Self-adaptive Cooperation Scheme in a Parallel ACO Algorithm for Binary Combinatorial Problems

Roberto Prado-Rodríguez, Patricia González, Julio R. Banga, and Ramón Doallo

Computer Arquitecture Group, Faculty of Computer Science, Universidade da Coruña, 15071 A Coruña, Spain
Computer Arquitecture Group, Faculty of Computer Science, Universidade da Coruña, 15071 A Coruña, Spain
Computational Biology Lab, MBG-CSIC, Spanish National Research Council, Pontevedra, Spain
Computer Arquitecture Group, Faculty of Computer Science, Universidade da Coruña, 15071 A Coruña, Spain
Correspondence: roberto.prado@udc.es

DOI: *https://doi.org/10.17979/spudc.000024.12*

Abstract: The ant colony optimization (ACO) is widely used for combinatorial optimization problems, although it can suffer from fast convergence to local minima. In order to provide a versatile implementation of ACO, we present a parallel multicolony strategy with an improved cooperation scheme for binary combinatorial problems. Our proposal is based on a self-adaptive method, which assigns appropriate run-time cooperation levels to each problem based on its size and available computational resources. We evaluate this proposal with problems with different levels of cooperation and number of processes. All these configurations combined show its flexibility as a versatile solver for this type of problems.

1 Introduction

One of the most popular metaheuristic used for general combinatorial optimization problem is Ant Colony Optimization (ACO) (Dorigo and Stützle, 2019). It is inspired by the social behavior of ant colonies, specifically in the deposition of pheromones along the explored paths during the search for food sources. ACO has been found to be robust and easily tailored to a wide range of optimization problems, and it has been applied to a number of binary combinatorial instances (Jang et al., 2011; Kashef and Nezamabadi-pour, 2015; Kong and Tian, 2005).

The basic ACO is a general-purpose algorithm, easy to understand and implement. ACO achieves good results in unimodal problems, that is, those defined by the fact that all solutions are guided towards the same optimal result without local minima. However, when tackling problems in which local minima abound, its convergence quickly suffers, easily stagnating in one of the local solutions.

In this work we explore an extension of a parallel multicolony ACO implementation to handle challenging binary combinatorial problems, that incorporates a self-adaptive mechanism for colony cooperation.

The structure of the paper is as follows. Section 2 describes the ACO algorithm adapted to handle binary combinatorial problems and its cooperative parallel scheme proposed is explained. In section 3, we present the experiments carried out and discuss the results. Finally, in Section 4 we summarize the conclusions of this work.

2 Parallel ACO for binary optimization problems

ACO is often used for problems that can be reduced to finding routes in graphs, such as the Traveling Salesman Problem (TSP) (Stützle et al., 1999). This problem is based on discovering the best route for a traveller who has to visit many cities. In a classical TSP problem, the objective is to visit all the cities and return to the origin covering the shortest possible distance. When it comes to binary combinatorial problems, this can be reduced to finding the optimal path that goes from one node to another by choosing between two possible paths: 0 or 1. Pheromones will be deposited on paths 0 or 1 in each of the N steps. Ants in the subsequent iterations will be influenced by the previously deposited pheromones.

As mentioned before, ACO offers good results for solving unimodal problems. However, when it comes to solving problems with many local minima, ACO tends to get stuck easily. To avoid premature convergence to local minima and, thus, the stagnation of metaheuristics, previous studies indicate the need of increasing the diversity in the search. A good way to achieve this is the use of parallel strategies. The following subsections describe the solution used in this work and the enhancement introduced in the cooperation strategy.

Different parallel strategies can be applied to metaheuristics in general (Alba, 2005), and ACO in particular (González et al., 2022). Most of them can be classified into fine-grained and coarse-grained strategies. A fined-grained parallelization attempts to find parallelism in the sequential algorithm.

A different solution is a coarse-grained approach, which involves looking for a parallel variant of the sequential algorithm. The most popular coarse-grained solution consists of implementing an island-based model. In these models, different distributed colonies exist where the original algorithm is executed in isolation and, from time to time, these colonies exchange information that allow them update their results with the information received from the rest. This parallel implementation is usually known as multicolony model.

Multicolony approaches aim to take advantage of distributed resources to extend the search for solutions. The most trivial multicolony solution consists of a parallel search on multiple non-cooperating colonies. Although this solution was found to yield good results, results usually stand out for approaches that include colony cooperation.

