Multi-GPU accelerating strategies of Ant Colony Optimization algorithms using Rank Based and Strong Elitist versions

Andrés I. Ávila
Departamento de Ingeniería Matemática
Universidad de La Frontera
Temuco, Chile
ORCID 0000-0001-7180-1479

Juan Manuel Aedo
Depto. de Ing. Matemática
Universidad de La Frontera
Temuco, Chile
ORCID 0009-0007-6037-3481

Mariela González-Flores
Depto. Cs. Nat. y Tecnología
Universidad de Aysén
Coyhaique, Chile
ORCID 0000-0003-4266-7949

***Abstract*—The most used problem to compare these metaheuristics is the classical Travelling Salesman Problem (TSP) on a network with sizes from tens to millions of nodes. Nature-based metaheuristics are the main source of optimization techniques to solve this problem. Although there is a zoo of possibilities, Ant Colony Optimization (ACO) is still one of the most efficient, parallel, and simple techniques to implement. The pheromone evaporation rate, α, β, and ant number M are the four parameters fitted for finding the best solutions. Candidate list and the use of a source solution are efficient strategies to optimize large problems, but there are other strategies to improve quality solutions such as local search strategies. Among the local search techniques, k-opt search has proved to be very efficient to deal with path-crossing. The initial route is split into k sub-routes connected in $(k-1)!2^{k-1}$ ways. Thus, 3-opt is an efficient strategy balancing complexity and precision. Moreover, the best α and β are the result of the used strategy. Finally, the solution is improved by accelerating through parallel versions with multi-GPUs.**

In this article, four ACO algorithms were developed using two variations (Rank Based 3-opt and Strong Elitist 3-opt), each one with two strategies (candidate list and restricted). To test the algorithms, TSPLIB95 was used with test problems between $N = 51$ and $4,461$ nodes in a server with 4 RTX A4000 GPUs. After improving the algorithms and parallelization, the parameters are tuned to get the highest performance improving the local minimum search. Statistical analysis of repetitions shows good stability of the chosen metaheuristics.

***Index Terms*—Multi-GPU, Ant Colony Optimization algorithms, metaheuristics, local search, high performance.**

I. Introduction

High performance computing is the main framework for approximating solutions to complex combinatorial problems through metaheuristics. On one hand, there is a hardware race to offer more powerful tools, where GPUs are one of the most cost-effective solutions for massive parallel problems such as population-based metaheuristics. On the other hand, massive simulations help to understand how algorithms work on these architectures and to develop new strategies to avoid memory and computing bottlenecks. Multicore architectures were the first approach to parallellize discrete optimization problems and later the emerge of GPUs offered a better choice especially for routing problems [1], [2]. Programming languages for GPUs, like CUDA, OpenCL, HIP, and SYCL allowed increased possibilities for developing algorithms and different accelerated architecture platforms appeared as CPU on GPU, CPU on APU, or GPU on APU tested with the Ant Colony Optimization algorithm [3]. More recently, Menezes et al. [4] also worked on a single GPU developing strategies for paralellizing the pheromone matrix and comparing the coarse-grain and fine-grain ACO.

Among the main combinatorial problems, TSP models several applications such as logistics, microchip manufacture, and any problem requiring finding a Hamiltonian cycle on graphs. Moreover, the increase in delivering services and energy-saving solutions require fast and precise solutions. Applications like mobile surveillance systems [5], green vehicle routing like [6], and large-scale problems such as the Santa Claus Challenge 2020 [7] need more powerful algorithms and hardware requirements.

Since its origin, ACO was developed to solve TSP [8]. The ants find shortest-path solutions between food source and nest through pheromone trails. Based on a Monte Carlo approach, a high number of random ants gets a better solution but at a high computational cost. To reduce this cost, several parallel strategies were developed based on classical parallel algorithm techniques, such as the master-slave model, coarse-grain or fine-grain models, the multicolony model, among others [9]. CUDA opened a new way to develop well-suited parallel ACO algorithms, improving tour construction and pheromone update with a single GPU [10], [11] and parallelizing ants or developing multiple colonies, including MAX-Min approach with two GPUs [12] both with over 20x speedups in problems with 2,000 nodes. Since ACO complexity is $O(iter \times nodes^3)$, more efficient parallel versions were developed based on the GPU architecture by reducing local/global pheromone updates and using a candidate set with warp size with similar speedups and size problems [13]. RAM GPU size is one of the bottlenecks to large problems. Dynamic load balancing on several GPU generations [14] and dynamic kernel strategies with larger RAMs allowing 4,000 nodes [15] were developed

979-8-3503-8160-3/23/$31.00 ©2023 IEEE
DOI 10.1109/SBAC-PADW60351.2023.00020

on a single GPU. More recently, new roulette strategies and tuning with GPU parameters (warp) also with nodes near to 4,000 [16]. To improve solutions, MAX-MIN Ant Systems together with 3-opt strategy allow us to reduce the complexity to $O(iter \times nodes^2)$ and speed up the solutions, where two GPU generations are compared, showing the importance of the hardware properties and increasing the node number up to 12,000 nodes [17]. Recently, using a static-dynamic balanced candidate set strategy, new algorithms solved large problems (up to 28,000 nodes) but reducing the ant number [18].

