Report Series

Parallelization Strategies for the Ant System

Bernd Bullnheimer
Gabriele Kotsis
Christine Strauß

Report No. 8
October 1997

October 1997

SFB
'Adaptive Information Systems and Modelling in Economics and Management Science'

Vienna University of Economics
and Business Administration
Augasse 2–6, 1090 Wien, Austria

in cooperation with
University of Vienna
Vienna University of Technology

http://www.wu-wien.ac.at/am

Papers published in this report series
are preliminary versions of journal articles
and not for quotations.

This paper was accepted for publication in:
Kluwer Series of Applied Optimization.
Selected Papers of High Performance Software for Nonlinear Optimization: Status and Perspectives (HPSNO'97),
edited by Murli A., Pardalos P., Toraldo G.

This piece of research was supported by the Austrian Science Foundation (FWF) under grant SFB#010 ('Adaptive Information Systems and Modelling in Economics and Management Science').

Parallelization Strategies for the Ant System

Bernd Bullnheimer[1]
Gabriele Kotsis[2]
Christine Strauss[1]

[1] Department of Management Science, University of Vienna, Austria
{bullnhei, strauss}@pom.bwl.univie.ac.at

[2] Institute of Applied Computer Science, University of Vienna, Austria
gabi@ani.univie.ac.at

Abstract

The Ant System is a new meta-heuristic method particularly appropriate to solve hard combinatorial optimization problems. It is a population-based, nature-inspired approach exploiting positive feedback as well as local information and has been applied successfully to a variety of combinatorial optimization problem classes. The Ant System consists of a set of cooperating agents (artificial ants) and a set of rules that determine the generation, update and usage of local and global information in order to find good solutions. As the structure of the Ant System highly suggests a parallel implementation of the algorithm, in this paper two parallelization strategies for an Ant System implementation are developed and evaluated: the synchronous parallel algorithm and the partially asynchronous parallel algorithm. Using the Traveling Salesman Problem a discrete event simulation is performed, and both strategies are evaluated on the criteria "speedup", "efficiency" and "efficacy". Finally further improvements for an advanced parallel implementation are discussed.

1 Introduction

The Ant System, introduced by Colorni, Dorigo and Maniezzo [5], [8], [10] is a new member in the class of meta-heuristics (cf. e.g. [14],[15]) to solve hard combinatorial optimization problems. Many well-known methods of this class are modeled on processes in nature. The same is true for the Ant System that imitates real ants searching for food. Real ants are capable to find the shortest path from a food source to their nest without strength of vision. They use an aromatic essence, called pheromone, to communicate information regarding the food source. While ants move along, they lay pheromone on the ground which stimulates other ants rather to follow that trail than to use a new path. The quantity of pheromone a single ant deposits on a path depends on the total length of the path and on the quality of the food source discovered. As other ants observe the pheromone trail and are attracted to follow it, the pheromone on the path will be intensified and reinforced and will therefore attract even more ants. Roughly speaking, pheromone trails leading to rich, nearby food sources will be more frequented and will grow faster than trails leading to low-quality, far away food sources.

The above described behavioral mechanism of real ants was the pattern for a new solving procedure for combinatorial optimization problems. It has inspired Ant System using the following analogies: artificial ants searching the solution space correspond to real ants searching their environment for food, the objective values are the equivalent to the food sources quality and an adaptive memory represents the pheromone trails. The artificial ants are additionally equiped with a local heuristic function to guide their search through the set of feasible solutions.

The application of the Ant System to Traveling Salesman [10], Quadratic Assignment [13], Vehicle Routing [4], Job Shop Scheduling [6], Graph Coloring [7] and Timetabling [2] is evidence for the methods' versatility. As problem size increases performance becomes a crucial criteria. Furthermore, the structure of the Ant System algorithm is highly suitable for parallelization. Both mentioned considerations were the motivation to improve the Ant Systems' performance using parallelization.

The remainder of the paper is organized as follows: initially we explain the Ant System algorithm and use the Traveling Salesman Problem for illustration purposes (§2). By identifying the problem-inherent parallelism we develop two parallelization strategies for the Ant System in Section 3. The performance of both strategies is compared using simulation experiments and algorithmic aspects as well as critical performance factors are discussed (§4), followed by a brief conclusion.

