MPI Parallelization of Bee Colony Optimization*

T. Davidović†
Mathematical Institute SASA,
Kneza Mihaila 36,
11001 Belgrade, Serbia
tanjad@mi.sanu.ac.rs

D. Ramljak
Center for DABI,
Temple University
322 Wachman Hall,
1805 N. Broad St.,
Philadelphia, PA 19122, USA
dusan.ramljak@temple.edu

M. Šelmić, D. Teodorović
Faculty of Transport and
Traffic Engineering,
University of Belgrade
Vojvode Stepe 305,
11010 Belgrade, Serbia
m.selmic@sf.bg.ac.rs
dusan@sf.bg.ac.rs

Abstract

The Bee Colony Optimization (BCO) algorithm is a meta-heuristic that belongs to the class of biologically inspired stochastic swarm optimization methods, based on the foraging habits of bees in nature. In this paper we propose two synchronous parallelization strategies for a distributed memory multiprocessor architecture under the MPI communication library. The presented experimental results addressing the problem of static scheduling independent tasks on identical machines show that our parallel BCO algorithm provides excellent performance when a coarse-grained parallelization strategy is used. On the other hand, a fine-grained strategy is not suitable for this kind of target multiprocessor architecture.

Keywords: Parallelization strategy; MPI communication library; meta-heuristics; scheduling problems.

1 Introduction

The Bee Colony Optimization (BCO) algorithm is a meta-heuristic that belongs to the class of biologically inspired stochastic swarm optimization methods, based on the foraging habits of honey bees. It has successfully been applied to various combinatorial optimization problems. The basic idea behind BCO is to create a multi-agent system (colony of artificial bees) capable of successfully solving difficult combinatorial optimization problems. The artificial bee colony behaves somewhat similarly to bee colonies found in nature, but differences between the two kinds of colonies are also evident. BCO is a meta-heuristic which belongs to the class of constructive methods. It was designed as a method which builds solutions from scratch within the execution steps, unlike the local search-based meta-heuristics which perform iterative improvements of the current best solution. To the best knowledge of the authors, no works have been reported on parallelization strategies of BCO, despite its inherent parallelism. Therefore, the main goal of this work is to study the potential strategies for the parallelization of BCO. We propose two synchronous parallelization strategies of BCO on distributed memory IBM HPC Linux Cluster Server+16 $\times$ 2 Dual Core Intel Processors on 2.33GHz/1333MHz with 4MB RAM, Ethernet 3rd Party e1350 SMC 8848M Switch Bundle. Our implementations use C as the programming language and utilize MPI communication library.

The rest of this paper is organized as follows. Section 2 contains a brief description of the BCO algorithm. Parallelization strategies and implementation details are given in Section 3. Experimental evaluation is described in Section 4, while Section 5 concludes the paper.

*This work has been supported by Serbian Ministry of Science and Technological Development, grants No. OI174010, OI174033 and TR36002.

†Corresponding author

2 Bee Colony Optimization

Lučić and Teodorović [6, 7] were among the first to use the basic principles of collective bee intelligence in solving combinatorial optimization problems. BCO is a population based algorithm in which a population of *artificial bees* (consisting of B individuals) searches for the optimal solution. Every artificial bee generates solutions to the problem within the construction steps, over multiple iterations until some predefined stopping criterion is met. Each step of the BCO algorithm is composed of two alternating phases: the *forward pass* and the *backward pass*, which are repeated until all solutions (one for each bee) are completed. During the forward pass, every bee is exploring the search space. It applies a predefined number of moves (NC), which construct a partial solution, yielding to a new (partial or complete) solution. Having obtained the new partial solutions, the bees start the second phase, the so-called backward pass.

At the beginning of the backward pass, all bees share information about their solutions. In nature, bees would perform a dancing ritual, which would inform other bees about the amount of food they have found, and the proximity of the patch to the hive. In the optimization search algorithm, the values of objective functions are compared. Each bee decides with a certain probability whether it will stay loyal to its solution or not. The bees with better solutions have a higher chance to keep and advertise them. The bees that are *loyal* to their partial solutions are called *recruiters*. Once a solution is abandoned, the bee becomes *uncommitted* and has to select one of the advertised solutions. This decision is made probabilistically, in such a way that the better advertised solutions have a bigger opportunity to be chosen for further exploration.