In this article, four ACO algorithms were developed using two variations (Rank Based 3-opt and Strong Elitist 3-opt), each one with two strategies (candidate list and restricted). To test the algorithms, TSPLIB95 was used with test problems between $N = 51$ and $4,461$ nodes in a server with 4 RTX A4000 GPUs. The parallelization of the algorithms was done by using the *CUDA Thrust* library to improve the sorting algorithm. After tuning parameters for speeding up the algorithms, the optimal parameters were $\alpha = 0.3$, $\beta = 5.0$, evaporation rate 0.01, and ant maximum number $M = 4,096$. Solutions between $0.0\% - 1.6\%$ for problems below 561 nodes and $1.15\% - 3.00\%$ for problems between $1,291$ to 446 nodes. Also, the standard deviation of the repetitions were computed showing the stability of the algorithm w.r.t the stochastic nature. Also, execution times were obtained between 1.1 to 180.0 sec giving a 4x increase in ant number and decreasing the error by half.

II. ACO AND PHEROMONE UPDATE STRATEGIES

A. Basic ACO algorithm

A pseudo-code of ACO algorithm is shown in Algorithm 1.

Algorithm 1 Basic Ant Colony Optimization (ACO)

Require: $\alpha, \beta, \rho, N, M, I$, tsp95libfile
Ensure: optimal route, optimal cost
1: BestSolution ← InitialSolution(tsp95libfile)
2: Initialize_pheromones_and_heuristics(BestSolution)
3: **while** Iteration < I **do**
4: **for** ants<M **do**
5: Tour_Construction($\alpha, \beta, route_{ants}$,N)
6: **end for**
7: Update_pheromone_matrix(ρ,$route_{ants}$)
8: Update_the_best_solution($route_{ants}$)
9: **end while**

The input parameters related with the complexity are the *ant number* M, *iteration number* I, and the *nodes number* N. Each ant builds its own tour using Tour_Construction() selecting $j \in J$ an *unvisited* node and defining a probability $P_{i,j}$, which needs the global parameters $0 < \alpha \leq 1$ and $0 \leq \beta \leq 5$, $\tau_{i,j}$ the amount of pheromone on the path from node i to node j, and $\eta_{i,j}$ the inverse of the distance between nodes i and j. The formula is given by (1)

$$P_{i,j} = \frac{\tau_{i,j}^{\alpha} \eta_{i,j}^{\beta}}{\sum_{j \in J} \tau_{i,j}^{\alpha} \eta_{i,j}^{\beta}}. \tag{1}$$

In Algorithm 2, Tour_Construction() starts each route of the ants at a random node of the graph and then chooses the next node with select_next_node(), which uses (1) to distribute the probabilities of visiting each node.

Algorithm 2 Tour_Construction()

Require: $\alpha, \beta, route_{ants}, N$,
Ensure: $route\_ants$
1: $route\_ants[0]$ ← select_initial_node()
2: visited_list ← $\{route_{ants}[0]\}$
3: j=0
4: **while** j < N-1 **do**
5: $route_{ants}$[j+1] ← select_next_node(α, β,visited_list)
6: Update_visited_list(visited_list,$route_{ants}[j+1]$)
7: j++
8: **end while**

The pheromones $\tau_{i,j}$ are updated using (2) and (3)

$$\tau_{i,j} \leftarrow \tau_{i,j}(1-\rho) + \sum_{n=1}^{M} Q \Delta \tau_{i,j}^{n}, \tag{2}$$

and

$$\Delta \tau_{i,j}^{n} = \begin{cases} \frac{1}{L_n} & \textit{if ant n chooses edge (i,j),} \\ 0 & otherwise, \end{cases} \tag{3}$$

where ρ is the evaporation rate, Q is a positive constant, $\Delta \tau_{i,j}^{n}$ is the pheromone deposited by ant n on i,j edge, and L_n is the tour length for each ant. At each iteration,the algorithm saves a *best solution* if some improvement was found by a minimal route.

Since its most basic version, improvements have been proposed for the algorithm to find better solutions, some of these are proposed in [19]. The following two versions for updating pheromones were selected: Strong Elitist (SE) and Rank Based (RB) Ant Systems. These strategies profit from the previous ant information on best routes by recording some critical information and there are not been tested using the classical local search and the new restricted search.

