# Parallel Shared Memory Strategies for Ant-Based Optimization Algorithms

Thang N. Bui
Penn State Harrisburg
tbui@psu.edu

ThanhVu Nguyen
University of New Mexico
tnguyen@cs.unm.edu

Joseph R. Rizzo Jr.
Concurrent Technologies
Corporation
rizzoj@ctc.com

## ABSTRACT

This paper describes a general scheme to convert sequential ant-based algorithms into parallel shared memory algorithms. The scheme is applied to an ant-based algorithm for the maximum clique problem. Extensive experimental results indicate that the parallel version provides noticeable improvements to the running time while maintaining comparable solution quality to that of the sequential version.

## Categories and Subject Descriptors

G.2.2 [**Discrete Mathematics**]: Graph Theory, Graph algorithms; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search, Heuristic methods

## General Terms

Algorithms, Design

## Keywords

Ant-Based Algorithms, Distributed Memory, Shared Memory, OpenMP, MPI, Max Clique

## 1. INTRODUCTION

Many real world applications bear resemblances to $\mathcal{NP}$-hard combinatorial problems. Hence, unless $\mathcal{P} = \mathcal{NP}$, they most likely have no algorithms that give exact or even good approximate solutions in acceptable time. Alternative strategies such as ant algorithms, algorithms that mimic the behavior of ants, have been shown to be successful in these situations, even though they do not guarantee solution quality. However, in many cases ant algorithms still require large computing resources and time, especially when the problem size increases. The nature of these ant algorithms makes them good candidates for parallelization and several works have been done in that direction, particularly under the distributed model of computation. The recent availability and

affordability of multi-core processors [15, 22] provide incentive for studying parallel implementations of ant algorithms, particularly in the shared memory model.

Dorigo first introduced ant systems in [10], which were later formalized as the Ant Colony Optimization (ACO) metaheuristics in [12]. Since then many variants of ACO have been proposed and used successfully to solve a wide variety of problems [12, 13, 19]. In ACO and its variants a sequence of ants is used to solve the problem at hand with later ants using information gathered by previous ants to help find a good solution. A different approach using ants was proposed in [4, 5, 6, 7] to solve various graph optimization problems. We will call this latter type of ant algorithms Ant-Based Optimization (ABO) algorithms. In ABO algorithms ants are used to identify an area of the search space that potentially contains a good solution, effectively reducing the search space. A local optimization algorithm is then used to search for a solution in this reduced search space.

In this paper, we first describe a skeleton for ABO algorithms, specifically for graph problems. We then describe the process of transforming this sequential skeleton algorithm into a parallel algorithm in the shared memory model. The technique is then applied to the sequential ABO algorithm for the Max-Clique problem of [4]. The resulting parallel algorithm was implemented and tested on a test set of 120 instances of the Max-Clique problem using machines with multi-core processors. Experimental results show that the parallel algorithm provides improvements in the running time of up to a factor of 6 using an 8-core processor, while maintaining the quality of the solutions. We believe that this conversion scheme can be effectively applied to ABO algorithms for other problems.

The rest of the paper is organized as follows. In Section 2 we give background information on ant-based algorithms, an overview of parallel computing, and our test bed Max-Clique problem. In Section 3 we present a generic sequential ABO algorithm, describe an equivalent algorithm in the shared memory model, and apply this conversion process to a sequential ant-based algorithm for the Max-Clique problem. Section 4 shows our experimental results. Finally, our conclusion and suggestions for future work are given in Section 5.

## 2. BACKGROUND

### 2.1 Ant-Based Algorithms

*Ant Colony Optimization* (ACO) is a metaheuristic that imitates the collective behavior of ants to solve problems [11].

In an ACO algorithm, ants take turns at solving the problem. Each ant solves the entire problem by itself based on rules that are applicable to all ants as well as information left by previous ants. Ants communicate with other ants in the algorithm by using pheromone [3]. Generally, once an ant has found a solution it leaves pheromone on the problem structure corresponding to the solution in such a way that the better the solution is the more pheromone is laid down. Other strategies include dividing the algorithm into cycles and allowing only the ant with the best solution in each cycle to lay down pheromone. Pheromone is also evaporated to mitigate the chance of ants being stuck at a locally optimal solution. ACO and its variants have proved to be effective for various problems including the traveling salesman, quadratic assignment, routing, and knapsack problems [12, 13, 19].

More recently, another type of ant inspired algorithm has been used effectively to solve graph problems such as the graph bisection, $k$-cardinality tree, degree-constrained spanning tree, and maximum clique problems [4, 5, 6, 7]. We will refer to this type of algorithms as *Ant-Based Optimization* (ABO) algorithms. In an ABO algorithm, ants are used to identify a region of the search space that might contain a good solution. A local optimization algorithm is then used to find a solution in this region. More specifically, ants are distributed on the structure of the problem instance, e.g., the vertices of a graph. The distribution of ants is done randomly or based on the result of some quick heuristics. Each ant then explores the structure of the problem instance, e.g., traversing from vertices to vertices. The exploration process as well as the pheromone placement strategy require each ant to "see" only the parts of the problem structure closest to it. Furthermore, all ants perform the exploration in parallel. In a sequential ABO algorithm this is accomplished by having the ants perform the exploration in lock-step. For example, the exploration is divided into a number of steps and all ants perform one step before another step is taken. When the exploration process terminates the pheromone and the positions of the ants are used to determine a potential solution or a candidate set that contains a potential solution.

The main difference between an ACO algorithm and an ABO algorithm is in the way ants are used. In an ACO algorithm, each ant produces a solution to the problem. In an ABO algorithm each ant does not solve the entire problem, rather all the ants collectively identify regions of the search space that may contain a good solution.

## 2.2 Shared and Distributed Memory Models

Shared and distributed memory are two prevalent interprocessor communication systems in parallel computing [23]. Systems using shared memory (SM), such as symmetric multiprocessor (SMP), allow a collection of identical processors to share main memory via the system bus (or crossbar for more than four processors) [1]. Distributed memory (DM) systems, such as Beowulf clusters, are composed of multiple stand-alone machines, each with its own processor and memory set [30]. These machines can be heterogeneous and communicate with one another by means of passing messages through a high-speed network. POSIX threads and OpenMP (Multi-Processing) are two standard shared memory Application Programming Interfaces (APIs) and Message Passing Interface (MPI) is the traditional API in message passing systems [31, 32].

Both shared and distributed memory models have advantages and disadvantages. Porting a sequential program to a SM system using APIs such as OpenMP often requires adding parallelism to appropriate sections of the sequential code. However, race conditions, deadlocks, and other problems associated with shared access might occur. Writing message passing programs in a DM system is more complicated; it typically involves designing efficient algorithms to divide tasks among processors with separate memory spaces. Moreover, processes coordination, data synchronizations, and network latency are common challenges. The major advantage of DM systems over SM systems is scalability. Adding more processors to a DM system is as simple as attaching new computers to the current environment, whereas doing so in a SM setting increases the bus traffic on the system, slows down memory access time, and possibly requires expensive and complex changes to the current hardware configuration, e.g., different motherboard design to support more processors. Some of these problems are alleviated in a multi-core system, which has cache sharing at various levels [25].