The two phases of the search algorithm, the forward and backward passes, alternate in order to generate all of the required solutions (one for each bee). When all solutions are completed the best among them is used to update the global best solution, and an iteration of BCO is completed. BCO iterations are repeated until a stopping condition is met. The possible stopping conditions can include the maximum total number of iterations, the maximum total number of iterations without improvement of the objective function, and the maximum allowed CPU time.Once the stopping condition is met, the global best solution is reported.

The BCO algorithm parameters whose values need to be set prior to the algorithm execution are: the number of bees B and the number of constructive moves during one forward pass (NC). The following is the pseudocode of the BCO algorithm:

Do

1. Initialization: an empty solution is assigned to each bee.
2. *For* $(s = 0; s < NC; s++)$ //count moves
   (a) *For* $(b = 0; b < B; b++)$ //**forward pass**
   1) Evaluate all possible moves;
   2) Choose one move using the roulette wheel.
   //**backward pass**
   (b) *For* $(b = 0; b < B; b++)$
   Evaluate (partial/complete) solution for bee b;
   (c) *For* $(b = 0; b < B; b++)$
   Loyalty decision using the roulette wheel for bee b;
   (d) *For* $(b = 0; b < B; b++)$
   If (b is uncommitted), choose a recruiter by the roulette wheel.
3. Evaluate all solutions and find the best one.

while stopping criteria is not satisfied.

3 Parallelization Strategies

The main goal of parallelization is to speedup the computations needed to solve a particular problem by engaging several processors and dividing the total amount of work between them. For stochastic algorithms this goal may be defined in one of the following two ways: 1) accelerate the search for the same quality solution or 2) improve the solution quality by allowing more processors to run the same amount of (CPU or wall-clock) time as a single one does. When meta-heuristics are considered, a combination of gains may be obtained: parallel execution can enable an efficient search of different regions of the solution space, yielding an improvement of the final solution quality within a smaller amount of execution time.

A significant amount of work has already been done on the parallelization of meta-heuristics. The approach can be twofold. The theoretical aspects of parallelization could be considered, and the practical applications of parallel meta-heuristics to different optimization problems proposed. The survey paper [3] summarizes these works and proposes an adequate taxonomy.

The well-known performance measures for parallel programs are *speed-up* S_q and *efficiency* E_q [1] and we also use them to evaluate and justify our approach. They are defined as follows:

$$S_q = \frac{T_{seq}^{best}}{T_q}, \quad E_q = \frac{S_q}{q} = \frac{T_{seq}^{best}}{qT_q}.$$

Here, T_{seq}^{best} denotes the execution time of best known sequential algorithm on a single processor, while T_q represents the execution time of the parallel algorithm on q processors.

When meta-heuristics are considered, the performance of parallelization strategy is also influenced by the quality of the final solution. Namely, meta-heuristics represent stochastic search procedures (and BCO is not an exception) which may not result in an identical solution even after repeated sequential executions. On the other hand, parallelization may assure the extension of the search space, which could result in either an improvement or a degradation of the final solution's quality. Therefore, the quality of the final solution should also be considered as a parameter of parallelization strategy performance.

3.1 Parallelization of BCO

The BCO algorithm is created as a multi-agent system which naturally provides a good basis for the parallelization on different levels. High-level parallelization assumes a coarse granulation of tasks and can be applied to iterations of BCO. Smaller parts of the BCO algorithm (the forward and backward passes within a single iteration) also contain a lot of independent executions, and are suitable for low-level parallelization. In this work we consider both strategies in a synchronous way.

Coarse-grained parallelization in its simplest form represents the independent execution of BCO on different processors. It could be obtained by the division of the stopping criterion among processors. For example, if the stopping criterion is allowed CPU time (given as a *runtime* value in seconds), we could run BCO in parallel on q processors for $runtime/q$ seconds. A similar rule can be introduced in the case when the stopping criterion is the allowed number of iterations. In both cases, each processor independently performs a sequential variant of BCO, but with a reduced value of the stopping criterion. We named this variant of parallelized BCO the distributed BCO (DBCO). Other way to implement the coarse-grained parallelization strategy could be the following: Instead of the stopping criterion, we could divide the number of bees. Namely, if the sequential execution uses B bees for the search, our parallel variant executing on q processors is using only B/q bees. This way results in a sequential BCO on each processor, but

with a smaller number of bees. This is also a distributed BCO, that we will refer to as BBCO since the bees are distributed among processors. Independent runs on different processors also allow us to change the search parameters, and we can therefore assure diversification of the search process.