B. Strong Elitist strategy (SE)

Instead of using (2) update, it is only added the ant pheromone passing through the best global solution (4),

$$\tau_{i,j} \leftarrow \tau_{i,j}(1-\rho) + e \Delta \tau_{i,j}^{best\_sol}, \tag{4}$$

which must be saved as the algorithm is executed. e is the number of elite ants and $\Delta \tau_{i,j}^{best\_sol}$ is the pheromone deposited by ant n on edge i,j in the best route.

C. Rank Based strategy (RB)

Instead of selecting only one ant, a fixed number of ants $u = best\_ants$ are selected, sorted according to the length of its tours, and add pheromone to the corresponding edges. In addition, a weight is assigned to their pheromone according

to a ranking of solutions in such a way that the ants with the shortest tour add the greatest amount of pheromone (5)

$$\tau_{i,j} \leftarrow \tau_{i,j}(1-\rho) + u\Delta\tau_{i,j}^{global\_sol} + \sum_{n=1}^{u-1}(u-n)\Delta\tau_{i,j}^{n} \quad (5)$$

D. 3-opt move

Link swap is one of the main techniques to improve best routes [20]. In case of three links, the *3-opt* algorithm takes a tour and randomly divides it into 3 sub-tours that are reconnected in 7 different ways. Then, it is evaluated if there is a positive change in the sum of the distances. If there is a positive change in any of the 7 cases, the route changes to the smallest of these positive changes.

E. Parameters

The following table shows the most used parameters:

TABLE I
ACO PARAMETERS

Parameter		Values
α	pheromone factor	[0,1]
β	heuristic factor	[1,5]
ρ	evaporation rate	[0,1]
cl_list	candidate list size	8,80
$3opt\_iterations$	iterations for $3-opt$	100
e	elite ant number	10
$best\_ants$	best ant number	10
Q	amount of pheromone per ant	1
τ_0	initial pheromone	$\frac{best\_ants}{aux\_sol}$
min_new_edges	minimum number of edges for using the source solution	8

Parameters aux_sol is selected from the greedy solution choosing the nearest neighbor of each node to build a route.

III. ACCELERATION STRATEGIES

A. ACO algorithms and 3-opt

Rank Based and Strong Elitist variations of the ACO algorithms were made together with the local search algorithm 3-opt. This search consists of taking the paths generated by the ants and improving each of them by using 3-opt such that the ants add the pheromones to the improved paths. In the Algorithm 1, the *3-opt*($route_{ants}$) step is added after step 5 and step 7 is replaced by Update_pheromone_matrix(ρ,$route_{ants}$) by the new Update_pheromone_matrix_RB/SE(ρ,$route_{ants}$) using (4) or (5) depending on the ACO version.

Algorithm 3 RB/SE3-opt

Require: $\alpha, \beta, \rho, N, M, I$, tsp95libfile
Ensure: optimal route, optimal cost
1: BestSolution ← InitialSolution(tsp95libfile)
2: Initialize_pheromones_and_heuristics(BestSolution)
3: **while** Iteration < I **do**
4: **for** ants<M **do**
5: Tour_Construction($\alpha, \beta, route_{ants}$,N)
6: 3-opt($route_{ants}$)
7: **end for**
8: Update_pheromone_matrix_RB/SE(ρ,$route_{ants}$)
9: Update_the_best_solution($route_{ants}$)
10: **end while**

B. Restricted Rank Based/Strong Elitist + 3-opt strategies

The Restricted (R) strategies are based on the idea of a $source\ solution$ proposed in [23]. When choosing an $edge$ to construct ant tour, verify that if the chosen $edge$ is in the $source\ route$, then go to the next node. If the $edge$ does not belong to the $source\ route$, it is added as a $new\ edge$. In this version, a maximum number of $new\ edges$ is fixed. Once the ant passes the limit of $new\ edges$, instead of deciding using the probability to choose the next node, it will fill in with the current $edges$ in the $source\ route$. This modification controls the number of $new\ edges$ and decrease the time on each ant. This $source\ route$ is the best $route$ for the iterations so far. The algorithm is shown in Algorithm 4, where S_S is the source solution and $succ()$ is the successor of the current node in S_S.