2 The Ant System Algorithm

In the following the Ant System algorithm is explained on the Traveling Salesman Problem (TSP), probably the most studied problem of combinatorial op-

timization, where a traveling salesman has to find the shortest route visiting several cities and returning to his home location.

More formally, the TSP can be represented by a complete weighted graph $G = (V, E, d)$ where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of vertices (cities) and $E = \{(v_i, v_j) : i \neq j\}$ is a set of edges. Associated with each edge (v_i, v_j) is a nonnegative weight d_{ij} representing the distance (cost) between cities v_i and v_j. The aim is to find a minimum length (cost) tour beginning and ending at the same vertex and visiting each vertex exactly once.

Given an n-city TSP, the artificial ants are distributed to the cities according to some rule. At the beginning of an iteration, all cities except the one the ant is located in, can be selected. Until the tour is completed, each ant decides independently which city to visit next, where only not yet visited cities are feasible for selection.

The probability that a city is selected is the higher the more intense the trail level leading to the city is and the nearer the city is located. The intensity of the trail can be interpreted as an *adaptive memory* and is regulated by a parameter α. The latter criteria can be interpreted as a measure of *desirability* and is called visibility. It represents the local heuristic function mentioned above and is regulated by a parameter β. The probability that city v_j is selected to be visited next after city v_i can be written in a formula as follows:

$$p_{ij} = \begin{cases} \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{h \in \Omega} [\tau_{ih}]^\alpha [\eta_{ih}]^\beta} & \text{if } j \in \Omega \\ 0 & \text{otherwise} \end{cases} \qquad (1)$$

$$\text{where } \eta_{ij} = \frac{1}{d_{ij}}$$

where	τ_{ij}	intensity of trail between cities v_i and v_j
	α	parameter to regulate the influence of τ_{ij}
	η_{ij}	visibility of city v_j from city i
	β	parameter to regulate the influence of η_{ij}
	Ω	set of cities, that have not been visited yet
	d_{ij}	distance between cities v_i and j

This selection process is repeated until all ants have completed a tour. In each step of an iteration the set of cities to be visited is reduced by one city and finally, when only one city is left, it is selected with probability $p_{ij} = 1$. For each ant the length of the tour generated is then calculated and the best tour found is updated.

Then the trail levels are updated: an underlying assumption of the Ant System concept is that the quantity of pheromone per tour is the same for all artificial ants. Therefore on shorter tours more pheromone is left per unit length. By analogy to nature, part of the pheromone trail evaporates, i.e. existing trails are reduced before new trails are laid. This is done to avoid early convergence

and is regulated by a parameter ρ. On basis of these updated trail levels the next iteration can be started. The updating of the trail levels τ_{ij} can be written in a formula as follows:

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \Delta\tau_{ij} \tag{2}$$

$$\text{where } \Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \text{ and } \Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{L_k} & \text{if ant } k \text{ travels on edge } (v_i, v_j) \\ 0 & \text{otherwise} \end{cases}$$

where		
	t	iteration counter
	$\rho \in [0,1]$	parameter to regulate the reduction of τ_{ij}
	$\Delta\tau_{ij}$	total change of trail level on edge (v_i, v_j)
	m	number of ants
	$\Delta\tau_{ij}^{k}$	change of trail level on edge (v_i, v_j) caused by ant k
	L_k	length of tour found by ant k

In Figure 1 the sequential algorithm for solving the n-city TSP in T iterations using m ants is given. Its computational complexity is of order $\mathcal{O}(T \cdot m \cdot n^2)$. Previous experiments have shown, that using $m = n$ ants for the n-city TSP and initially placing one ant in each city yields good results with respect to both, the quality of the best solution found and the rate of convergence [10]. We will therefore in the following assume $m = n$ and refer to this parameter as the problem size. As the number of iterations is independent of the problem size, the computational complexity is thus $\mathcal{O}(m^3)$.