Each artificial bee acts as an individual agent during the forward pass when partial solutions are generated. The generation of a partial solution is independent from the rest of the computations. This leads us to the fine-level parallelization. Within the concrete implementation, we have the following scenario: the forward pass is executed independently on each processor, while the backward pass requires a tight coordination between processors. For the corresponding computations within the backward pass it is necessary to have the information about all generated partial solutions. Nevertheless, those computations could also be spread among all processors and accompanied by the required communication. Our implementation of this strategy will be called FBCO.

The implementation of the fine-grained parallelization strategy required us to define the relation between the number of processors q and the number of bees B. Namely, each processor is responsible for B/q bees and these two numbers should be divisible. More details about the implementation and experimental evaluation of our approach to the parallelization of BCO are given in the next section.

4 Experimental Evaluation

The proposed parallelization strategies are tested on a scheduling problem. In our previous paper [5] the implementation of BCO for scheduling independent tasks to identical machines is proposed. This implementation represents an excellent starting point for developing parallelization strategies of the BCO method due to the simplicity of the treated problem. Therefore, we use it as a pattern in spite of the fact that the sequential version performed well.

For the experimental evaluation various problem instances, the instances that have been used in [5], are used. This allows us to easily compare the sequential and parallel versions of BCO, and measure the performance of various parallelization strategies.

We have chosen the representative subset of test examples, namely the hard test instances from [4] with *a priori* known optimal solutions.

Our target architecture for parallelized BCO is a homogeneous and completely connected network of processors. We distinguish the processor that communicates with the user and name it *master*. It is usually marked as processor 0. The other $q-1$ processors are called *working processors* or *slaves*. Their marks are processor 1 up to processor $q-1$. The parallel versions of BCO execute on all q processors, i.e. the computations are assigned to the master as well. In our experiments we used a different number of processors, ranging from 2 to 12.

To calculate speedup end efficiency, we assume that BCO from [5] is the best sequential algorithm. To assure fairness of obtained results, we compared the parallel versions of BCO with the original sequential version executed on a single processor of our parallel architecture (instead of the parallel version executed for $q = 1$).

4.1 Distributed BCO (DBCO)

First, we tested our coarse-grained parallelization strategy, named distributed BCO (DBCO). Table 1 contains the scheduling results for test instance with (*a priori*) known optimal solution consisting of 100 tasks while the number of machines is changed. For all examples, within DBCO we set $B = 5$, $NC = 10$ and the stopping criterion is selected to be 1000 iterations.

Table 1: DBCO Scheduling results - test problems with known optimal solutions from [4]

example	m	q	OPT	DBCO	DBCO CPU time	S_q	E_q
Iogra100	2	1	800	800	4.60	1.00	1.00
		2		800	2.55	1.80	0.90
		3		800	1.70	2.70	0.90
		4		800	1.27	2.60	0.90
		5		800	1.01	4.55	0.91
Iogra100	4	1	800	800	4.79	1.00	1.00
		2		800	2.54	1.89	0.94
		3		800	1.72	2.78	0.93
		4		800	1.28	3.74	0.93
		5		800	1.03	4.65	0.93
Iogra100	6	1	800	801	4.78	1.00	1.00
		2		801	2.58	1.85	0.93
		3		802	1.73	2.76	0.92
		4		801	1.29	3.71	0.93
		5		801	1.04	4.60	0.92
Iogra100	8	1	800	803	4.80	1.00	1.00
		2		803	2.58	1.86	0.93
		3		804	1.71	2.81	0.94
		4		804	1.28	3.75	0.94
		5		805	1.03	4.66	0.93
Iogra100	9	1	800	807	5.03	1.00	1.00
		2		806	2.59	1.94	0.97
		3		809	1.74	2.89	0.96
		4		808	1.33	3.78	0.95
		5		808	1.05	4.79	0.96
Iogra100	12	1	800	811	4.97	1.00	1.00
		2		814	2.62	1.9	0.95
		3		813	1.76	2.82	0.94
		4		813	1.32	3.76	0.94
		5		814	1.07	4.78	0.96
Iogra100	16	1	800	819	5.15	1.00	1.00
		2		821	2.62	1.96	0.98
		3		822	1.83	2.81	0.94
		4		819	1.38	3.73	0.93
		5		828	1.05	4.90	0.98