Algorithm 4 Tour_Construction_R

Require: $\alpha, \beta, route_{ants}, N$,
Ensure: $route_{ants}$
1: $route_{ants}[0]$ ← select_initial_node()
2: visited_list ← $\{route_{ants}[0]\}$
3: j=0
4: **while** j < N-1 **do**
5: $route_{ants}$[j+1] ← select_next_node(α, β,visited_list)
6: Update_visited_list(visited_list,$route_{ants}[j+1]$)
7: j++
8: **if** $(route_{ants}[j-1], route_{ants}[j]) \notin S\_S$ **then**
9: num_new_edges++
10: **end if**
11: **if** num_new_edges≥ min_new_edges **then**
12: copy_node=succ(S_S,$route_{ants}[j]$)
13: **while** copy_node $\notin route_{ants}$ **do**
14: $route_{ants}[j+1]$ ← copy_node
15: copy_node ← succ(S_S,$route_{ants}[j+1]$)
16: j++
17: **end while**
18: **end if**
19: **end while**

C. Restricted Rank Based/Strong Elitist + 3-opt strategies with Candidate List

The strategy Candidate List (CL) is based on limiting the number of neighbors that each ant considers to select the next node. In this case, it only scans for a fixed number of nearest neighbors. In the case of none of the list is available, it chooses the next available nearest node [24]. This modifies the probabilities given in (1), where the set J changes to the set of unvisited nearest neighbors.

D. GPUs and distributed work

In GPUs-accelerated applications, the sequential part of the workload runs on the CPU, which is tuned for single-threaded performance, while the computationally intensive part of the application runs on thousands of GPUs cores. To improve the execution time, the tour construction was parallelized. Each ant is a thread using kernel configuration $\frac{M}{8}$ blocks and 2 threads. In the cost computation and local search, it used the kernel configuration $\frac{M}{4}$ blocks and 1 thread. In updating the pheromone, it used the kernel configuration N blocks and 1 thread, where each ant updates N edges in parallel. The random numbers were generated on the GPU using the function cuda_random_uniform() from $cuRAND$ library.

To evaporate the pheromone and compute the combination between the pheromone matrix and the matrix of the inverse of the distance, a special number of blocks and threads are used in such a way that 32 threads are used. the smallest integer greater than $N^2/32$ is considered.

E. Distributed work in multi-GPU

To use large ant numbers without increasing the computational time, the construction of the ant tour, the ant cost, and the local 3opt search is distributed in m GPUs with $\frac{M}{m}$ ant per GPU, as is shown in Fig 1 with $m = 4$. Each GPU uses a local data structure for constructing ant tour, computing the tour cost, and performing the local search. Next, all tours and costs are collected in one common data structure to sort tours and finally update the pheromone matrix.

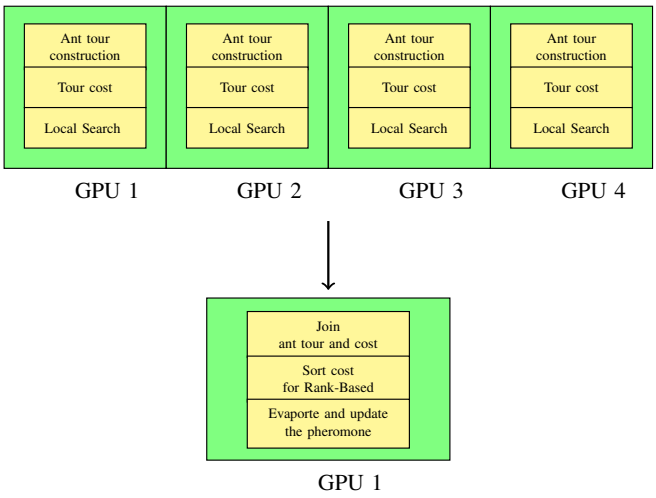

Fig. 1. GPU processes distribution diagram

Ant tour construction and local search are the processes using more computation time. Then, these processes are performed in parallel for each ant by a large number of ants distributed on 4 GPUs. Fig. 1 shows how the workload is distributed on 4 GPUs and how the routes and costs are joined in the master GPU. Thanks to CUDA's thrust library and its sort_by_key() function, the sorting cost algorithm is executed by a parallelized sorting algorithm on GPUs. Then, the master GPU evaporates and updates the pheromone matrix and this matrix is distributed throughout the 4 GPUs for the next iteration. This process is repeated until the maximum number of iterations is reached.

IV. Results

A. Setup

All algorithms were executed in a 4-GPU architecture with the following hardware: (i) CPU: Intel(R) Xeon(R) Silver 4310 CPU @ 2.10GHz RAM: 256GB, (ii) GPU: NVIDIA RTX A4000 RAM:16,376MB. For programming on NVIDIA GPUs, CUDA version 11.7 was used https://developer.nvidia.com/cuda-zone

The following TSPLIB95 problems were used: *eil51*, *berlin52*, *gr120*, *ch150*, *kroA200*, *pcb442*, *pa561*, *d1291*, *d1655*, *d2103*, *fl3795*, *fnl4461*, where the numbers in the name of each instance shows the node size N.

The relative error metric (6)

$$ER\% = \frac{100(x_{BKS} - \bar{x})}{x_{BKS}} \tag{6}$$

was used to compare the performance of the algorithms for different problems, where x_{BKS} is the current best known solution and $\bar{x}$ is the solution of the algorithm.