3 The Parallel Algorithmic Idea

The computational complexity of the sequential algorithm hinders its use for solving large problems. To reduce the computation time, we introduce a parallel version of the Ant System. The "sequential" algorithm contains a high degree

```
Initialize
For t = 1 to T
    For k = 1 to m do
        Repeat until ant k has completed a tour
            Select city v_j to be visited next
            with probability p_ij given by equation (1)
        Calculate the length L_k of the tour generated by ant k
    Update the trail levels τ_ij on all edges according to
    equation (2)
End
```

Figure 1: Sequential Ant System Algorithm in PseudoCode

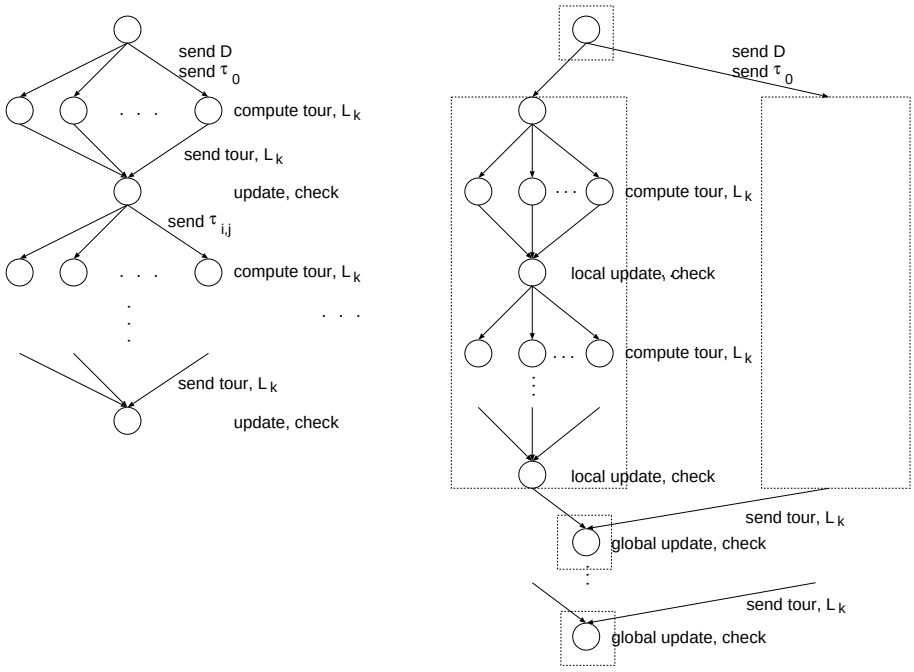

Figure 2: Synchronous (left) and Partially Asynchronous (right) Parallel Algorithm

of natural parallelism, i.e. the behavior of a single ant during one iteration is totally independent of the behavior of all other ants during that iteration.

Exploiting this problem-inherent parallelism, we develop an algorithmic idea for parallelization. Furthermore we discuss two implementation strategies: a synchronous (fork-join) algorithm and a partially asynchronous algorithm.

A straight forward parallelization strategy for the Ant System algorithm is to compute the TSP tours in parallel. This would result in a fork-join structure as shown in Figure 2 (left). An initial process (master) would spawn a set of processes, one for each ant. After distributing initial information about the problem (i.e. the distance matrix D and the initial trail intensities τ_0), each process can start to draw up the path and compute the tour length for its ant. After finishing this procedure, the result (tour and tour length L_k) is sent from each process back to the master process. The master process updates the trail levels by calculating the intensity of the trails and checks for the best tour found so far. A new iteration is initiated by sending out the updated trail levels.

Ignoring any communication overhead, this approach would imply optimum (asymptotic) speedup, assuming that an infinite number of processing elements (workers) is available, i.e. one process is assigned to one worker:

$$S_{asymptotic}(m) = \frac{T_{seq}(m)}{T_{par}(m,\infty)} = \frac{\mathcal{O}(m^3)}{\mathcal{O}(m^2)} = \mathcal{O}(m)$$

where $T_{seq}(m) = \mathcal{O}(m^3)$ is the computational complexity of the sequential algorithm for problem size m and $T_{par}(m, \infty) = \mathcal{O}(m^2)$ is the computational complexity of the parallel algorithm for problem size m and for infinite system size.