## 2.3 Previous Work

Parallel ant algorithms often fall under the classes of parallel ants [8, 24, 29], parallel ant colonies [8, 14, 18, 20], or hybridization of various parallel techniques [1, 14, 21, 28]. In the parallel ants approach, each ant occupies a separate processor and sends pheromone updates to others at each step of the algorithm. In the parallel ant colonies approach, each processor houses an ant colony. The program periodically broadcasts the pheromone structure from the best performing colony to others. The hybrid model combines parallel ant or parallel ant colonies with different strategies for exchanging information. For instance, the processors may be arranged in a ring, star, or hypercube topology and may exchange pheromone and solution information in such a way that each processor only communicates with its neighbors.

An OpenMP based model creates multiple data structures (e.g., solution trees, pheromone matrix) and assigns each to a thread (processor) [9]. These threads work independently on their assigned data structures and merge them when needed. However there are potential issues with managing data structures with dynamically allocated values such as inserting and deleting nodes from trees.

With the exception of [9], the above approaches assume the more popular distributed memory model rather than the shared memory one. In addition, the popular ACO based algorithms are used in most of these studies.

## 2.4 The Max-Clique Problem

A *clique* is a complete subgraph in an undirected graph, i.e., a clique is a subgraph in which there is an edge between any two vertices in the subgraph. The *size* of a clique is the number of vertices in the clique. The *Max-Clique* optimization problem is the problem of finding the largest clique in a graph. Max-Clique arises in problems such as finding good codes, identifying faulty processors in multi-processor systems, and finding counterexamples to Keller's conjecture in geometry [2, 17, 26, 27].

It is known that Max-Clique is $\mathcal{NP}$-hard. Furthermore, it has been shown that even finding a good approximation is difficult for the Max-Clique problem. For instance, it is

---

[1] We consider systems with chip-level muti-core processors to belong to the SM model albeit with some minor differences.

known that unless $\mathcal{P} = \mathcal{NP}$ no polynomial time algorithm can achieve an approximation factor of $n^{1-\epsilon}$ for Max-Clique for arbitrarily small $\epsilon > 0$ [16]. Thus, in practice, heuristics such as ant-based algorithms are often used.

# 3. ALGORITHMS

In this section we provide a generic (sequential) ant-based optimization algorithm, specifically for graph problems. We then describe a shared memory parallel version for it. Finally, we show how this conversion scheme is applied to the ABO algorithm for Max-Clique (ABOMC) of [4].

## 3.1 Sequential ABO Algorithm

The main ideas of a generic ABO algorithm are given in Figure 1. The algorithm distributes ants on the graph based on an initial solution, usually found by a simple and fast heuristic. Typically, more ants are placed on locations that are occupied by the initial solution in order to start the algorithm at a local optimum. As ants move, they lay pheromone to attract other ants. The amount of pheromone stored in a region increases as more ants explore that region. Periodically the algorithm constructs a configuration, e.g., a set of vertices and/or edges, from the graph using the locations of the ants and pheromone levels. A local optimization algorithm is then used to extract a solution from this configuration. The best solution found is kept and returned when the algorithm terminates.

Analogous to evolution algorithms, ABO uses ants and their pheromone production to narrow the search space, i.e., to identify promising regions of the search space. To avoid areas with many ants which could potentially lead to local optima, ABO periodically evaporates pheromone and perturbs the distribution of ants. This lets ants better explore the search space.

---

**Algorithm ABO$_{\text{SEQ}}$**$(G = (V, E))$

construct a feasible configuration $C$ by using
   a quick heuristic method
distribute ants on $G$ based on $C$

**repeat**
  **for** each ant $\alpha$ **do**
    $\alpha$ performs some tasks, moves to another location
      on the graph, and lays pheromone
  **end-for**

  construct a new configuration $C'$ based on
    ant locations & pheromone information
  use some local optimization algorithm to extract
    a solution from $C'$
  perturb ant distribution on $G$
**until** some terminating criteria are met

**return** the best solution found

---

**Figure 1: Generic Sequential ABO Algorithm**

## 3.2 Parallel Shared Memory ABO Algorithm

The common programming paradigm for shared memory system is the *fork-join* model. A single *master* thread is created when the program starts, executed sequentially, and

---

**Algorithm ABO$_{\text{SM}}$**$(G = (V, E))$

*//Candidate region for parallelism*
find a starting configuration $C$ in $G$.

*//Sequential region*
distribute ants on $G$ based on the initial configuration $C$

**repeat**
  *//Candidate region for parallelism*
  **for** each ant **do**
    ant does the work in parallel where applicable
  **end-for**

  *//Candidate region for parallelism*
  use several local optimization techniques/instances
    to construct a configuration based on ants' locations.

  *//Sequential region*
  perturb ants distribution on $G$
**until** some terminating criteria are met

**return** the best solution found

---

**Figure 2: Generic Shared Memory Model for ABO**

expanded into multiple *slave* threads when parallelism is required. These threads work concurrently through the parallel region then merge back to the single master thread when they are done.

To use the *fork-join* model, parallel regions from the sequential code need to be specified. Figure 2 suggests several sections in ABO$_{\text{SEQ}}$ that are potentially efficient and safe for parallelism, i.e., which reduce the execution time while not degrading the solution quality. The work done by ants in the ABO algorithm's core seems well suited for parallelization. However, the main challenge of parallelization is to continue to reduce the execution time as the number of processors (or cores) increases while maintaining or improving on the quality of solutions.

The heuristic function that generates a starting configuration for the ants and the local optimization are potential candidates for parallelism. Instead of generating one starting configuration, several instances of the same or different heuristic algorithms can run in parallel producing multiple configurations, of which the best one is selected. Similarly, different optimization algorithms can run concurrently to achieve the best configuration. In addition, the same heuristic method can run with different parameter settings or searching techniques, e.g., different selection criteria. A variety of searching techniques might help ABO perform better on different types of graphs or problem parameters. We note that choosing the best of multiple runs is a greedy-based approach and therefore might cause the search to be trapped at local optima.

It is also advantageous to apply techniques from parallel ant colonies with shared memory. The colonies can be run in parallel on multiple cores or processors while exchanging information as necessary. While this technique does not decrease the running time of the ants working in a colony, it might help the quality of the solution as the search space is explored by multiple ant colonies.