B. Preliminary analysis of RB3-opt and RB3-optR

A preliminary analysis was performed to analyze the effect of the strategy of restricting exploration through the $source\ solution$. It was compared the algorithms RB3-opt and RB3-optR, both RB versions with $\alpha = 1$ and $\beta = 5$, with the PACO3 in [22]. This algorithm was tested in several datasets giving the possibility to compare our versions with it. This comparison considered the average solution, which results are shown in Table II. Best solutions are highlighted in bold face.

TABLE II
AVERAGE SOLUTION FOR THE ALGORITHMS USING 3,000 ITERATIONS FOR RB3-OPT AND 1,200 ITERATIONS FOR RB3-OPTR

Instance	BKS	PACO-3-opt [22]	RB3-opt	RB3-optR
eil51	426	**426.35**	426.45	426.80
berlin52	7542	**7542.00**	**7542.00**	**7542.00**
st70	675	**677.85**	678.80	680.70
eil76	538	538.85	539.00	**538.00**
rat99	1211	1217.10	1213.00	**1212.30**
kroA100	21282	21326.80	21336.75	**21322.30**
eil101	629	630.55	635.20	**629.60**
gr120	6942	-	7049.10	**6970.20**
ch150	6528	**6563.95**	6566.40	6565.60
kroA200	29368	29644.50	29571.95	**29475.50**
pcb442	50778	-	**51405.70**	51422.10
pa561	2763	-	**2816.35**	2825.60

Table II shows that RB3-optR is the algorithm that obtains most of the solutions closest to the **Best Known Solution (BKS)**. In the cases in which some of the other algorithms obtain better solutions, RB3-optR obtains solutions close to them.

Fig. 2 shows that RB3-optR approaches the optimum faster than compared to RB3-opt requiring a lower iteration number. In this case RB3-optR required 758 (the green star in Fig. 2) iterations while RB3-opt 2,893 iterations.

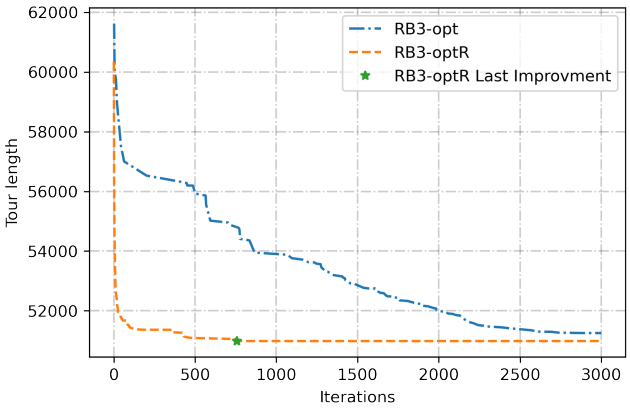

Fig. 2. Iterations of RB3-opt and RB3-optR for the pcb442 instance.

It was observed that when restricting the exploration to the source solution, it improved the execution time as shown in Table III including the number of required iterations and the quality of the solutions.

TABLE III
EXECUTION TIMES IN SECONDS, WITH 3,000 ITERATIONS FOR RB3-OPT AND WITH 1,200 ITERATIONS FOR RB3-OPTR

Instance	RB3-opt	RB3-optR
eil51	2.871	**1.11**
berlin52	2.676	**1.05**
gr120	4.400	**1.90**
ch150	7.698	**2.63**
kroA200	10.734	**4.05**
pcb442	42.930	**8.31**
pa561	71.940	**12.37**

C. α and β value effect in performance

Experiments were carried out for different levels of α and β using 1,200 iterations and running 10 times per instance to obtain the average iteration time and average solution. These experiments were performed for both instances $pcb442$ shown in Figure 3 and 4, and $pa561$ shown in Figure 5 and 6.

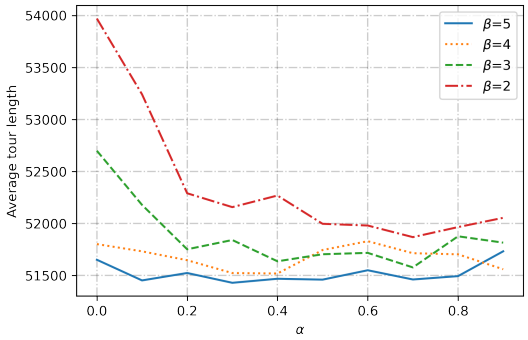

Fig. 3. α and β effect in SE3-optR Average cost in the pcb442, with $M = 4,096$ and $\rho = 0.01$.