Departing from the above assumption communication overhead certainly cannot be disregarded and has to be taken into account, and further the system size (number of processing elements N) is restricted and is typically smaller ($N \ll m$) than the problem size (number of ants m). Therefore, a set of processes (ants) would be assigned to a physical processing element (worker), thus increasing the *granularity* of the application. Balancing the load among the workers is easily accomplished by assigning to worker j ($j = 1 \ldots N$) the processes (ants) m_i ($i = 1 \ldots m$) according to $\{m_i : j = i \bmod N\}$, thus each worker holds about the same number of processes and each process is of the same computational complexity.

When considering communication overhead, the ratio of the amount of computation assigned to a worker and the amount of data to be communicated has to be balanced. In the synchronous approach the frequency and volume of communication is rather high. After each iteration, all completed tours and their lengths have to be sent to a central process (master). There the new trail levels need to be computed and then broadcasted to each worker which only then can start a new iteration. This synchronization and communication overhead $T_{ovh}(m, N)$ typically slows down the performance and consumes part of the parallelization benefits. Speedup is reduced to

$$S(m, N) = \frac{\mathcal{O}(m^3)}{\mathcal{O}(m^3/N) + T_{ovh}(m, N)}$$

An estimate on $T_{ovh}(m, N)$ depends on the communication behavior of the underlying parallel architecture and typically cannot be given in a closed form.

In a next step of improvement we will reduce the frequency of communication. For this purpose we propose a partially asynchronous strategy as presented in Figure 2 (right). In this approach, each worker holds a certain number of processes (denoted by the dashed boxes in Fig. 2) and performs - independently of other workers - a certain number of iterations of the sequential algorithm on them consecutively. Only after these *local* iterations a *global* synchronization among all workers will take place, i.e. the master will globally update the trail levels.

While this approach is expected to reduce the communication overhead considerably, good and promising values obtained during local iterations might be ignored by other workers. For that reason a carefully chosen local/global iteration ratio is crucial.

4 Exploiting Parallelism

Parallel program behavior can either be evaluated using analytical techniques, using simulation models or by conducting measurement experiments on a real

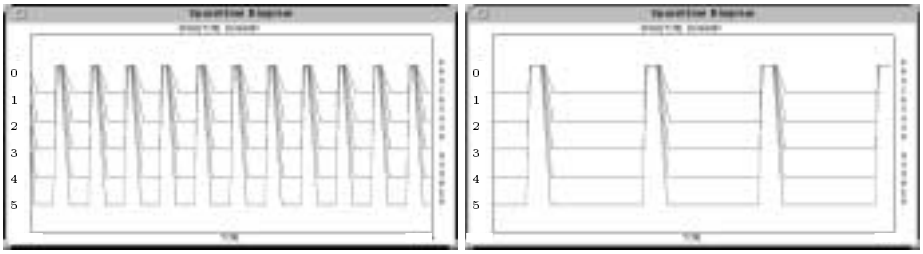

Figure 3: Communication Behavior for Synchronous (left) and Asynchronous (right) Version

implementation.

Analytical methods are based on an abstract, simplified model of the parallel program characteristics. For a detailed performance analysis and comparison of the two parallelization strategies, analytical methods are insufficient due to the complexity of estimating the communication overhead T_{ovh}. As the characteristics of the particular parallel machine will bias the performance of a real implementation, results obtained from measurements would be valid only for this distinct environment. Therefore, we decided to use discrete-event-simulation as an appropriate means to evaluate the parallel program behavior.

We apply a problem-specific tool called N-MAP [12]. The input for the simulator is the task behavior specification which is a C-like description of the parallel program structure and the resource requirements specification which includes an estimate for the computational complexity. For our program, we have defined three computational tasks (compute_tour, local_update and global_update) and two communication blocks (broadcast of trails, collection of paths).

When simulating the parallel execution, several assumptions on the communication behavior have to be made: in our experiments, we assumed, that sending a message involves a fixed startup time and a variable time depending on the size of the packet, which is reasonable for most parallel architectures.

