, much of the research in Swarm Intelligence revolves around understanding and modeling these interactions. Indirect communication occurs when one agent modifies the environment in which it resides in such a way that the behavior of subsequent agents (including itself) is affected. Thus, storage of information occurs not at the individual level but instead at the colony level. Because of this, reliance on any specialized type of agent is reduced and information can be maintained over time with no apparent effort on the agent's part.

This coordination through modification is called *stimergy*. By exploiting the stimergic approach to coordination, researchers have been able to design a number of successful algorithms in such diverse application fields as combinatorial optimization, routing in communication networks, task allocation in a multi-robot system, exploratory data analysis, and graph drawing and partitioning [5]. In this paper, I will describe the application of one such algorithm to the Traveling Salesman Problem.

The Traveling Salesman Problem

The Traveling Salesman Problem is a well-known NP-complete problem that is often used as a benchmark when measuring the performance of new general-purpose heuristics. If one wishes to find the optimal solution for a given problem set, all of the possible solutions must be explored; this value grows exponentially as the problem size increases, thus requiring intense computation.

In the Traveling Salesman Problem, the goal is to find a closed tour of minimal length connecting n given cities. Each city must be visited once and only once. The problem can be defined generally on a graph, $G = (V, E)$, in which the vertices (V) represent cities and the edges of the graph (E) are the connections between the cities. A value, d_{ij}, is assigned to each edge and represents the distance between city i and city j.

Ant Algorithms

Ant System

In his pioneering paper on the field of Swarm Intelligence [7], Marco Dorigo introduced *Ant System*, the problem solving heuristic derived from the study of any colonies. The algorithms that use Ant System are called a*nt algorithms*.

Dorigo was interested in the use of ant colonies as a metaphor for problem solving. He was not interested in the realistic simulation of ant colonies. Because of this, Dorigo's artificial ants (called *ants*) have some major differences with their real counterparts.

First, ants have a memory. In the case of the Traveling Salesman Problem, this memory is used to store a list of previously visited cities.

Second, ants are not completely blind. While natural ants rely on chemical signals to navigate, Dorigo's ants can sample their external environment by whatever means necessary. For the purposes of the problem at hand, ants attempting to solve the Traveling Salesman Problem are aware of the distances between cities.

Finally, due to obvious constraints imposed by the architecture of current computers, artificial ants live in a world where time is discrete.

Ant System and the Traveling Salesman Problem

To solve the Traveling Salesman Problem using the ideas of Ant System, a graph G is randomly populated with m ants, where

$$m = \sum_{i=1}^{n} b_i(t) \qquad (1)$$

and $b_i(t)$ is the number of ants in town i at time t.

Each ant has the following characteristics:

- To model pheromone deposits, when traveling from town *i* to town *j*, each ant lays a substance, called *trail*, on edge (*i, j*);
- It chooses the town to go to with a probability that is a function of the town distance and the amount of trail present on the connecting edge;
- A data structure, called a *tabu list*, is associated with each ant. It is used to store the cities visited by the ant and forbids the ant to visit them again until a cycle has been completed. The vector containing the tabu list of the k-th ant is $\mathbf{tabu_k}$ and $\mathbf{tabu_k(s)}$ is the s-th element of the tabu list of the k-th ant.

Trail Updating

Let $\tau_{ij}(t)$ be the *intensity of the trail* on edge (*i, j*) at time *t*. After *n* iterations of the algorithm, the trail intensity becomes

$$\tau_{ij}(t+n) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t, t+n) \tag{2}$$

where ρ is a coefficient such that (1 - ρ) represents the evaporation of trail between time *t* and time $t + n$ and

$$\Delta\tau_{ij}(t, t+n) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t, t+n) \tag{3}$$

where $\Delta\tau_{ij}^{k}$ is the quantity per unit length of trail substance laid on edge (*i, j*) between time *t* and $t + n$ by ant *k*.

Trail updating may be done after each successive move of the ant (termed *local updating*) or after all ants have completed one cycle (*global updating*). For the purposes of this study, both strategies were used in combination; see [7] for a comparison between the use of the two strategies.