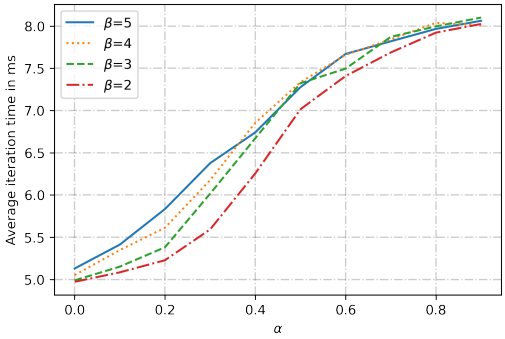

Fig. 4. α and β effect in SE3-optR iteration time, with $M = 4,096$ and $\rho = 0.01$.

In both versions of the ACO algorithms, it was found that as α approaches to 1 the iteration time increases, while the solutions get worse as α approaches to 0. It is observed that the best solutions are obtained for $\alpha \geq 0.3$. In Figure 3, it is shown that $\beta = 5$ gets the best performance. The same behavior was observed for the candidate list algorithms as shown in Figure 7 and 8. To keep a good solution quality and low iteration time,

$\alpha = 0.3$ and $\beta = 5$ were chosen as the optimal parameters for both versions.

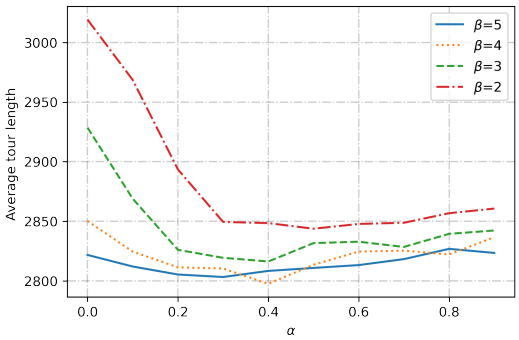

Fig. 5. α and β effect in RB3-optR Average cost in the pa561 with $M = 4,096$ and $\rho = 0.01$.

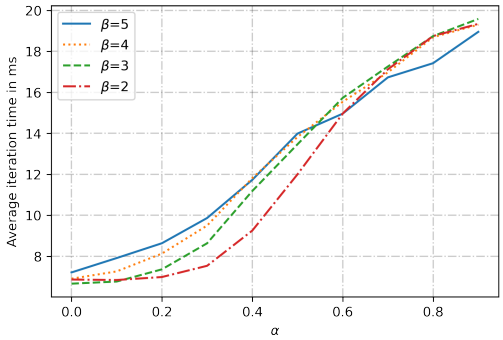

Fig. 6. α and β effect in RB3-optR iteration time in the pa561 with $M = 4,096$ and $\rho = 0.01$.

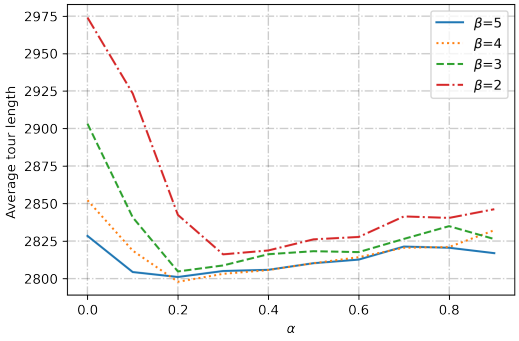

Fig. 7. α and β effect in RB3-optRCl Average cost in the pa561 with $M = 4,096$ and $\rho = 0.01$.

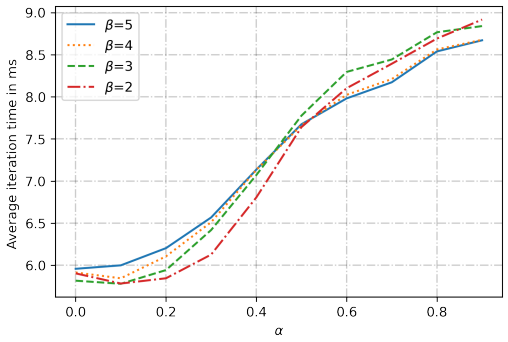

Fig. 8. α and β effect in RB3-optRCl iteration time in the pa561 with $M = 4,096$ and $\rho = 0.01$.

D. Ant number effect

A larger ant number should improved the solutions by searching a large search space. For this purpose, the algorithm was run for $M = 2^{k+8}$, $k = 1, 2, 3, 4, 5$ spread over the 4 GPUs, where each GPU has $\frac{M}{4}$ ants. Results for RB3-optR in $pa561$ instance are shown in Figure 9 and 10 using the average of 10 repetitions with $\rho = 0.01$, $\alpha = 0.3$, and $\beta = 5$.