The simulator will generate a trace file of the simulated parallel execution, which contains time stamps for the start and end of computation and communication blocks. Based on the time stamps in the trace file, the total execution time as well as several other performance measures can be derived. In addition, the behavior can be visualized. Figure 3 is a snapshot of the simulated communication behavior for a problem size of 50 using five workers and one master process for both strategies. The broadcasts from the master (process 0) and the collects from the workers (processes 1-5) are depicted as diagonal lines connecting the sender and the receiver. The horizontal lines correspond to phases of computation or to idle times due to waiting for communication. The diagrams show clearly the higher communication frequency of the synchronous version.

In the following sections 4.1 and 4.2, we will compare the two strategies with

respect to several performance indices [11], namely

1. the ratio of computation, communication and idle times in relation to the total simulated execution time,
2. the speedup $S(N) = T(1)/T(N)$,
3. the efficiency $E(N) = S(N)/N$, and
4. the efficacy $\eta(N) = S(N) \cdot E(N)$.

For the comparison, we have chosen exemplarily three different problem sizes: small ($m = 50$), medium ($m = 250$), and large ($m = 500$). The number of workers (excluding the master) has been varied from $N = 5, 10, 15, 20, 25$. In 4.3 we will present and discuss algorithmic modifications of the parallel Ant System algorithm.

4.1 Synchronous Parallel Implementation

Fig. 4 (left) shows the simulated busy, communication and idle times for the synchronous parallel implementation. The busy time contains all periods of the total execution time, where computations were performed. The communication time consists of the time necessary to prepare and send a message and depends on the amount of data which is sent in the message. The idle time is defined as the time where neither computation nor communication takes place, i.e. a process is waiting for a message or has not started / already finished its computation. The sum of computation, communication and idle times gives the total simulated execution time, which is an accumulated time measure due to overlapping computation, communication and idle phases.

For the small problem size, the idle phases are the dominant time fraction due to the frequent waiting for messages. With increasing problem size, the ratio of computation and idle times improves significantly and communication time becomes negligible. However, this effect is caused by the assumptions made in the simulation, that multiple communications can be performed simultaneously without contention.

The ratio of computation time decreases with an increasing number of workers indicating the decrease in utilization. In order to find the optimum number of processing elements, we have to compare the increase in speedup and the loss in efficiency. The corresponding results are presented in the diagrams in Figure 5 (left), which depicts the change of the three key indicators as a function of the number of workers used.

The left top diagram shows the speedup, the ratio of the simulated execution time of the sequential version to the simulated execution time of the parallel version using N workers. The gained speedup is poor for small problems. It even decreases if the number of workers is too large, thus resulting in a relative "slowdown" and in an efficiency close to zero (left middle diagram).

For larger problems, speedup is nearly optimum $S(N) \approx N$ (close to the number of workers) and efficiency decreases much slower. Relating speedup

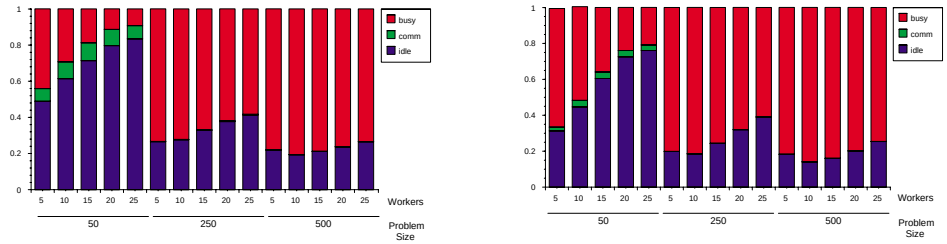

Figure 4: Relative Computation, Communication, and Idle Times for the Synchronous (left) and the Asynchronous (right) Parallel Implementation

and efficiency, we can determine the optimum number of workers which is the maximum of the efficacy curve (left bottom diagram). The slope of the efficacy curve for the largest problem size investigated indicates, that even more than 25 workers could be efficiently used to solve the problem.