In the cooperative scheme proposed in (González et al., 2022), when a promising new solution arrives at a colony and improves the *best-solution-so-far*, the latter is always replaced by the former. Therefore, a colony that receives a better solution is diverted from its own search. All colonies converge to the same local solution, reinforcing the same path, and eventually getting stuck at the same local minimum.

In this work an efficient selective cooperative scheme is explored. When a colony obtains a promising solution, this solution is spread to the rest and all processes receive the promising solutions. However, to avoid the problem mentioned, only a few processes introduce these solutions into their colony, modifying the pheromone matrix. To determine if a solution that has just arrived in a process should be included in the colony, two aspects are taken into account.

First, although all the colonies cooperate by spreading their promising solutions, some colonies keep their execution outside the influence of the rest, for which they never use the solutions received from outside the colony. This ensures the desired diversity in the ACO progression. The number of colonies that remain independent depends on an integer parameter called $cfreq$.

Second, to further avoid the danger of premature convergence due to early cooperation, the processes will only use the solutions received from other colonies once they have been stalled for a certain number of iterations. This number of iterations are defined by the $cstall$ parameter.

As a general rule, it is very complicated to know the optimal level of cooperation for a problem before dealing with it. To address this situation, a self-adaptive approach to automatically determine the tier of cooperation is also proposed in this paper. To do this, the problem size and the number of resources to be used are taken as a basis, and the parameters are tuned at runtime. Note that the hardness of the problem, which also influences the level of cooperation, is impossible to determine beforehand. The cooperation-index is the ratio between the size of the problem and the number of processes to be used. The higher this index is, the more intensive the cooperation between the colonies should be. Ranges of this index are established to set an upper limit to the *cstall* parameter. The minimum *cstall* is always 0 (full cooperation). At the start of the execution, *cstall* takes its maximum value. The cooperation at the beginning of the execution is scarce. However, each time the algorithm gets stuck and a restart is triggered, *cstall* is reduced by 10% of restart-iterations size, so the algorithm increases their rely on incoming solutions after being reinitialized. If reboots continue to occur, the *cstall* will continue to drop and, thus, the algorithm increases the cooperation between colonies. When the minimum *cstall* is reached (0 iterations), the algorithm returns to the maximum value. In this way, even in multimodal problems, different cooperation degrees are explored in a round-robin fashion.

3 Experimental results

In this section, a series of experiments are shown to assess the value of the strategies proposed in this work.

All the benchmarks used to carry out the experiments reported in this paper are obtained from the W-Model (Weise et al., 2020). The W-Model is a tunable black-box discrete optimization benchmarking problem (BB-DOB) that uses a bit-string representation of the data. The W-Model framework creates different benchmarks by means of different input parameters that modulate different features for the problems. Six challenging benchmarks labeled as B1 to B6 have been defined here. The W-model parameters used to define these challenging problems can be seen in Table 1.

Table 1: W-Model parameters for benchmarks B1 to B6.

Benchmark	Problem size	Neutrality	Epistasis	Ruggedness/Deceptiveness
B1	640	High	130 (81%)	10000 (78%)
B2	720	High	150 (83%)	12000 (75%)
B3	1000	High	200 (80%)	25000 (80%)
B4	640	High	130 (81%)	0
B5	720	High	150 (83%)	0
B6	1000	High	200 (80%)	0

All the experiments were performed at the Galicia Supercomputing Center (CESGA) using the FinisTerrae-III supercomputer. Each FinisTerrae-III node is composed of two Intel Xeon Ice Lake 8352Y CPUs running at 2.2 GHz, with 32 cores per processor (64 cores per node), and 256 GB of RAM. The nodes are connected using an Mellanox InfiniBand HDR 100 Gbps interconnect using a fat-tree topology.

Table 2 lists the different cooperation configurations that have been compared in this work, being *n* the problem size. Note that M1 configuration corresponds to a non-cooperative parallel solution, while M6 corresponds to the cooperation scheme proposed in (González et al., 2022), that is, a full cooperation between colonies. Besides, M7 configuration corresponds to the self-adaptive solution proposed in this paper.