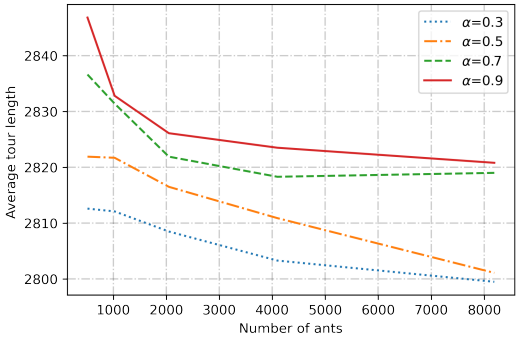

Fig. 9. Effect of M on the solutions, with $\beta = 5$ and $\rho = 0.01$ for RB3-optR in $pa561$.

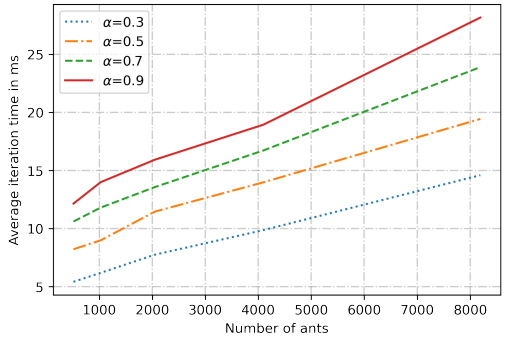

Fig. 10. Effect of M on the iteration time in ms, with $\beta = 5$ and $\rho = 0.01$ for RB3-optR in $pa561$.

It is observed that as the number of ants increases, the solution approaches to the optimum and the iteration time increases. To keep the iteration times low and best solutions, $M = 4,096$ was selected.

TABLE IV
ANT TOUR CONSTRUCTION EXECUTION TIME IN MS FOR RB3-OPTR WITH PCB442.

	Ant number			
GPU number	1024	2048	4096	8192
0	121.27	233.03	492.35	1007.18
1	4.10	5.54	9.00	16.07
2	3.78	4.44	6.07	9.83
4	3.68	4.04	4.92	7.01

TABLE V
ANT TOUR CONSTRUCTION EXECUTION TIME IN MS FOR RB3-OPTR WITH PA561.

	Ant number			
GPU number	1024	2048	4096	8192
0	125.95	252.30	497.93	1023.40
1	6.06	8.28	13.69	23.74
2	5.59	6.80	9.35	14.94
4	5.58	6.52	7.60	10.25

To study scalability, the CPU time (0 GPU), 1,2, and 4 GPUs are compared in Table IV and Table V. The two largest test examples were considered (pcb442 and pa561). By using GPUs, the speedups are 26x and 19x. In addition, the importance of more GPUs for large ant number compared with the small ant number is clear.

E. Evaporation rate effect

The evaporation rate regulates the relevance of the previous iterations for the current iteration. An evaporation rate $\rho = 1$ implies that the previously deposited pheromone has no relevance, while $\rho = 0$ implies that the previously deposited pheromone has the same relevance as the one deposited in the current iteration. Repeating 20 times each run, it is obtained the results of the restricted algorithms with candidate list (RB/SE3-optRCl) in Table VI and without candidate list (RB/SE3-optR) in Table VII with different levels of evaporation rate. To determine ρ giving the best solutions, the selected values $\alpha = 0.3$, $\beta = 5$, and $M = 4,096$ were used.

TABLE VI
EVAPORATION RATE EFFECT IN AVERAGE SOLUTION AND LAST IMPROVEMENT ITERATION FOR PA561 INSTANCE AND RB3-OPTRCL

ρ	Avg Cost	Avg LII
0.50	2831.75	159.15
0.10	2814.25	312.80
0.05	2806.15	503.25
0.01	**2802.30**	954.35

It was noticed that as the evaporation rate decreased, the solutions improved, but require more iterations, as shown by the Last Iteration Improved (LII). To keep the number of iterations at $1,200$ for problems below 561 nodes, the evaporation rate was set to $\rho = 0.01$.

TABLE VII
EVAPORATION RATE EFFECT IN AVERAGE SOLUTION AND LAST IMPROVEMENT ITERATION FOR PCB442 INSTANCE AND RB3-OPTR

ρ	Avg Cost	Avg LII
0.50	51597.25	194.30
0.10	51343.25	668.45
0.05	51382.60	705.20
0.01	**51208.00**	961.60

F. Performance of the acceleration strategies

A performance comparison of the Restricted (RB/SE3-optR) and Restricted with candidate list (RB/SE3-optRCl) algorithms was done for instances with less than 1,000. These algorithms were run 20 times.