4.2 Partially Asynchronous Parallel Implementation

The core idea of the partially asynchronous parallel algorithm is to perform a number of local iterations before synchronizing information among all workers in a global exchange (cf. Figure 2 right). In the experiments, we have assumed a local/global iteration ratio of five to one.

Figure 4 (right) shows the behavior for the partially asynchronous strategy, which is similar to the behavior of the synchronous strategy, i.e. idle times decrease with increasing problem size and increase with increasing number of workers. Utilization benefits from the lower communication frequency which can be seen in the lowered idle times in the aggregated representation of Figure 4. Communication times are negligible even for the small problem size.

By reducing the communication frequency, the idle times were significantly reduced as the number of synchronization points was reduced. The effects of this improvement can be seen in Figure 5. For all problem sizes investigated the speedup (right top diagram) for the asynchronous parallel version outperforms the synchronous version. Analogous results were obtained with respect to efficiency (right middle diagram) and efficacy (right bottom diagram).

As the asynchronous algorithm focuses on the reduction of communication frequency, especially problems of small size benefit from this strategy, because of their relatively high communication and idle ratio. Considering - in contrast to the assumptions in the simulation experiments - a class of architectures where the simultaneous sending of messages is not possible, the asynchronous algorithm would be even more advantageous.

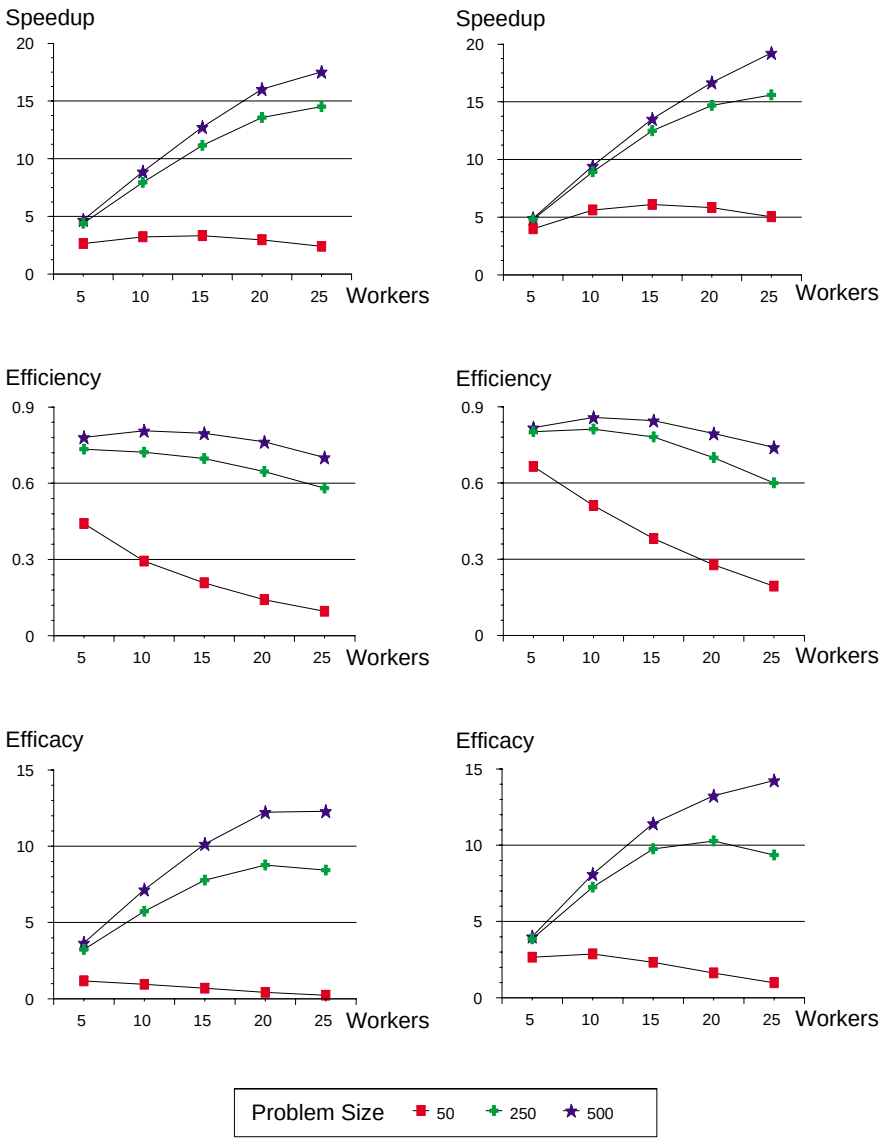

Figure 5: Speedup (top), Efficiency (middle) and Efficacy (bottom) for the Synchronous (left) and the Asynchronous (right) Parallel Implementation

4.3 Algorithmic Aspects

Variants of the two parallel algorithms introduced will focus on modifications which are expected to either gain further speedup while maintaining at least the same level of quality of results or to improve the solution quality without a loss in computation speed, or both.

While in the synchronous algorithm, the only tunable parameter is the rule for grouping the processes and assigning them to the workers, in the asynchronous parallel implementation also the number of local iterations can be varied.

The larger the *number of local iterations* the more the frequency of communication can be reduced, but at the same time the convergence rate would increase. The higher the local/global iteration ratio the more likely it is for workers to get trapped in a local optimum without knowing about good and promising solutions found by ants assigned to other workers. A static approach would perform an a priori fixed amount of local iterations or would iterate within a predefined fixed time-window. A usefull dynamic approach would perform only a few local iterations in the beginning to avoid early convergence and then successively increase the local/global iterations ratio, a concept similar to the cooling scheme of Boltzmann machines [1]. This may happen on a number-of-iterations or flexible-time-windows basis.