In local updating, pheromone values are updated on edge *(i, j)* every time an ant moves from city *i* to city *j*. The amount of new pheromone added to the edge is equal to

$$\Delta\tau_{ij}^{k}(t, t+1) = \begin{cases} \dfrac{Q}{d_{ij}} & \text{If the k}^{\text{th}}\text{ ant goes from } i \text{ to } j \\ & \text{between time t and t + 1} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Where Q is some constant quantity of pheromone. The new quantity of pheromone deposited on an edge is inversely proportional to the edge length; thus, over time, shorter edges will receive more pheromone, which leads to a positive feedback loop of increased use and further reinforcement.

During global updating, only one ant is allowed to update the trail values. This *elitist strategy* requires that only the ant with the iteration-best tour be allowed to deposit additional pheromone. The amount of pheromone deposited on edge *(i, j)* is equal to

$$\Delta\tau_{ij}{}^{k}(t,t+n) = \begin{cases} \dfrac{Q}{L^{ib}} & \text{If the k}^{\text{th}}\text{ ant uses edge } (i,j) \text{ in its tour} \\ 0 & \text{otherwise} \end{cases} \qquad (5)$$

Where L^{ib} is equal to the tour length of the iteration best ant. Trail updating is also done using L^{gb}, the length of the globally-best solution found thus far. In the algorithm used for this study, global updating was done after every 15 iterations of the ants, and used L^{gb} to determine the quantity of pheromone deposited.

Similar to local updating, the new quantity of pheromone deposited is inversely proportional to a certain value; this time, the value in question is tour length, not edge length. Thus, edges which constitute shorter tours are reinforced more which leads to another positive feedback loop of more use and greater reinforcement.

To counteract the positive feedback loops implicit in both local and global updating, this study introduced *sporadic best path* updating, in which all trail values were reset to an arbitrary value and the best tour was reinforced sporadically. That is, edges present in the best tour were reinforced with pheromone with a probability of p_r. For the purposes of this study, p_r = .85. This strategy aimed to increase tour diversity while still maintaining exploration in the neighborhood of good solutions.

Transition Probability

The transition probability from town *i* to town *j* for the k-th ant is

$$p_{ij}{}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}\cdot[\eta_{ij}]^{\beta}}{\sum_{u\in\text{allowed}}[\tau_{iu}(t)]^{\alpha}\cdot[\eta_{iu}]^{\beta}} & \text{If j} \in \text{allowed} \\ 0 & \text{otherwise} \end{cases} \qquad (6)$$

where **allowed** = {j is not in $\mathbf{tabu_k}$} and η_{ij} is the *visibility* of town *j* from town *i*, which is simply the value $1/d_{ij}$. The values α and β are parameters that control the effect of trail and visibility on the transition. Essentially, by manipulating the values of α and β, one can transform the transition probability from a greedy heuristic that values visibility over trail ($\beta >> \alpha$) into one that uses only the autocatalytic process for decision-making ($\alpha >> \beta$). Best results have been found when both values are approximately in the same range [4, 7].

A Modified Ant System Algorithm

The algorithm used by the author to solve the Traveling Salesman Problem closely mimics the *MAX-MIN* approach presented by Stützle and Hoos [16]. One major difference is the exclusion of a local search optimizing procedure such as 3-Opt [12], which results in a faster algorithm with less accuracy. However, as the results will later show, this exclusion does not impact the performance of the algorithm in any major way.

A basic sequential algorithm of the process used by the author to solve the TSP is shown below:

1. Initialize all data structures. Set $t = 0$ and set an initial pheromone trail value on every edge. Place the m ants randomly on the nodes of the graph.
2. For each ant, do:
   2.1. Until every city has been visited, do:
      2.1.1. Choose the town to move to, with probability p_{ij}, which is given by equation (6) and move the ant to the chosen town.
      2.1.2. When traveling from city i to city j, apply the local trail updating rule given in equations (2) and (4) with $n = 1$
      2.1.3. Insert the chosen town into the ant's *tabu list*.
3. Every 15 iterations, update trail values using by equations (2) and (5) and the best tour found thus far.
   3.1. Every 40 iterations, clear all trail values and sporadically reinforce the edges included in the best tour found thus far.
4. Memorize the shortest path found thus far and empty all *tabu lists*.
5. If the end condition (usually defined as a number of cycles) is not met, then:
   - Set t = t + n
   - Go to step 2