## 3.3 A Parallel Shared Memory Algorithm for Max-Clique

For convenience we describe the Max-Clique algorithm of [4] in Figure 3, which we refer to as ABOMC$_{\text{SEQ}}$. It is easily seen that this algorithm fits into the model of the generic sequential ABO algorithm given in Figure 1. It starts with a greedy method to quickly find a clique. A large fraction of the ants are then distributed at random on vertices of the clique. The remainder of the ants are distributed randomly on the vertices of the graph. The algorithm then goes through a number of stages. Each stage consists of a number of cycles and at the end of each stage a clique is constructed. In each cycle a fraction of the ants are selected to move and they do so with certain probability. Their destinations are determined by various factors such as pheromone concentration and structure of the neighborhood around the ants. As an ant traverses an edge of the graph it also puts down a certain amount of pheromone on that edge. To allow for more exploration and possible escape from local optima, pheromone evaporates over time. In addition to using pheromone as a means of communication, ants in ABOMC$_{\text{SEQ}}$ also use their positions to communicate. For example, in computing a destination to move to an ant prefers vertices that are currently occupied by a large number of ants. At the end of each stage, a local optimization algorithm is used to construct a set of candidate vertices based on the current distribution of the ants and the pheromone levels on the edges. The algorithm then uses a greedy algorithm to extract a clique from the set of candidate vertices. This clique is then further grown, if possible, using vertices not in the candidate set. A small portion of ants is then shuffled around the graph before moving on to the next stage. After finishing all stages, the algorithm returns the largest clique found in all stages.

Techniques such as pheromone evaporation and ants shuffling are used to mitigate problems caused by premature convergence and local optima. Moreover, adaptive behaviors are given to the ants to encourage a gradual transition from exploration in the beginning to exploitation near the end. In early stages, young ants are more active and rely less on pheromone and more on the structure of the graph. As the algorithm progresses, ants get older and are less likely to move and they make more use of pheromone in determining their movement. More details can be found in [4].

We are now ready to apply the strategies of Section 3.2 to ABOMC$_{\text{SEQ}}$ to obtain a parallel shared memory ABO algorithm for Max-Clique, called ABOMC$_{\text{SM}}$. First we run multiple instances of the greedy algorithm in parallel and use the largest clique found as the starting point for the ants. Similarly we run multiple versions of the local optimization algorithm (and the greedy algorithm) in parallel to construct the largest clique after each stage. Since these heuristic algorithms contain random choices, the clique returned by one run is most likely different from another, thus allowing a more aggressive starting configuration or a larger clique at the end of each stage. Note that for small graphs we decided not to run these functions in parallel since preliminary experimental results indicate that there is no noticeable improvement in solution quality but the running time is increased due to additional communication overhead.

Figure 4 shows the parallel work of the ants in the parallel algorithm, ABOMC$_{\text{SM}}$. The *stage* and *cycle* loops cannot be parallelized since each stage requires information from

---

**Algorithm ABOMC$_{\text{SEQ}}$$(G = (V, E))$**

use a greedy method to get a clique $C$ in $G$
distribute ants on the vertices of $G$ such that
   vertices in $C$ have a higher ant concentration
**for** $s = 1$ **to** $nStages$ **do**
  **for** $i = 1$ **to** $nCycles$ **do**
    **for** $\alpha = 1$ **to** $nAnts$ **do**
      ant $\alpha$ decides where to move based on
        local information
      ant $\alpha$ moves to its new destination
        and deposits pheromone on edges along the way
      evaporate a small amount of pheromone from areas
        close to ant $\alpha$'s previous location
    **end-for**
  **end-for**
  use a local optimization algorithm to find a candidate
    set $C'$ of vertices based on ant locations & pheromone
  use a greedy algorithm to find a clique $K$ in $C'$
  grow $K$
  shuffle ants
**end-for**
**return** the largest clique found

**Figure 3: Sequential ABO Algorithm for Max-Clique**

previous stage and an ant can only perform an action (movement) per time cycle. We then designate the *ant* section as the main parallel region since parallelizing this part would least likely affect the quality of the results. Moreover, the size of the ant colony is approximately six times the number of vertices of the input graph. ABO algorithms often use a much larger ant population than traditional ACO. In a typical ACO algorithm each ant solves the entire problem and hence the number of ants is in some sense the number of iterations. On the other hand, in a typical ABO algorithm, ants are used aggregately as a means to explore and reduce the search space. Consequently, the work of the ants consumes more than half of the total time execution in ABOMC$_{\text{SEQ}}$. [2].

In each cycle each ant determines where to move and moves there. These activities are expensive since the ant examines the surrounding neighborhood to decide its next destination, deposits pheromone on the edges along the way, and updates its new position. In the sequential version, an ant's action influences other ants, even when they occur in the same time cycle. We redesign this part by separating it into two operations and parallelize them both as illustrated in Figure 4. This refinement allows the ants to make decision based on the current configuration independently of other ants in the same time cycle. Hence, in addition to speed gain, the revision provides better imitation of the inherent parallelism in natural ant behavior.

When an ant leaves a vertex, a small amount of pheromone is evaporated from edges in the surrounding region. This evaporation procedure is time costly since the algorithm checks whether pheromone on the edges adjacent to the ant has already been evaporated. Moreover, this part cannot be parallelized since the pheromone on each edge is evaporated at most once per cycle.

---

[2]The GNU profile utility *gprof* was used here.

---
**Algorithm ABOMC$_{\mathbf{SM}}$**$(G = (V, E))$

*//Parallel region*
find an initial clique $C$ by running multiple instances of
   the same greedy algorithm with different seeds
   and keeping the largest clique
*//Sequential region*
distribute ants on $G$ with heavier concentration on $C$
**for** $s = 1$ **to** $nStages$
  **for** $i = 1$ **to** $nCycles$ **do**
    *//Parallel region*: ants make decisions
    **for** $\alpha = 1$ **to** $nAnts$ **do**
      ant $\alpha$ decides where to move based on
        local information
    **end-for**
    *//Parallel region*: ants move
    **for** $\alpha = 1$ **to** $nAnts$ **do**
      ant $\alpha$ moves to its new destination
        and deposits pheromone on edges along the way
        areas closed to ant $\alpha$'s previous location are
          marked for pheromone evaporation
    **end-for**
    *//Sequential region*
    evaporate small amount of pheromone from
      the marked areas
  **end-for** *//nCycles*

  *//Parallel region*
  run multiple instances of local optimization & greedy
    algorithms to construct candidate sets & cliques
  *//Sequential region*
  select the best clique $K$ from the cliques found above
  grow $K$
  perturb ants distribution on $G$
**end-for** *//nStages*
**return** the largest clique found
---

**Figure 4: Parallel ABO Algorithm for Max-Clique**

We resolve this situation by selecting the edges for phero-
mone evaporation in parallel and performing the actual evap-
oration sequentially. A shared Boolean array of size equal
to the number of edges is created with everything initial-
ized to *false*. When the pheromone on an edge is updated,
the corresponding value of that edge in the array is marked
as *true*. Hence, even if simultaneous updates occur on an
edge, its corresponding value in the shared Boolean array
remains the same[3]. After all the ants move, the pheromone
on the array's marked edges is sequentially updated. We
could parallelize this step; however, experiments show that
no significant improvement is obtained by doing so.