The decision upon synchronizing in a global exchange can also be made autonomously and self-regulated, e.g. if a worker finds a significant improvement in the tour length. However, such a worker-initiated synchronization requires a more sophisticated implementation, and might - as a consequence - lead to more communication overhead. The tradeoff between this overhead and the possible improvements for the search has to be found by experimental studies.

The criteria *grouping of processes* has two aspects: assignment and dynamics. Processes may be assigned randomly or may be assigned using the distance-criterion: ants initially placed on clustered cities could be treated on one worker or - in the contrary - rather distant processes could be assigned to a single worker. Another possibility would be an assigment according to a specific rule: it may assure that the agents are evenly distributed regarding their quality, i.e. every worker holds about the same number of "good" and "bad" agents (initial cities) or opposed to that, the "good" and "bad" ants are grouped, i.e. worker 1 holds the m/N best ants and so on and finally, worker N holds the m/N worst of them.

With respect to dynamics, the assignment process may be performed only once and the sets of processes may be kept in a static manner. The dynamic alternative implies repetitious selection and assignment processes after several global or local exchanges. Again, thorough testing will be needed to discover the most promising configuration.

Furthermore, recent publications addressing adaptive memory and trail update improved the general Ant System algorithm considerably. In [3] the ants are ranked according to solution quality and only the best ranked ants contribute to the trail update, whereas in [9] and [16] only the very best ant is

considered. In addition to that, they use local search to improve the solution generated by the artificial ants. Such combination of meta-heuristics with effective local search procedures is a new concept in combinatorial optimization that outperforms most other approaches. To investigate the effects of these ideas on the parallel strategies as proposed in this work is subject for future research.

5 Conclusion

The Ant System is a new distributed meta-heuristic for hard combinatorial optimization problems. It was first used to solve the Traveling Salesman Problem, and has later been applied successfully to other standard problems of combinatorial optimization such as Quadratic Assignment, Vehicle Routing, Job Shop, Scheduling, Graph Coloring and Timetabling using sequential implementations of the Ant System algorithm. As shown in [3] the Ant System shows good worst case results for large problems compared to other classical meta-heuristics. The desired application of the Ant System to large-scale problems and the distributed, modular structure of the algorithm were the motivation to parallelize the algorithm.

We developed two parallelization strategies, the Synchronous Parallel Algorithm and the Partially Asynchronous Parallel Algorithm, and used discrete event simulation to evaluate their performance. Based on the simulation output performance indices like speedup, efficiency, and efficacy were derived. Comparing the two strategies, we conclude, that the performance of the partially asynchronous strategy benefits from the reduced communication frequency, which is particularly important for an implementation on a real parallel architecture.