   Else:
   - Print the shortest path and stop

In words, the algorithm works as follows:

At time $t = 0$, initialization occurs and ants are distributed on the map. Initial trail intensities, usually quite high, are set for each edge. As ants move from town i to town j with probability p_{ij}, edge (i, j) is updated with a new trail value, as evaporation and pheromone deposits are simulated. This city to city movement is repeated for each ant until a valid TSP tour has been formed. Tour formation is repeated for all ants, at which point the tour with the shortest total length is determined. If the shortest tour found during this iteration is less than the previously-found shortest tour, the new tour is memorized. Every 15 iterations, trail pheromone is added to the edges that comprise the best tour. Every 40 iterations, the trail values are reset and the edges in the best tour are sporadically reinforced with probability equal to some user-chosen value of p_r. Tabu lists are then cleared, ants are placed anew on the map, and the algorithm repeats until the user-defined condition, usually a certain number of cycles, is met.

Parallel Solution Strategies

While it is true that this population-based approach lends itself naturally to parallel processing, implementation details prove to be non-trivial and worthy of further investigation. Broadly put, a different strategy is required when running the algorithm in parallel as opposed to sequential execution. Overviews of different approaches can be found in [2, 13, 15].

This study focuses on the information exchange strategy between simultaneous instances of the ant algorithm running in a distributed parallel environment. "Information exchange strategy" is a broad term that encompasses not only the actual sending of data across communication lines, but also the frequency of the communications and the level of importance assigned by one instance to data gleaned from fellow parallel instances.

Design strategy and implementation

To allow for rapid development and easy modification, the initial sequential program was created using Sun Microsystems' Java. Although a slower language than similar languages such as C++ in terms of computational performance, the large API and automatic garbage collection mechanism made Java an attractive choice for this study. One negative side effect of this decision, however, is the inability to compare the results of this study to results of other studies in terms of execution times, since Java's interpreted bytecode makes it inherently slow. It is hoped, however, that the strategies discussed here can be put to use in other similar algorithms regardless of the implementation environment.

The choice of Java led to a search for an appropriate message-passing parallel framework. A PVM [17] clone, JPVM [9], was found and used. The use of JPVM was very similar to the popular PVM for C/C++ and FORTRAN, but with the convenience of working in an all-Java environment.

Worker Communication

Because of the stimergic nature of the artificial ant colony, the only information that is necessary to share amongst the instances of the algorithm is the pheromone trail information. As in real ant colonies, the artificial pheromone represents the major channel of colony communication, directing the ants towards good solutions. Information sharing should not be too frequent, however, for constant knowledge of the state of the global system can impair a colony's diversity, which must be maintained if new, better, solutions are to be generated. Conversely, if too little information is shared, colonies may find themselves stagnated in local optima. Therefore, it is important to choose a work/update schedule that gives fair play to both requirements. From observations of the system at work, it was determined that a 250 local iterations per update schedule would be followed. This value is by no means a proven optimum and further study into the matter would be worthwhile.

When designing the parallel algorithm, a distinction was made between the *global-best* tour and the *local-best* tour. The former is the best tour found thus far by all instances of the algorithm, while the latter is a record of the best tour found thus far by an instance, which is often a different value for each instance in a particular run. These values are stored separately by each instance and are used in different ways by the trail update mechanism. The locally-best solution is the value used to determine the quantity of pheromone deposited when performing the sporadic trail update, while the globally-best solution is used for complete trail reinforcement, and is discussed in the next section.

Trail Update Pressure

Another issue to consider when sharing information between colonies is the degree of influence the globally-best solution is allowed to have over the pheromone trail values in each instance. If too much pressure is exerted by the globally-best solution, the search space will narrow to an unacceptably small range and no new solutions will be discovered. Too little pressure will have the opposite effect – the search will be unfocused and sub-optimal solutions will result.