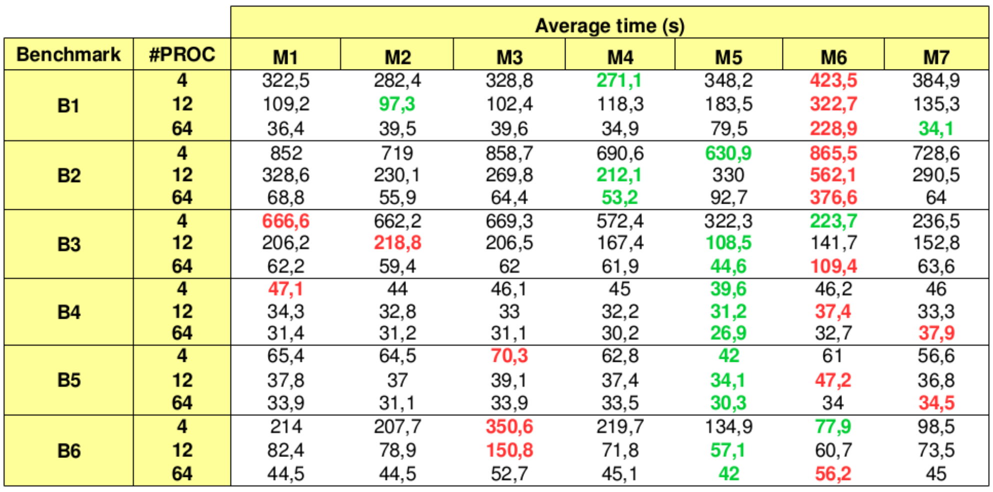
Table 3 shows the preliminary results of the average execution time for each experiment. Some conclusions can be extracted based on the results. A medium-term cooperation like the M4 configuration overcomes the other configurations for 4 colonies in benchmark B1. In a harder problem like B2, there is a need for increasing the cooperation, and results obtained

Table 2: Configurations with different cooperation degrees.

Configuration	$cfreq$	$cstall$	Comments
M1	∞	-	No cooperation at all.
M2	2	$n/25$	1 of 2 receive after stagnation $\geqslant$ problem size/25.
M3	1	$n/10$	All receive after stagnation $\geqslant$ problem size/10.
M4	1	$n/25$	All receive after stagnation $\geqslant$ problem size/25.
M5	1	$n/50$	All receive after stagnation $\geqslant$ problem size/50.
M6	1	0	Full cooperation, all receive all the time.
M7	1	$self-adapted$	Self-adapted at runtime.

for M5 configuration are better than the others when 4 colonies are used. This behavior is restated in B3 problem, bigger and even more harder than B2. The degree of cooperation that achieves the best performance with the same processes is M6. In brief, the larger and harder to solve the problem, the larger the need for cooperation between the colonies. However, these three problems need less cooperation when you increase the processes to 12. In this case, the optimal level decreases from M4 to M2, from M5 to M4 and from M5 to M4, respectively. In other words, the more processes, the lower the level of cooperation required.

Table 3: Comparison of different cooperative configurations in BiPCACO. Average time (in seconds) achieved on experiments. Highlighted in green and red the best and worst times by configuration, respectively.



		Average time (s)						
Benchmark	#PROC	M1	M2	M3	M4	M5	M6	M7
B1	4	322,5	282,4	328,8	271,1	348,2	423,5	384,9
	12	109,2	97,3	102,4	118,3	183,5	322,7	135,3
	64	36,4	39,5	39,6	34,9	79,5	228,9	34,1
B2	4	852	719	858,7	690,6	630,9	865,5	728,6
	12	328,6	230,1	269,8	212,1	330	562,1	290,5
	64	68,8	55,9	64,4	53,2	92,7	376,6	64
B3	4	666,6	662,2	669,3	572,4	322,3	223,7	236,5
	12	206,2	218,8	206,5	167,4	108,5	141,7	152,8
	64	62,2	59,4	62	61,9	44,6	109,4	63,6
B4	4	47,1	44	46,1	45	39,6	46,2	46
	12	34,3	32,8	33	32,2	31,2	37,4	33,3
	64	31,4	31,2	31,1	30,2	26,9	32,7	37,9
B5	4	65,4	64,5	70,3	62,8	42	61	56,6
	12	37,8	37	39,1	37,4	34,1	47,2	36,8
	64	33,9	31,1	33,9	33,5	30,3	34	34,5
B6	4	214	207,7	350,6	219,7	134,9	77,9	98,5
	12	82,4	78,9	150,8	71,8	57,1	60,7	73,5
	64	44,5	44,5	52,7	45,1	42	56,2	45

These results also prove that an intense cooperation like M6 configuration is effective when using few colonies to solve very complicated problems. In those benchmarks, the possibilities of finding a great solution are low, so the need to share potentially successful solutions as soon as possible is high in order to speed up the progress of the search. However, when the number of processes increases, the chances for one of them to find a good solution on its own increases, and cooperation can interfere with this search and damage diversity. If a process frequently accepts foreign solutions, the colony deviates from their own search and ends converging to the same local minima as the rest of the colonies.