Many of the loop sections in the sequential code can be
executed in parallel with the *fork-join* model; however, not
all tasks make good choices for optimization. Low com-
plexity tasks, such as shuffling ants, may worsen the overall
running time as the overhead caused by forking and joining
threads surpasses the speed gained from parallel process-
ing. Nonetheless, even without speed improvement, in some
cases parallelism enhances the solution quality. Instead of
parallelizing the already efficient greedy and local optimiza-

---
[3]Alternatively we can create multiple copies of this array,
assign each to a parallel thread, and merge the results for
the sequential update.

tion algorithms, we independently process them in parallel
and choose the best answers. For example, at the end of
each stage, we run multiple instances of a local optimiza-
tion/greedy combination to find candidate sets and cliques,
using different selection criteria in each instance. We select
the largest clique from these cliques and then grow it. Our
experiments show on average the algorithm achieves good
solutions in fewer iterations when combined with these tech-
niques.

## 4. EXPERIMENTAL RESULTS

We implemented ABOMC$_{\mathrm{SM}}$ in C++ and compiled it us-
ing GNU gcc with OpenMP support for our shared memory.
The OpenMP API consists of compiler directives and library
functions to help compilers execute instructions such as fork-
ing and joining threads [32]. The program was tested on a
computer running Linux with an 8-core Intel Xeon 2.8 GHz
and 16 GB memory.

We test our algorithm on 120 benchmark graphs from [4,
33, 34]. Due to space limitations, only 60 of those instances
are presented here (supplemental results are available in [35]).
Table 1 shows the solution quality of ABOMC$_{\mathrm{SM}}$. For each
benchmark graph and for each setting of the number of cores
$c$, where $c = 1, 2, 4, 6, 8$, we ran our algorithm for 100 trials.
For each instance and for each core setting, we list the best
solution found by ABOMC$_{\mathrm{SM}}$, the average (Avg), and the
standard deviation (StdDev) of the results. We also list the
optimal or best known clique size for each instance (Opt).
As can be seen from Table 1, the parallel results are compa-
rable to the sequential results ($c = 1$).

For $c \in \{2, 4, 6, 8\}$, let $IF(c) = T_1/T_c$, where $T_1$ is the
running time with one core, and $T_c$ is the running time with
$c$ cores. We think of $IF(c)$ as an *improvement factor* in the
running time. The average running time, $T_{\mathrm{avg}}$, and improve-
ment factor in the running time are given in Table 2. Note
that in most cases simple graphs such as `johnson8-2-4` did
not benefit from parallelism. This is as expected since the
overhead caused by the fork-join operations dominate the
program's short running times (a fraction of a second). For
medium size graphs such as `c-fat500-5`, the running time is
improved by a factor of approximately $1.5, 2.4, 2.9, 3.3$ with
$2, 4, 6, 8$ cores, respectively. On larger and more complicated
graphs like `keller-6` the running time is improved by a fac-
tor of approximately $1.9, 3.6, 5, 6.1$ with $2, 4, 6, 8$ cores, re-
spectively. Table 3 shows the 95% confidence interval, ob-
tained using nonparametric bootstrap, for the improvement
factor $IF(c)$ for graphs of various sizes.

## 5. CONCLUSION

In this paper we presented a shared memory parallel frame-
work for general ABO approach and used an existing ABO
algorithm for the Max-Clique problem as a testbed. We
maintained the solution quality of the sequential ABO al-
gorithm while achieving noticeable running time improve-
ments. From experimental results, we found that the run-
ning times are improved around 40% with dual-core system
and up to six times faster with an eight-core system. Given
the increasing prevalence of multi-core processors and the
extent to which the ABO approach lends itself to parallelism,
we believe the technique demonstrated here could be applied
to other ABO algorithms as well as other problem domains.
Our future plans include constructing a hybrid distributed-