To better understand the effects of the trail update pressure, κ, this study ran experiments at levels of $\kappa = \{0.0, 0.5, 1.0, 3.0, 6.0\}$. Global trail update values were computed using an equation similar to (2):

$$\tau_{ij}(t+n) = \rho \cdot \tau_{ij}(t) + \kappa(\Delta\tau_{ij}(t, t+n)) \qquad (2')$$

Thus, κ affects the amount of new trail (determined by L^{gb}) that is added to the old.

Results and Discussion

To check the veracity of the trail pressure update hypothesis, tests were run on two problem sets: the 58-city brazil58 and 42-city swiss42 problems, both of which are available at the TSPLIB [14]. All tests were run in parallel on four similar Intel Celeron (one 466 MHz, three 500 MHz) machines running Debian Linux 2.2.19. Message passing was implemented through JPVM and as noted earlier, the work to update ratio was 250:1. Each machine ran for 2000 iterations and thus exchanged information with the master worker a total of seven times. Parameter settings used for all cases were $\alpha = 1$, $\beta = 3$, $\rho = 0.45$, and $p_r = 0.85$. Twenty runs were performed for each level of $\kappa = \{0.0, 0.5, 1.0, 3.0, 6.0\}$. Two pieces of data were extracted from the output and recorded: the shortest path found after completion of the program run and the number of exchanges required until the discovery of the best (not necessarily optimal) solution.

Results for the experiment were somewhat surprising. While behavior for large values of κ did indeed constrict exploration and lead to a high incidence of sub-optimal solutions, the same expected behavior for small values of κ did not appear. Instead, a small κ (0.0, 0.5) led to faster discovery of shorter tours. No evidence was shown towards the negative effects of an unfocused search.

Parallel Results

Figures 1 and 2 show the average tour length found by the algorithm over the range of trail reinforcement pressure values, κ. A general upward trend is noted for the tour length in both cases and best performance is observed when $\kappa = 0.0$.

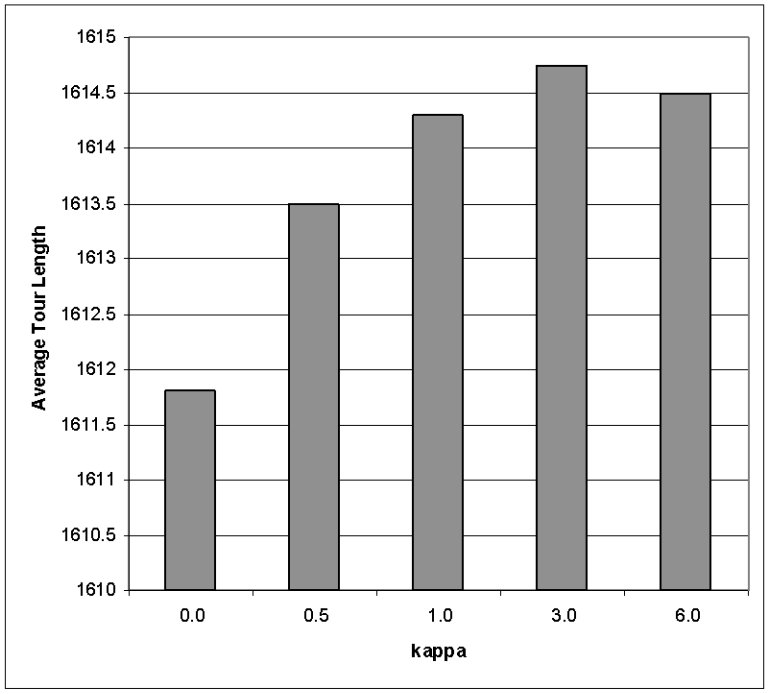

Figure 1: Effect of Global-Best Trail Update Pressure (κ) on Average Tour Length – bayg29