In the first column of Table 1 the name of the example is given. The second column contains the number m of machines within each example. The number of parallel processors q executing DBCO is given in the third column of our table. The optimal schedule length represents the content of column four, while lengths of schedules obtained by DBCO for different q are placed in the fifth column. Column six contains the CPU time required by DBCO to complete 1000

iterations (actually the CPU time required by q processor to complete $1000/q$ iterations). The corresponding speedup S_q and efficiency E_q are given in the last two columns. It is important to note that the CPU time required by DBCO to complete all necessary computations is actually the CPU time of the processor that is the last one to finish its work, i.e. it is equal to the maximum of all processors' running times. The resulting tables include the best obtained schedule length, as well as the worst (longest) required CPU time. To illustrate the results on each particular processor, Table 2 presents the running times and obtained schedule lengths for each processor when scheduling Iogra100 onto 6 machines. As can be seen from this table, both the running time and the solution quality may differ from processor to processor. The smallest schedule length and the largest execution time characterize distributed execution of BCO.

Table 2: DBCO detailed results for Iogra100 scheduled on 6 machines

q	1	2	3	4	5
CPU time	4.48	2.26	1.53	1.14	**1.04**
		2.58	1.69	**1.29**	0.92
			1.73	1.27	1.04
				1.22	1.02
					1.01
DBCO	801	804	803	805	803
		801	**802**	**801**	805
			804	803	804
				803	**801**
					805

Based on the results presented in Table 1 we can conclude that for those examples DBCO achieves an almost linear speedup with efficiency above 90%. Regarding the solution quality, we noticed that it is preserved for easier examples. Sometimes we noticed a slight improvement in the solution quality, while for larger examples (scheduling on larger number of machines) the parallelization may cause some degradation due to the reduction of the stopping criterion - (2.7% at most).

4.2 Bees distribution within BCO (BBCO)

The second way to implement the coarse-grained parallelization strategy (by dividing the total number of bees among the processors) we named BBCO. We tested this implementation on Iogra100 assuming that $B = 12$, $NC = 10$ and the stopping criterion is set to 1000 iterations. The number of processors q takes the following values: 2, 3, 4, 6, 12. This actually means that sequential BCO works with 12 bees, BBCO on $q = 2$ processors executes 1000 iterations with 6 bees, and so on. Finally, on $q = 12$ processors, BBCO is searching for schedules with one bee per processor. The results for the larger test instances are given in Table 3.

This type of coarse-grained parallelization strategy assures an excellent speedup and efficiency, due to the reduction of computations assigned to each processor. This resulted in a superlinear speedup and efficiency greatly above 1.00. Nevertheless, the difficulty of a given scheduling problem can not always be conquered and a small degradation of the solution quality may occur. On the other hand, as Table 3 shows, sometimes improvements are also evident.

Table 3: BBCO Scheduling results - test problems with known optimal solutions from [4]

example	m	q	OPT	BBCO	BBCO CPU time	S_q	E_q
Iogra100	9	1	800	807	11.88	1.00	1.00
		2		807	5.82	2.04	1.02
		3		806	3.54	3.56	1.12
		4		806	2.18	5.42	1.36
		6		806	0.80	14.85	2.48
		12		806	0.25	47.52	3.96
Iogra100	12	1	800	811	12.01	1.00	1.00
		2		813	5.88	2.04	1.02
		3		813	3.59	3.35	1.12
		4		814	2.19	5.48	1.37
		6		817	0.86	13.97	2.33
		12		817	0.27	44.48	3.71
Iogra100	16	1	800	819	12.18	1.00	1.00
		2		821	5.89	2.07	1.03
		3		822	3.59	3.39	1.13
		4		825	2.22	5.49	1.37
		6		825	0.91	13.38	2.23
		12		823	0.26	46.84	3.90

4.3 Fine-grained BCO (FBCO)

The implementation of the fine-grained parallelization strategy, is named FBCO. Due to the intensive communication between processors, we needed to minimize the amount of data exchanged during the search. Once again we used Iogra100 examples assuming $B = 12$, $NC = 10$ and 1000 iterations as the stopping criterion. The number of processors q takes the following values: 2, 3, 4, 6, 12. Results obtained by FBCO are given in Table 4.