Table 1: ABOMC$_{\text{SM}}$ Solution Quality

| Graph | Opt | ABOMC Best | Solution Avg/StdDev when run on $c$ cores | | | | |
|---|---|---|---|---|---|---|---|
| | | | $c = 1$(Seq) | $c = 2$ | $c = 4$ | $c = 6$ | $c = 8$ |
| brock200_1 | 21 | 21 | 19.53 / 0.52 | 19.57 / 0.57 | 19.59 / 0.55 | 19.68 / 0.53 | 19.59 / 0.51 |
| brock200_2 | 12 | 11 | 10.03 / 0.43 | 10.04 / 0.31 | 10.09 / 0.32 | 10.07 / 0.38 | 10.09 / 0.35 |
| brock400_1 | 27 | 24 | 22.74 / 0.69 | 22.68 / 0.65 | 22.72 / 0.69 | 22.83 / 0.63 | 22.83 / 0.63 |
| brock400_2 | 29 | 25 | 22.69 / 0.72 | 22.84 / 0.61 | 22.83 / 0.62 | 22.85 / 0.65 | 22.79 / 0.67 |
| brock800_1 | 23 | 20 | 18.52 / 0.64 | 18.55 / 0.61 | 18.64 / 0.59 | 18.59 / 0.57 | 18.67 / 0.63 |
| brock800_2 | 24 | 21 | 18.50 / 0.64 | 18.68 / 0.53 | 18.64 / 0.59 | 18.74 / 0.61 | 18.80 / 0.53 |
| c-fat500-1 | 14 | 14 | 14.00 / 0.00 | 14.00 / 0.00 | 14.00 / 0.00 | 14.00 / 0.00 | 14.00 / 0.00 |
| c-fat500-5 | 64 | 64 | 64.00 / 0.00 | 64.00 / 0.00 | 64.00 / 0.00 | 64.00 / 0.00 | 64.00 / 0.00 |
| c-fat500-10 | 126 | 126 | 126.00 / 0.00 | 126.00 / 0.00 | 126.00 / 0.00 | 126.00 / 0.00 | 126.00 / 0.00 |
| c250.9 | $\geq 44$ | 44 | 41.16 / 0.98 | 41.01 / 0.94 | 41.08 / 1.06 | 41.33 / 1.13 | 41.07 / 0.85 |
| c500.9 | $\geq 57$ | 55 | 50.97 / 1.09 | 51.19 / 1.19 | 51.27 / 1.08 | 51.37 / 1.07 | 51.26 / 1.23 |
| c1000.9 | $\geq 68$ | 63 | 58.98 / 1.22 | 59.30 / 1.13 | 59.43 / 1.12 | 59.50 / 1.29 | 59.57 / 1.10 |
| c2000.5 | $\geq 16$ | 16 | 13.98 / 0.55 | 14.09 / 0.38 | 14.16 / 0.46 | 14.18 / 0.46 | 14.17 / 0.43 |
| c2000.9 | $\geq 77$ | 71 | 66.32 / 1.17 | 66.33 / 0.98 | 66.54 / 1.09 | 66.80 / 1.21 | 66.84 / 1.07 |
| c4000.5 | $\geq 18$ | 16 | 14.95 / 0.50 | 15.15 / 0.48 | 15.00 / 0.32 | 15.10 / 0.30 | 15.10 / 0.44 |
| dsjc500.5 | $\geq 13$ | 13 | 11.91 / 0.53 | 12.05 / 0.55 | 12.07 / 0.50 | 12.10 / 0.48 | 12.05 / 0.38 |
| dsjc1000.5 | $\geq 15$ | 15 | 13.05 / 0.50 | 13.10 / 0.44 | 13.14 / 0.51 | 13.16 / 0.39 | 13.16 / 0.44 |
| gen200_p0.9_44 | 44 | 40 | 37.23 / 1.04 | 37.21 / 0.90 | 37.47 / 0.83 | 37.46 / 1.02 | 37.22 / 0.78 |
| gen200_p0.9_55 | 55 | 55 | 42.35 / 5.31 | 41.45 / 4.56 | 41.89 / 4.83 | 42.06 / 5.02 | 41.51 / 4.55 |
| gen400_p0.9_55 | 55 | 51 | 48.33 / 0.91 | 48.64 / 1.04 | 48.59 / 0.97 | 48.35 / 0.99 | 48.45 / 0.97 |
| gen400_p0.9_65 | 65 | 65 | 47.86 / 3.52 | 47.37 / 2.16 | 47.91 / 3.28 | 48.46 / 4.12 | 47.78 / 2.93 |
| gen400_p0.9_75 | 75 | 75 | 51.69 / 7.22 | 54.27 / 9.55 | 51.85 / 7.34 | 52.73 / 8.68 | 51.38 / 6.89 |
| hamming6-2 | 32 | 32 | 32.00 / 0.00 | 32.00 / 0.00 | 32.00 / 0.00 | 32.00 / 0.00 | 32.00 / 0.00 |
| hamming6-4 | 4 | 4 | 4.00 / 0.00 | 4.00 / 0.00 | 4.00 / 0.00 | 4.00 / 0.00 | 4.00 / 0.00 |
| hamming8-2 | 128 | 128 | 128.00 / 0.00 | 128.00 / 0.00 | 128.00 / 0.00 | 128.00 / 0.00 | 128.00 / 0.00 |
| hamming8-4 | 16 | 16 | 16.00 / 0.00 | 16.00 / 0.00 | 16.00 / 0.00 | 16.00 / 0.00 | 16.00 / 0.00 |
| hamming10-2 | 512 | 512 | 512.00 / 0.00 | 512.00 / 0.00 | 512.00 / 0.00 | 512.00 / 0.00 | 512.00 / 0.00 |
| hamming10-4 | 40 | 40 | 34.92 / 0.91 | 35.00 / 0.95 | 35.34 / 1.12 | 35.48 / 0.87 | 35.50 / 1.10 |
| johnson8-2-4 | 4 | 4 | 4.00 / 0.00 | 4.00 / 0.00 | 4.00 / 0.00 | 4.00 / 0.00 | 4.00 / 0.00 |
| johnson8-4-4 | 14 | 14 | 14.00 / 0.00 | 14.00 / 0.00 | 14.00 / 0.00 | 14.00 / 0.00 | 14.00 / 0.00 |
| johnson16-2-4 | 8 | 8 | 8.00 / 0.00 | 8.00 / 0.00 | 8.00 / 0.00 | 8.00 / 0.00 | 8.00 / 0.00 |
| johnson32-2-4 | 16 | 16 | 16.00 / 0.00 | 16.00 / 0.00 | 16.00 / 0.00 | 16.00 / 0.00 | 16.00 / 0.00 |
| keller4 | 11 | 11 | 10.11 / 0.85 | 9.94 / 0.87 | 10.17 / 0.91 | 10.02 / 0.87 | 10.14 / 0.84 |
| keller5 | 27 | 25 | 21.46 / 0.85 | 21.52 / 1.04 | 21.58 / 0.86 | 21.81 / 0.89 | 21.71 / 0.96 |
| keller6 | $\geq 59$ | 48 | 43.27 / 1.45 | 43.19 / 1.47 | 43.74 / 1.30 | 43.71 / 1.34 | 43.57 / 1.31 |
| mann_a9 | 16 | 16 | 16.00 / 0.00 | 16.00 / 0.00 | 16.00 / 0.00 | 16.00 / 0.00 | 16.00 / 0.00 |
| mann_a27 | 126 | 126 | 125.00 / 0.00 | 125.00 / 0.00 | 125.01 / 0.10 | 125.00 / 0.00 | 125.00 / 0.00 |
| mann_a45 | 345 | 342 | 342.00 / 0.00 | 342.00 / 0.00 | 342.00 / 0.00 | 342.00 / 0.00 | 342.00 / 0.00 |
| mann_a81 | $\geq 1100$ | 1096 | 1096.00 / 0.00 | 1096.00 / 0.00 | 1096.00 / 0.00 | 1096.00 / 0.00 | 1096.00 / 0.00 |
| p_hat500-2 | 36 | 36 | 35.94 / 0.24 | 35.93 / 0.26 | 35.93 / 0.26 | 35.98 / 0.14 | 35.97 / 0.17 |
| p_hat500-3 | $\geq 50$ | 50 | 49.00 / 0.53 | 48.98 / 0.49 | 49.08 / 0.52 | 49.00 / 0.49 | 49.13 / 0.48 |
| p_hat700-2 | $\geq 44$ | 44 | 43.62 / 0.49 | 43.67 / 0.47 | 43.71 / 0.45 | 43.62 / 0.49 | 43.65 / 0.48 |
| p_hat700-3 | $\geq 62$ | 62 | 61.24 / 0.53 | 61.28 / 0.51 | 61.33 / 0.47 | 61.41 / 0.53 | 61.38 / 0.49 |
| p_hat1000-2 | $\geq 46$ | 46 | 45.20 / 0.49 | 45.29 / 0.57 | 45.34 / 0.55 | 45.31 / 0.54 | 45.30 / 0.46 |
| p_hat1000-3 | $\geq 68$ | 66 | 64.04 / 0.65 | 64.06 / 0.60 | 64.36 / 0.67 | 64.25 / 0.61 | 64.25 / 0.55 |
| p_hat1500-2 | $\geq 65$ | 65 | 63.58 / 0.51 | 63.59 / 0.65 | 63.71 / 0.57 | 63.67 / 0.51 | 63.69 / 0.58 |
| p_hat1500-3 | $\geq 94$ | 93 | 91.61 / 0.92 | 91.71 / 0.79 | 91.84 / 0.81 | 91.95 / 0.82 | 91.81 / 0.83 |
| san400_0.7_1 | 40 | 40 | 23.55 / 4.95 | 25.25 / 6.93 | 25.03 / 6.56 | 24.39 / 5.86 | 24.12 / 5.40 |
| san400_0.7_2 | 30 | 30 | 18.33 / 2.14 | 18.15 / 1.73 | 18.45 / 2.40 | 18.34 / 2.10 | 18.21 / 1.45 |
| san400_0.7_3 | 22 | 17 | 15.81 / 0.50 | 15.89 / 0.49 | 15.96 / 0.40 | 15.92 / 0.50 | 15.98 / 0.49 |
| san400_0.9_1 | 100 | 100 | 71.48 / 21.51 | 74.69 / 21.88 | 73.43 / 21.62 | 78.86 / 22.05 | 78.75 / 21.58 |
| san1000 | 15 | 15 | 10.02 / 0.53 | 9.97 / 0.17 | 9.96 / 0.20 | 9.96 / 0.20 | 10.09 / 0.72 |
| sanr400_0.5 | 13 | 13 | 12.11 / 0.34 | 12.02 / 0.45 | 12.18 / 0.48 | 12.08 / 0.31 | 12.11 / 0.47 |
| sanr400_0.7 | 21 | 21 | 19.42 / 0.60 | 19.39 / 0.63 | 19.51 / 0.59 | 19.53 / 0.59 | 19.52 / 0.62 |
| frb45-21-1 | 45 | 39 | 36.73 / 0.69 | 36.88 / 0.77 | 36.90 / 0.69 | 37.10 / 0.81 | 36.93 / 0.72 |
| frb50-23-1 | 50 | 44 | 40.77 / 0.82 | 40.91 / 0.86 | 41.08 / 0.93 | 40.99 / 0.79 | 40.97 / 0.77 |
| frb53-24-1 | 53 | 45 | 42.41 / 0.84 | 42.31 / 0.88 | 42.51 / 0.78 | 42.55 / 0.73 | 42.56 / 0.86 |
| frb56-25-1 | 56 | 48 | 44.84 / 0.91 | 44.91 / 0.75 | 45.07 / 0.89 | 45.05 / 0.79 | 45.14 / 0.93 |
| frb59-26-1 | 59 | 51 | 47.66 / 0.84 | 47.90 / 0.78 | 47.84 / 0.86 | 47.80 / 0.82 | 47.99 / 0.89 |
| frb100-40 | 100 | 82 | 77.06 / 1.05 | 77.37 / 1.04 | 77.55 / 0.95 | 77.66 / 1.13 | 77.63 / 1.08 |