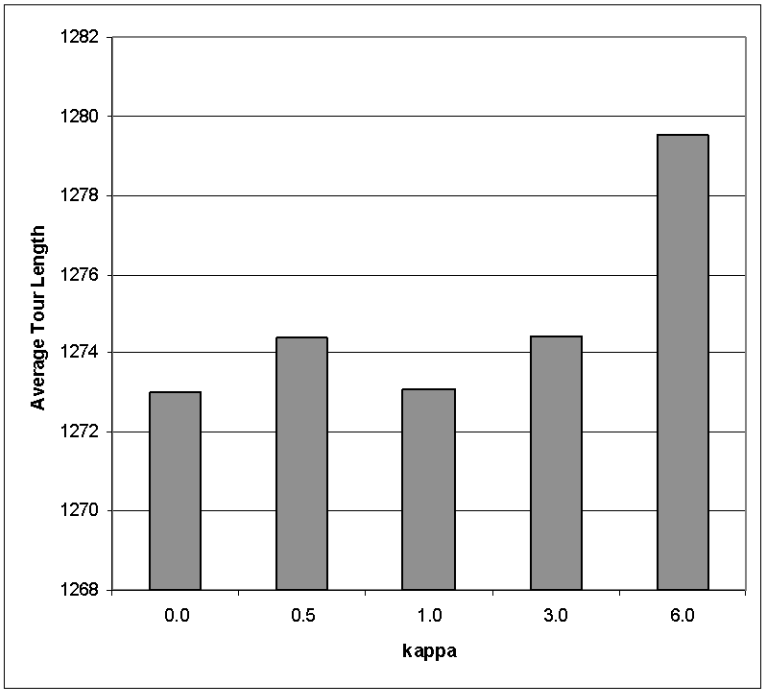

Figure 2: Effect of Global-Best Trail Update Pressure (κ) on Average Tour Length – swiss42

Using the Mann-Whitney test with $\alpha = 0.05$, near-statistical significance (p-value = .0674) was found between the tour lengths in the swiss42 test when comparing the $\kappa_1 = 0.0$ and $\kappa_2 = 6.0$ results.

To determine an overall best κ-value, the number of optimal solutions found for each κ-value and the average number of global iterations required to find that value were compared for each problem set using the data listed in Table 1.

Table 1: Number of optimal solutions for a given κ-value and the average number of global iterations to find the optimal value.

	0.0	0.5	1.0	3.0	6.0
bayg29					
Number of times optimal tour was found.	17	13	13	11	13
Average global iterations to optimal tour.	3.59	3.62	3.76	5.09	4.73
swiss42					
Number of times optimal tour was found.	20	19	18	18	15
Average global iterations to optimal tour.	2.80	2.59	3.06	3.28	3.8

In comparing the number of global iterations required to achieve the best (not necessarily optimal) solution for a given run, again, $\kappa = 0.0$ proved to be the best general-purpose setting. In the swiss42 problem, $\kappa_1 = 0.0$ was significant against $\kappa_2 = 6.0$ with a p-value of 0.405. $\kappa_1 = 0.5$ was also significant against $\kappa_2 = 6.0$ with a p-value of 0.0106. In the case of the bayg29 problem, significance was achieved for $\kappa_1 = 0.0$ against $\kappa_2 = 3.0$ and $\kappa_2 = 6.0$ with p-values of 0.0177 and 0.0362, respectively.

Discussion of Parallel Results

From the results of the experiments, it is clear that the best overall performance of the system in terms of accuracy and speed is achieved when $\kappa = 0.0$. This result is somewhat surprising, for it was hypothesized before the experiment that low values of κ would lead to an unfocused search space and sub-optimal solutions.

One possible explanation for the exhibited behavior is the relatively small problem size of both the bayg29 and swiss42 problems. It may be that the solution spaces of both problems are small enough for the underlying algorithm, regardless of the κ-value, to fully explore and determine the optimal tour using only locally-best tour information. This is a plausible theory, for the algorithm upon which this parallel system was based is the *MAX-MIN* algorithm, which has outperformed several other ant algorithms when applied to the Traveling Salesman Problem [16]. It would be a worthwhile endeavor to repeat the experiment on larger problem instances such as d198, att532, and rat738 [14] and observe the optimal κ-values that emerge. With an exponentially larger solution space, global information could play an increasingly important role in tour construction.