A critical discussion of the algorithmic aspects like number of local iterations, the assignment rules of ant processes to workers, and static versus dynamic approaches gives an impetus for further research.

References

[1] Aarts, E.H.; Korst, J.H.: Boltzmann Machines and Their Applications. In: Parallel Architectures and Languages Europe, Vol. I: Parallel Architectures. J.W. de Bakker, A.J. Nijman, and P.C. Treleaven (eds.), pp. 34 – 50. Springer Verlag, 1988.

[2] Bullnheimer, B.; Dorigo, M.: An Examination Scheduling Model to Maximize Students' Study Time and an Ant Approach to Solve it. Presentation at the 2^{nd} international conference on the Practice And Theory of Automated Timetabling (PATAT'97), Toronto, Canada, 1997.

[3] Bullnheimer, B.; Hartl, R.F.; Strauss, C.: A New Rank Based Version of the Ant System - A Computational Study. Working Paper No. 1, SFB Adaptive Information Systems and Modelling in Economics and Management Science, Vienna, 1997.

[4] Bullnheimer, B.; Hartl, R.F.; Strauss, C.: Applying the Ant System to the Vehicle Routing Problem. Paper presented at 2^{nd} International Conference on Metaheuristics (MIC'97), Sophia-Antipolis, France, 1997.

[5] Colorni, A.; Dorigo, M.; Maniezzo, V.: Distributed Optimization by Ant Colonies. In: Proceedings of the European Conference on Artificial Life (ECAL'91, Paris, France). Varela, F.; Bourgine, P. (eds.), pp. 134 – 142, Elsevier Publishing, Amsterdam, 1991.

[6] Colorni, A.; Dorigo, M.; Maniezzo, V.; Trubian, M.: Ant system for Job-Shop Scheduling. In: JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science 34 (1), pp. 39 – 53, 1994.

[7] Costa, D.; Hertz, A.: Ants can color graphs. In: Journal of the Operational Research Society 48, pp. 295 – 305, 1997.

[8] Dorigo, M.: Optimization, Learning and Natural Algorithms. Doctoral Dissertation, Politecnico di Milano, Italy (in Italian), 1992.

[9] Dorigo, M.; Gambardella, L.M.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. In: IEEE Transactions on Evolutionary Computation 1, pp. 53 – 66, 1997.

[10] Dorigo, M.; Maniezzo, V.; Colorni, A.: Ant System: Optimization by a Colony of Cooperating Agents. In: IEEE Transactions on Systems, Man, and Cybernetics 26 (1), pp. 29 – 41, 1996.

[11] Eager, D.L.; Zahorjan, J.; Lazowska, E.D.: Speedup versus Efficiency in Parallel Systems. In: IEEE Transactions on Computers, 38 (3), pp. 408 – 423, 1989.

[12] Ferscha, A.; Johnson, J.: Performance Prototyping of Parallel Applications in N-MAP. In: ICA^{3}PP96, Proc. of the 2^{nd} Internat. Conf. on Algorithms and Architectures for Parallel Processing, IEEE CS Press, pp. 84 – 91, 1996.

[13] Maniezzo, V.; Colorni, A.; Dorigo, M: The Ant System applied to the Quadratic Assignment Problem. Technical Report IRIDIA / 94-28, Universite Libre de Bruxelles, Belgium, 1994.

[14] Osman, I.H.; Kelly, J.P. (eds.): Meta-Heuristics: Theory & Applications, Kluwer, Boston, 1996.

[15] Osman, I.H.; Laporte, G.: Metaheuristics: A bibliography. In: Annals of Operations Research 63, pp. 513 – 623, 1996.

[16] Stuetzle, T.; Hoos H.: The MAX-MIN Ant System and Local Search for the Traveling Salesman Problem. In: Proceedings of ICEC'97 - 1997 IEEE 4th International Conference on Evolutionary Computation, IEEE Press, pp. 308 – 313, 1997.