shared memory framework of ABO and analyzing the effects of different network topologies on communications among colonies of ants. These ideas will be investigated more thoroughly in future studies.

## 6. ACKNOWLEDGMENTS

## 7. REFERENCES

[1] S. Alonso, O. Cordon, I. Fernandez de Viana, F. Herrera, "Integrating Evolutionary Computation Components in Ant Colony Optimization," Recent Developments in Biologically Inspired Computing, L.Nunes de Castro, F.J. Von Zuben (Eds.), Idea Group Publishing, 2004, pp. 48–180.

[2] P. Berman and A. Pelc, "Distributed Fault Diagnosis For Multiprocessor Systems," Proc. of the $20^{th}$ Annual International Symposium on Fault-Tolerant Computing, Newcastle, UK, 1990, pp. 340–346.

**Table 2: ABOMC$_{SM}$ Average Running Time and Improvement Factor in Running Time**

| Graph | Vertices/Edges | Running Time†: $T_{avg}/IF(c)$ when run on $c$ cores | | | | |
|---|---|---|---|---|---|---|
| | | $c = 1$(Seq) | $c = 2$ | $c = 4$ | $c = 6$ | $c = 8$ |
| brock200_1 | 200 / 14834 | 2.43 | 1.59 / 1.53 | 1.03 / 2.36 | 0.86 / 2.83 | 0.77 / 3.16 |
| brock200_2 | 200 / 9876 | 1.68 | 1.13 / 1.49 | 0.75 / 2.25 | 0.65 / 2.59 | 0.61 / 2.76 |
| brock400_1 | 400 / 59723 | 9.68 | 6.24 / 1.55 | 3.83 / 2.53 | 2.95 / 3.28 | 2.47 / 3.92 |
| brock400_2 | 400 / 59786 | 9.71 | 6.25 / 1.55 | 3.82 / 2.54 | 2.97 / 3.27 | 2.46 / 3.95 |
| brock800_1 | 800 / 207505 | 47.95 | 27.70 / 1.73 | 15.99 / 3.00 | 11.80 / 4.06 | 9.64 / 4.98 |
| brock800_2 | 800 / 208166 | 48.20 | 27.82 / 1.73 | 16.06 / 3.00 | 11.84 / 4.07 | 9.71 / 4.96 |
| c-fat500-1 | 500 / 4459 | 1.11 | 0.91 / 1.22 | 0.70 / 1.59 | 0.72 / 1.54 | 0.76 / 1.46 |
| c-fat500-5 | 500 / 23191 | 4.02 | 2.65 / 1.52 | 1.69 / 2.38 | 1.37 / 2.93 | 1.23 / 3.25 |
| c-fat500-10 | 500 / 46627 | 8.30 | 5.33 / 1.56 | 3.26 / 2.54 | 2.53 / 3.27 | 2.23 / 3.72 |
| c250.9 | 250 / 27984 | 4.70 | 3.01 / 1.56 | 1.89 / 2.48 | 1.46 / 3.23 | 1.43 / 3.29 |
| c500.9 | 500 / 112332 | 20.16 | 12.53 / 1.61 | 7.59 / 2.66 | 7.04 / 2.86 | 5.03 / 4.01 |
| c1000.9 | 1000 / 450079 | 135.20 | 74.08 / 1.83 | 41.12 / 3.29 | 29.98 / 4.51 | 24.43 / 5.54 |
| c2000.5 | 2000 / 999836 | 363.98 | 194.34 / 1.87 | 105.85 / 3.44 | 76.91 / 4.73 | 62.32 / 5.84 |
| c2000.9 | 2000 / 1799532 | 701.52 | 370.25 / 1.89 | 201.49 / 3.48 | 145.18 / 4.83 | 117.55 / 5.97 |
| c4000.5 | 4000 / 4000268 | 1729.46 | 899.67 / 1.92 | 484.65 / 3.57 | 350.00 / 4.94 | 283.38 / 6.10 |
| dsjc500.5 | 500 / 125248 | 10.33 | 7.24 / 1.43 | 4.33 / 2.38 | 3.39 / 3.05 | 2.83 / 3.66 |
| dsjc1000.5 | 1000 / 499652 | 64.55 | 37.89 / 1.70 | 21.41 / 3.01 | 15.69 / 4.11 | 12.80 / 5.04 |
| gen200_p0.9_44 | 200 / 17910 | 3.00 | 1.94 / 1.55 | 1.21 / 2.49 | 1.00 / 3.01 | 0.90 / 3.35 |
| gen200_p0.9_55 | 200 / 17910 | 3.04 | 1.95 / 1.55 | 1.22 / 2.50 | 1.01 / 3.02 | 0.92 / 3.32 |
| gen400_p0.9_55 | 400 / 71820 | 11.82 | 7.64 / 1.55 | 4.61 / 2.56 | 3.56 / 3.32 | 2.95 / 4.00 |
| gen400_p0.9_65 | 400 / 71820 | 11.82 | 7.61 / 1.55 | 4.62 / 2.56 | 3.56 / 3.32 | 2.96 / 3.99 |
| gen400_p0.9_75 | 400 / 71820 | 11.86 | 7.64 / 1.55 | 4.62 / 2.57 | 3.56 / 3.33 | 2.96 / 4.01 |
| hamming6-2 | 64 / 1824 | 0.39 | 0.28 / 1.39 | 0.23 / 1.71 | 0.23 / 1.70 | 0.23 / 1.66 |
| hamming6-4 | 64 / 704 | 0.15 | 0.13 / 1.15 | 0.14 / 1.07 | 0.14 / 1.08 | 0.14 / 1.07 |
| hamming8-2 | 256 / 31616 | 6.32 | 3.92 / 1.61 | 2.38 / 2.65 | 1.89 / 3.34 | 1.57 / 4.02 |
| hamming8-4 | 256 / 20864 | 3.44 | 2.21 / 1.55 | 1.38 / 2.49 | 1.11 / 3.10 | 1.02 / 3.37 |
| hamming10-2 | 1024 / 518656 | 196.52 | 106.87 / 1.84 | 59.38 / 3.31 | 43.49 / 4.52 | 35.39 / 5.55 |