Comparison to a Sequential Approach

Figures 3 and 4 compare the average tour length found by the parallel algorithm with κ = 0.0 to the average tour length found by a similar sequential algorithm for both the bayg29 and swiss42problems.

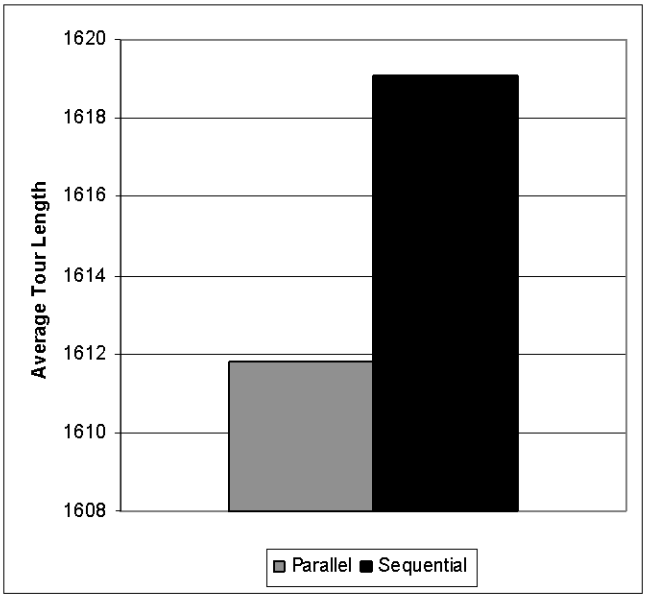

Figure 3: Comparison of Average Tour Lengths Found by Parallel and Sequential Algorithms - bayg29

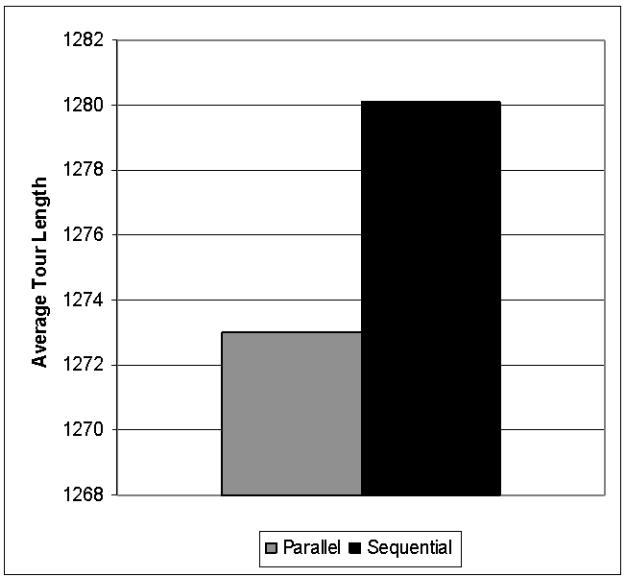

Figure 4: Comparison of Average Tour Lengths Found by Parallel and Sequential Algorithms - swiss42

As shown in figures 3 and 4, the parallel setup found shorter tours than the similar sequential setup. Using the Mann-Whitney test for significance, the performance of the

parallel algorithm for the bayg29 problem in terms of the average tour length found is significantly better than the performance of the sequential algorithm, with a p-value of 0.0009. The tour lengths of the swiss42 parallel runs, while on the average shorter than the tour lengths of the sequential runs, were not able to achieve statistical significance.

Additionally, the parallel algorithm outperformed its sequential counterpart in terms of number of optimal solutions found. The sequential approach found the optimal bayg29 tour 0 out of 20 times and the optimal swiss42 tour 16 out of 20 times. This compares to success rates of 17/20 and 20/20, respectively, for the parallel algorithm.

Increased performance by the parallel algorithm is not surprising, for the addition of three other machines working in parallel allows for a more through coverage of the solution space in approximately the same amount of time. The separation of the global-best tour from the local-best tour (which does not occur in the sequential algorithm) also helped the parallel instances maintain a higher degree of autonomy, which encouraged diversity, thus producing better solutions.