These custom-defined benchmarks are not only big in size, but also three of them (B1 to B3) show characteristics that make them very hard to solve. They are *multimodal* and are defined by owing many local minima. In those cases, the colonies cooperation benefits the convergence ratio of the algorithm. Problems B4 to B6 are unimodal, i.e. the search is oriented smoothly to the global minimum. Since the algorithm does not get stuck due to the absence of local minima, it does not need cooperation, and therefore in problems B4 and B5 there is almost no significant difference in times among all methods. In practice, all behave in a non-cooperative fashion. Problem B6, even unimodal, is too big in size and cooperation becomes highly necessary with few processes.

Based on previous experiments, it can be concluded that there are essentially three features of the problem at hand that will determine the degree of cooperation that benefits BiPCACO execution: (1) amount of processes: the smaller the number of processes, the higher the cooperation between colonies must be; (2) multimodality: The higher the bias towards the multimodal landscape, the larger the probability that the algorithm will get stuck; and (3) problem size: the larger the problem, the greater the need for cooperation.

The problem of the previous conclusions is the difficulty for the user to know the features of the problem in advance, and therefore the difficulty of adjusting the configuration parameters before the execution. It is at this point where the self-tuned approach is especially appealing. Moreover, results of Table 3 evidence that it is also competitive when compared with the solution obtained with the best configuration in each problem.

To better illustrate the behavior of the proposed M7 self-tuned cooperation method, logarithmic scale plots for benchmarks B1 and B3 are shown in Figure 1. Those subfigures show the cumulative probability of reaching the optimum related to the execution time, for the 100 runs of each experiment. We choose benchmarks B1 and B3 to show how, despite being both large and difficult problems, in one case cooperation is more beneficial than in the other, however, this is somewhat difficult to know beforehand.

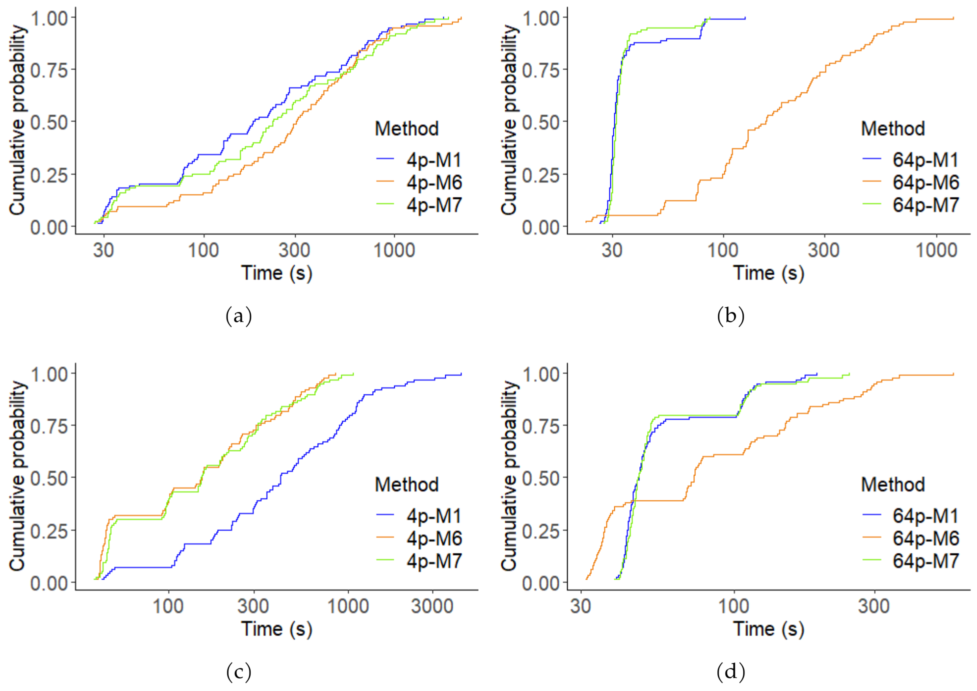


Figure 1: Cumulative probability of reaching the optimum for (a) benchmark B1 using 4 processes, (b) benchmark B1 using 64 processes, (c) benchmark B3 using 4 processes and (d) benchmark B3 using 64 processes, where M1-7 indicates different cooperation schemes.