| hamming10-4 | 1024 / 434176 | 127.18 | 69.47 / 1.83 | 38.42 / 3.31 | 27.97 / 4.55 | 22.68 / 5.61 |
| johnson8-2-4 | 28 / 210 | 0.05 | 0.05 / 0.92 | 0.07 / 0.73 | 0.08 / 0.66 | 0.08 / 0.64 |
| johnson8-4-4 | 70 / 1855 | 0.36 | 0.26 / 1.39 | 0.22 / 1.65 | 0.22 / 1.65 | 0.22 / 1.68 |
| johnson16-2-4 | 120 / 5460 | 0.91 | 0.62 / 1.48 | 0.43 / 2.10 | 0.40 / 2.31 | 0.40 / 2.29 |
| johnson32-2-4 | 496 / 107880 | 18.08 | 11.40 / 1.59 | 6.75 / 2.68 | 5.08 / 3.56 | 4.22 / 4.29 |
| keller4 | 171 / 9435 | 1.57 | 1.04 / 1.51 | 0.68 / 2.31 | 0.60 / 2.62 | 0.57 / 2.77 |
| keller5 | 776 / 225990 | 54.05 | 30.81 / 1.75 | 17.64 / 3.06 | 12.96 / 4.17 | 10.65 / 5.08 |
| keller6 | 3361 / 4619898 | 994.50 | 1035.62 / 1.93 | 558.68 / 3.57 | 402.71 / 4.95 | 324.74 / 6.14 |
| mann_a9 | 45 / 918 | 0.20 | 0.16 / 1.29 | 0.16 / 1.26 | 0.15 / 1.32 | 0.15 / 1.33 |
| mann_a27 | 378 / 70551 | 13.89 | 8.66 / 1.60 | 5.13 / 2.71 | 3.95 / 3.52 | 3.36 / 4.13 |
| mann_a45 | 1035 / 533115 | 199.46 | 108.58 / 1.84 | 60.70 / 3.29 | 45.03 / 4.43 | 36.99 / 5.39 |
| mann_a81 | 3321 / 5506380 | 2863.99 | 1507.33 / 1.90 | 837.19 / 3.42 | 616.96 / 4.64 | 505.63 / 5.66 |
| p_hat500-2 | 500 / 62946 | 11.51 | 7.29 / 1.58 | 4.43 / 2.60 | 3.43 / 3.36 | 2.87 / 4.02 |
| p_hat500-3 | 500 / 93800 | 16.29 | 10.39 / 1.57 | 6.20 / 2.63 | 4.70 / 3.46 | 3.97 / 4.11 |
| p_hat700-2 | 700 / 121728 | 25.07 | 15.44 / 1.62 | 9.21 / 2.72 | 6.92 / 3.62 | 5.61 / 4.47 |
| p_hat700-3 | 700 / 183010 | 40.99 | 24.18 / 1.70 | 13.89 / 2.95 | 10.25 / 4.00 | 8.42 / 4.87 |
| p_hat1000-2 | 1000 / 244799 | 68.53 | 38.49 / 1.78 | 21.86 / 3.14 | 15.99 / 4.29 | 13.09 / 5.23 |
| p_hat1000-3 | 1000 / 371746 | 107.85 | 59.44 / 1.81 | 33.33 / 3.24 | 24.24 / 4.45 | 19.78 / 5.45 |
| p_hat1500-2 | 1500 / 568960 | 205.26 | 110.73 / 1.85 | 61.09 / 3.36 | 44.19 / 4.64 | 35.95 / 5.71 |
| p_hat1500-3 | 1500 / 847244 | 298.42 | 160.31 / 1.86 | 87.91 / 3.39 | 63.69 / 4.69 | 51.78 / 5.76 |
| san400_0.7_1 | 400 / 55860 | 9.15 | 5.93 / 1.54 | 3.69 / 2.48 | 3.01 / 3.04 | 2.34 / 3.91 |
| san400_0.7_2 | 400 / 55860 | 9.01 | 5.81 / 1.55 | 3.60 / 2.51 | 2.78 / 3.24 | 2.34 / 3.85 |
| san400_0.7_3 | 400 / 55860 | 9.04 | 5.82 / 1.55 | 3.59 / 2.52 | 2.79 / 3.24 | 2.32 / 3.89 |
| san400_0.9_1 | 400 / 71820 | 11.87 | 7.66 / 1.55 | 4.64 / 2.56 | 3.59 / 3.31 | 2.98 / 3.98 |
| san1000 | 1000 / 250500 | 63.37 | 35.87 / 1.77 | 20.36 / 3.11 | 15.08 / 4.20 | 12.29 / 5.16 |
| sanr400_0.5 | 400 / 39984 | 6.59 | 4.25 / 1.55 | 2.66 / 2.47 | 2.09 / 3.15 | 1.79 / 3.69 |
| sanr400_0.7 | 400 / 55869 | 9.07 | 5.87 / 1.54 | 3.86 / 2.35 | 2.82 / 3.22 | 2.35 / 3.86 |
| frb45-21-1 | 945 / 386854 | 111.17 | 61.51 / 1.81 | 34.35 / 3.24 | 25.11 / 4.43 | 20.41 / 5.45 |
| frb50-23-1 | 1150 / 580603 | 186.44 | 100.98 / 1.85 | 55.77 / 3.34 | 40.44 / 4.61 | 32.87 / 5.67 |
| frb53-24-1 | 1272 / 714129 | 238.53 | 128.33 / 1.86 | 70.70 / 3.37 | 51.39 / 4.64 | 41.74 / 5.71 |
| frb56-25-1 | 1400 / 869624 | 300.38 | 161.17 / 1.86 | 88.45 / 3.40 | 64.09 / 4.69 | 52.03 / 5.77 |
| frb59-26-1 | 1534 / 1049256 | 375.84 | 200.63 / 1.87 | 109.89 / 3.42 | 79.54 / 4.73 | 64.57 / 5.82 |
| frb100-40 | 4000 / 7425226 | 3310.85 | 1715.33 / 1.93 | 926.45 / 3.57 | 666.59 / 4.97 | 537.15 / 6.16 |

†All times are in seconds.

[3] E. Bonabeau, M. Dorigo, and G. Theraulaz, "Inspiration for Optimization from Social Insect Behavior," Nature, Vol. 406, July 6, 2000, pp. 39–42.

[4] T. Bui and J. Rizzo, "Finding Maximum Cliques with Distributed Ants," Proc. of the Genetic and Evolutionary Computation Conf., 2004, pp. 24–35.