As can be seen from Table 4, this parallelization strategy slows the computations. This is due to the communication delays which are caused by the intensive data exchange between processors. This strategy is obviously more suitable for shared-memory multiprocessor systems. In addition, asynchronous parallelization strategies are definitely the most interesting challenge for future work.

5 Conclusion

Two synchronous parallelization strategies for Bee Colony Optimization (BCO) meta-heuristic are proposed in this paper. They are implemented on completely connected homogeneous multiprocessor system, in which processors communicate by exchanging messages. The first strategy is based on independent execution of portions of the BCO algorithm on different processors and exchanging the best solution at the end. It is implemented in two ways, both of which proved to be very efficient. For the first variant the obtained speedup is almost linear, and quality of the solution is not degraded significantly (below 3% with respect to the sequential result). The sec-

Table 4: FBCO Scheduling results - test problems with known optimal solutions from [4]

example	m	q	OPT	FBCO	FBCO CPU time	S_q	E_q
Iogra100	4	1	800	800	11.51	1.00	1.00
		2		800	14.02	0.82	0.41
		3		800	14.99	0.77	0.26
		4		800	15.67	0.73	0.18
		6		800	16.42	0.70	0.12
		12		800	18.28	0.63	0.05
Iogra100	16	1	800	819	12.18	1.00	1.00
		2		821	14.30	0.85	0.43
		3		822	14.94	0.82	0.27
		4		823	15.75	0.77	0.19
		6		825	16.46	0.74	0.12
		12		825	18.39	0.66	0.06

ond way performed even better, resulting in a superlinar speedup and a negligible degradation of the quality of the solution. The fine-grained parallelization strategy represents parallelization at a lower level, and is actually based on cooperative work between several processors. Since it requires an intensive exchange of data between the processors during the search, it is more suitable for shared-memory multiprocessor systems. Therefore, our implementation of this strategy resulted in a small increase of the parallel BCO's execution time. This may be improved by a reduction of message exchange frequency or by developing an architecture-dependent parallelization strategy that would minimize the number of messages travelling through the system. The latest concepts represent interesting ideas for further research on this topic.

References

[1] S. Brawer. *Introduction to Parallel Programming.* Academic Press, Inc., 1989.

[2] T. G. Crainic, M. Gendreau, P. Hansen, and N. Mladenović. Cooperative parallel variable neighborhood search for the p-median. *J. Heur.*, 10(3):293–314, 2004.

[3] T. G. Crainic and N. Hail. Parallel meta-heuristics applications. In E. Alba, editor, *Parallel Metaheuristics*, pages 447–494. John Wiley & Sons, Hoboken, NJ., 2005.

[4] T. Davidović and T. G. Crainic. Benchmark problem instances for static task scheduling of task graphs with communication delays on homogeneous multiprocessor systems. *Comput. Oper. Res.*, 33(8):2155–2177, Aug. 2006.

[5] T. Davidović, M. Šelmić, D. Teodorović, and D. Ramljak. Bee colony optimization for scheduling independent tasks to identical processors. *Submitted for publication.* 2009.

[6] P. Lučić and D. Teodorović. Bee system: modeling combinatorial optimization transportation engineering problems by swarm intelligence. In *Preprints of the TRISTAN IV Triennial Symposium on Transportation Analysis*, pages 441–445. Sao Miguel, Azores Islands, 2001.

[7] P. Lučić and D. Teodorović. Vehicle routing problem with uncertain demand at nodes: the bee system and fuzzy logic approach. In J. L. Verdegay, editor, *Fuzzy Sets based Heuristics for Optimization*, pages 67–82. Physica Verlag: Berlin Heidelberg, 2003.