As it can be seen, for benchmark B1, a configuration with no cooperation (M1) is better than an intensive cooperation (M6). However, a self-tuned cooperation (M7) adapts during runtime and offers a competitive result versus the non-cooperation solution.

On the other hand, for benchmark B3, the best solution turns out to be a configuration with an intensive cooperation (M6) compared to lack of cooperation (M1) for 4 processes. And, on the contrary, the absence of cooperation (M1) is better compared to an intense cooperation (M6) for 64 processes. Besides, the self-tuned solution (M7) is always competitive when compared with the best solution in each situation.

These previous results show that the self-tuned solution, although it may not improve the superior configuration, allows the user to get rid of the responsibility of choosing the most appropriate parameters, making the algorithm reconfigure itself, at execution time, depending on the progress of the search. Results of the self-tuned solution stands out as a competitive alternative.

4 Conclusions

An improved multicolony ACO for binary combinatorial problems is explored in this paper. The goal is to compensate for the main drawback of the algorithm: its trend to get stuck in local minima. Here, a parallel cooperative ACO strategy is evaluated, in which all colonies share promising routes with each other, but only use them for their own search if certain conditions are met. That is, they collaborate only when necessary. This manages to maintain diversity while avoiding rapid convergence to the same local minimum of all colonies. An implementation that adjust the parameters of the cooperative algorithm at runtime has been also evaluated. This improvement allows us to find a good solution for each problem without the need to know in advance the characteristics of the problem in consideration.

This new self-adapted cooperative approach prevents the user from knowing in advance which configuration is the most appropriate for the problem in consideration and saves them the time needed to set the parameters of the new algorithm.

As future work, we consider to improve and refine the cooperative self-adaptation, as well as to explore the hybridization with other metaheuristics.

Acknowledgments

RPR, PGG and RDB acknowledges funding from Grants PID2019-104184RB-I00 and PID2022-136435NB-I00, funded by MCIN/AEI/ 10.13039/501100011033, PID2022 also funded by "ERDF A way of making Europe", EU; Xunta de Galicia and FEDER funds of the EU (Centro de Investigación de Galicia accreditation 2019–2022, ref. ED431G 2019/01; Consolidation Program of Competitive Reference Groups, ref. ED431C 2021/30).

JRB acknowledges funding from the Ministry of Science and Innovation of Spain MCIN / AEI / 10.13039/501100011033 through grant PID2020-117271RB- C22 (BIODYNAMICS).

Authors also acknowledge the Galician Supercomputing Center (CESGA) for the access to its facilities.

Bibliography

E. Alba. *Parallel metaheuristics: a new class of algorithms*, chapter 5–14, pages 105–346. John Wiley & Sons, 2005.

M. Dorigo and T. Stützle. Ant colony optimization: overview and recent advances. *Handbook of metaheuristics*, pages 311–351, 2019.

P. González, R. R. Osorio, X. C. Pardo, J. R. Banga, and R. Doallo. An efficient ant colony optimization framework for HPC environments. *Applied Soft Computing*, 114:108058, 2022.

S.-H. Jang, J.-H. Roh, W. Kim, T. Sherpa, J.-H. Kim, and J.-B. Park. A novel binary ant colony optimization: Application to the unit commitment problem of power systems. *Journal of Electrical Engineering and Technology*, 6(2):174–181, 2011.

S. Kashef and H. Nezamabadi-pour. An advanced aco algorithm for feature subset selection. *Neurocomputing*, 147:271–279, 2015.

M. Kong and P. Tian. A binary ant colony optimization for the unconstrained function optimization problem. In *International Conference on Computational and Information Science*, pages 682–687. Springer, 2005.

T. Stützle, M. Dorigo, et al. Aco algorithms for the traveling salesman problem. *Evolutionary algorithms in engineering and computer science*, 4:163–183, 1999.

T. Weise, Y. Chen, X. Li, and Z. Wu. Selecting a diverse set of benchmark instances from a tunable model problem for black-box discrete optimization algorithms. *Applied Soft Computing*, 92: 106269, 04 2020.