[5] T. Bui and G. Sundarraj, "Ant System for the $k$-Cardinality Tree Problem," Proc. of the Genetic and Evolutionary Computation Conf., 2004, pp. 36–47.

[6] T. Bui and L. Strite, "An Ant System Algorithm for Graph Bisection," Proc. of the Genetic and Evolutionary Computation Conf., 2002, pp. 43–51.

[7] T. Bui and C. Zrncic, "An Ant-Based Algorithm for Finding Degree-Constrained Minimum Spanning Tree," Proc. of the Genetic and Evolutionary Computation Conf., 2006, pp. 11–18.

[8] B. Bullnheimer, G, Kotsis, and C. Strauss, "Parallelization Strategies for the Ant System," High

**Table 3: Improvement Factor in the Running Time, $IF(c)$**

| Graph | Vertices/Edges | 95% Confidence Interval for $IF(c)^{\ddagger}$ | | | |
|---|---|---|---|---|---|
| | | $c = 2$ | $c = 4$ | $c = 6$ | $c = 8$ |
| johnson8-2-4 | 28/210 | (0.833, 1.200) | (0.714, 0.857) | (0.625, 0.833) | (0.625, 0.750) |
| hamming6-4 | 64/704 | (1.071, 1.231) | (1.000, 1.364) | (0.938, 1.250) | (0.938, 1.231) |
| c-fat500-5 | 500/23191 | (1.504, 1.560) | (2.083, 2.457) | (2.735, 3.116) | (2.804, 3.398) |
| gen400_p0.9_55 | 400/71820 | (1.523, 1.569) | (2.468, 2.646) | (3.118, 3.542) | (3.752, 4.157) |
| hamming10-2 | 1024/518656 | (1.805, 1.890) | (3.268, 3.382) | (4.473, 4.632) | (5.494, 5.703) |
| keller6 | 3361/4619898 | (1.908, 1.983) | (3.535, 3.672) | (4.904, 5.096) | (6.111, 6.320) |

$^{\ddagger}$Resampling population of size $10,000$ is used in the bootstrap.

Performance Algorithms and Software in Nonlinear Optimization, Kluwer, Dordrecht, 1998, pp. 87–100.

[9] P. Delisle, M. Krajecki, M. Gravel, and C. Gagné, "Parallel Implementation of An Ant colony Optimization Metaheuristic With OpenMP," Proc. of the 3rd European Workshop on OpenMP (EWOMP' 01), Barcelona, Spain, 2001.

[10] M. Dorigo, "Optimization, Learning and Natural Algorithms," Ph.D. Thesis, Politecnico di Milano, Italy, [in Italian], 1992.

[11] M. Dorigo and G. Di Caro, "The Ant Colony Optimization Meta-Heuristic," New Ideas in Optimization, McGraw-Hill, 1999, pp. 11–32.

[12] M. Dorigo and L. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," IEEE Trans. on Evol. Computation, 1(1), 1997, pp. 53–66.

[13] M. Dorigo, L. Gambardella, and E. Taillard, "Ant Colonies for the Quadratic Assignment Problem," Journal of the Operational Research Society, Vol. 50, 1999, pp. 167–176.

[14] I. Ellabib, P. Calamai, and O. Basir, "Exchange Strategies for Multiple Ant Colony System," Information Sciences, 177(5), March 2007, pp. 1248–1264.

[15] D. Geer, "Chip Makers Turn to Multicore Processors," Computer, Vol. 38, No. 5, May 2005, pp. 11–13.

[16] J. Hastad, "Clique Is Hard to Approximate within $n^{1-\epsilon}$," Acta Mathematica, 182, 1999, pp. 105–142.

[17] J. Lagarias and P. Shor, "Keller's Cube-Tiling Conjecture Is False In High Dimensions," Bulletin of the American Mathematical Society, 27(2), 1992, pp. 279–283.

[18] M. Manfrin, M. Birattari, Thomas Stützle, and M. Dorigo, "Parallel Ant Colony Optimization for the Traveling Salesman Problem," M. Dorigo et al. (Eds.): ANTS 2006, Lecture Notes in Computer Science, 4150, 2006, pp. 224–234.

[19] V. Maniezzo and A. Carbonaro, "Ant Colony Optimization: An Overview," Essays and Surveys in Metaheuristics, C. Ribeiro editor, Kluwer Academic Publishers, 2001, pp. 21–44.

[20] R. Michels and M. Middendorf, "An Ant System for the Shortest Common Supersequence Problem," in D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, 1999, pp. 51–61.

[21] M. Middendorf, F. Reischle, and H. Schmeck, "Multi Colony Ant Algorithms," Journal of Heuristic, 8, 2002, pp. 305–320.

[22] J. Parkhurst, J. Darringer, and B. Grundmann, "From Single Core to Multi-Core: Preparing for a New Exponential," Proc. of the 2006 IEEE/ACM International Conference on Computer-Aided Design, 2006, pp. 67–72.

[23] D. Patterson and J. Hennessy, "Computer Organization and Design ($2^{nd}$ Edition)," Morgan Kaufmann Publishers, 1998.

[24] M. Randall and A. Lewis, "A Parallel Implementation of Ant Colony Optimization," Journal of Parallel and Distributed Computing, 62(9), 2002, pp. 1421–1432.

[25] J. E. Savage and M. Zubair, "A Unified Model for Multicore Architectures," Proc. of the $1^{st}$ International Forum on Next-Generation Multicore/Manycore Technologies, 2008.

[26] N. Sloane, "Unsolved Problems in Graph Theory Arising from the Study of Codes," Graph Theory Notes of New York, XVIII, 1989, pp. 11–20.

[27] N. Sloane and F. MacWilliams, "The Theory of Correcting Codes," North Holland, Amsterdam, 1979.

[28] T. Stützle, "Parallelization Strategies for Ant Colony Optimization," Proc. of Parallel Problem Solving from Nature, Lecture Notes in Computer Science, 1498, Springer, 1998, pp. 722–741.

[29] E.-G. Talbi, O. Roux, C. Fonlupt, and D. Robillard, "Parallel Ant Colonies for Combinatorial Optimization Problems," Feitelson & Rudolph (Eds.), Job Scheduling Strategies for Parallel Processing: IPPS '95 Workshop, Lecture Notes in Computer Science, 949, Springer, Vol. 11, 1999.

[30] The Beowulf Project. http://www.beowulf.org. Last accessed March 2009.

[31] MPI - The Message Passing Interface Standard. http://www-unix.mcs.anl.gov/mpi/. Last accessed March 2009.

[32] OpenMP Architecture Review Board. http://www.openmp.org/specs/. Last accessed March 2009.

[33] Clique Benchmark Instances. http://www.cs.hbg.psu.edu/benchmarks/. Last accessed March 2009.

[34] BHOSLIB: Benchmarks with Hidden Optimum Solutions for Graph Problems. http://www.nlsde.buaa.edu.cn/ kexu/benchmarks/graph-benchmarks.htm. Last accessed March 2009.

[35] Supplemental results. http://www.cs.unm.edu/∼tnguyen/Files/Papers/mc-sup.pdf. Last accessed March 2009.