Conclusion

This paper introduced the ideas of Swarm Intelligence and ant algorithms and has shown how these innovative models of problem solving taken from the natural world can be applied to problems in the field of computer science. Application of these ideas to one particular problem, the Traveling Salesman Problem, was the main focus of this paper. A sequential algorithm was introduced and then refined into a more powerful parallel model. Design and implementation considerations were discussed, including the choice of Java and JPVM for the programming language and message-passing environment, respectively.

Information exchange, and more specifically, trail update pressure, in a parallel system was studied. Results, which favored low κ-values, were surprising and warrant further investigation.

Comparisons between parallel and sequential implementations were also made, and the parallel system was shown to be superior to the sequential system in all measurable aspects.

Possible directions for future research in parallel ant algorithms include additional study into κ-values for larger problem sets, as well as variable κ-values that adapt to the amount of variation in the current search space. A broader look at inter-colony communication in general, especially dynamic reporting of significant gains within one colony to the global environment, would also be an interesting topic.

References

[1] Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Oxford: Oxford University Press.

[2] Bullnheimer, B., Kotsis, G., & Strauss, C. (1997). *Parallelization Strategies for the Ant System*. Technical report POM 9/97. Vienna: University of Vienna.

[3] Colorni, A., Dorigo, M., & Maniezzo, V. (1991). Distributed Optimization by Ant Colonies. In *Proceedings of ECAL91*, Elsevier Publishing, 132-142.

[4] Colorni, A., Dorigo, M., & Maniezzo, V. (1992). An Investigation of some properties of an "Ant algorithm." In *Proceedings of PPSN92*, Elsevier Publishing, 509-520.

[5] Dorigo, M., Bonabeau, E., & Theraulaz, G. (2000). Ant algorithms and stimergy. *Future Generation Computer Systems, 16*, 851-871.

[6] Dorigo, M., & Gambardella, L.M. (1997). Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation, 1*, 53-66.

[7] Dorigo, M., Maniezzo, V., & Colorni, A. (1991). *Positive Feedback as a Search Strategy*. Technical report n. 91-016. Milan: Department of Electronics, Milan Polytechnic Institute.

[8] Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics, 26*, 29-41.

[9] Ferrari, A. (1999). JPVM: The Java Parallel Virtual Machine. Retrieved November 1, 2001, from http://www.cs.virginia.edu/~ajf2j/jpvm.html

[10] Freisleben, B. & Merz, P. (1996). Genetic local search algorithms for solving symmetric and asymmetric traveling salesman problems. In *Proceedings of IEEE International Conference on Evolutionary Computation, IEEE-EC '96*, 616-621.

[11] Krohn, J. (2001). Ant Algorithms and the Swarm Intelligence Model of Problem Solving. In *Proceedings of UMM Computer Science Discipline Seminar Conference*, Morris: University of Minnesota.

[12] Lin, S., & Kernighan, B.W. (1973). An Effective Heuristic Algorithm for the TSP. *Operations Research, 21*, 498-516.

[13] Middendorf, M., Reischle, F., & Schmeck, H. (2000). Information Exchange in Multi Colony Ant Algorithms. In *Proceedings of IEEE Symposium on Parallel and Distributed Processing, Third Workshop on Biologically Inspired Solutions to Parallel Processing Problems, IPDPS 2000*, 645-652.

[14] Reinelt, G. (2001). *TSPLIB: Traveling Salesman Library*. Retrieved December 1, 2001, from http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

[15] Stützle, T. (1998). *Parallelization Strategies for Ant Colony Optimization.* Research report AIDA-98-03. Darmstadt: Department of Computer Science, Darmstadt University of Technology.

[16] Stützle, T., & Hoos, H.H. (2000). *MAX-MIN* Ant System. *Future Generation Computer Systems, 16*, 889-914.

[17] Sunderam, V.M. (1990). PVM: A Framework for Parallel Distributed Computing. *Concurrency: Practice and Experience, 2*, 315-339.

Acknowledgements

The author would like to thank Dr. Dian Lopez of the University of Minnesota, Morris, for her assistance with the preparation of this paper.