TABLE VIII
RELATIVE ERROR OF THE SOLUTIONS

Instance	RB3-optR	RB3-optRCl	SE3-optR	SE3-optRCl
eil51	0.19%	**0.11**%	0.20%	0.16%
berlin52	**0.00**%	**0.00**%	**0.00**%	**0.00**%
st70	0.65%	**0.29**%	0.73%	0.38%
eil76	**0.00**%	**0.00**%	**0.00**%	**0.00**%
rat99	**0.00**%	**0.00**%	**0.00**%	**0.00**%
kroA100	0.04%	**0.00**%	0.15%	**0.00**%
eil101	**0.02**%	0.06%	0.10%	0.04%
gr120	0.49%	**0.30**%	0.40%	0.48%
ch150	0.51%	**0.49**%	0.58%	0.53%
kroA200	0.21%	0.22%	0.28%	**0.20**%
pcb442	1.08%	**0.88**%	1.27%	1.24%
pa561	**1.39**%	1.49%	1.60%	1.64%

TABLE IX
AVG SOLUTION

Instance	BKS	RB3-optR	RB3-optRCl	SE3-optR	SE3-optRCl
eil51	426	426.80	**426.45**	426.85	426.70
berlin52	7542	**7542.00**	**7542.00**	**7542.00**	**7542.00**
st70	675	679.40	**676.95**	679.95	677.55
eil76	538	**538.00**	**538.00**	**538.00**	**538.00**
rat99	1211	**1211.00**	**1211.00**	**1211.00**	**1211.00**
kroA100	21282	21291.10	**21282.00**	21314.85	**21282.00**
eil101	629	**629.15**	629.40	629.60	629.25
gr120	6942	6975.75	**6963.05**	6970.00	6975.25
ch150	6528	6561.45	**6559.70**	6565.75	6562.90
kroA200	29368	29429.60	29434.05	29449.35	**29425.35**
pcb442	50778	51323.90	**51225.10**	51424.90	51405.95
pa561	2763	**2801.30**	2804.30	2807.10	2808.20

There is no clear winner in terms of relative errors, but there is a difference between problems with less than 200 nodes and with more than 200 nodes. Since all 4 algorithms obtain a relative error of less than 1%, while for problems with more than 200 nodes ($pcb442$ and $pa561$) there is an increase in errors of more than 1% for the algorithms. These results of relative error and average solution are shown in Table VIII and Table IX, respectively. In terms of iteration time, there is also a difference between less than 200 nodes and more than 200 nodes. For less than 200 nodes, the Restricted algorithms

(RB/SE3-opt) have smaller iteration times, while above 200 nodes the Restricted versions with candidate lists (RB/SE3-optCl) are the ones with smallest iteration times. These results are shown in Table X. The effect of the node number in the iteration time is observed, which is clearly slow for cases pcb442 and pa561. The time trend is linear for large problems.

TABLE X
ITERATION TIME IN MS

Instance	RB3-optR	RB3-optRCl	SE3-optR	SE3-optRCl
eil51	0.923	1.705	**0.893**	1.703
berlin52	0.880	1.721	**0.856**	1.704
st70	**1.068**	1.868	1.082	1.843
eil76	**1.201**	1.897	1.258	1.927
rat99	1.565	2.424	**1.511**	2.566
kroA100	**1.400**	2.509	1.436	2.451
eil101	**1.507**	2.511	1.536	2.524
gr120	**1.587**	2.709	1.663	2.769
ch150	**2.196**	3.511	2.406	3.282
kroA200	3.379	3.989	**3.356**	4.036
pcb442	6.927	**5.915**	7.835	6.660
pa561	10.310	7.090	10.785	**7.054**

The standard deviation of the solutions is shown in Table XI.

TABLE XI
STANDARD DEVIATION OF THE SOLUTIONS

Instance	RB3-optR	RB3-optRCl	SE3-optR	SE3-optRCl
eil51	0.40	0.50	**0.36**	0.46
berlin52	**0.00**	**0.00**	**0.00**	**0.00**
st70	2.78	**2.67**	2.89	2.84
eil76	**0.00**	**0.00**	**0.00**	**0.00**
rat99	**0.00**	**0.00**	**0.00**	**0.00**
kroA100	27.30	**0.00**	118.76	**0.00**
eil101	**0.36**	0.92	1.59	0.89
gr120	22.34	**21.35**	23.38	25.99
ch150	8.24	**3.84**	12.03	11.46
kroA200	40.48	42.23	**36.69**	42.88
pcb442	312.72	**235.58**	332.35	316.71
pa561	9.77	**8.14**	12.09	10.80

For the medium scale instances with problems $d1291$, $d1655$, $d2103$, $fl3796$ and $fnl4461$, the algorithms with the lowest iteration times for larger instances were used with $3,000$ iterations, in this case the RB/SE3-optRCl algorithms. These algorithms were run 20 times and the results are shown in Table XII and Table XIII.

TABLE XII
RELATIVE ERROR AND STANDARD DEVIATON FOR MEDIUM-SCALE INSTANCES FOR THE TWO VERSIONS WITH BEST ITERATION TIME

Instances	RB3-optRCl	RB3-optRCl STD	SE3-optRCl	SE3-optRCl STD
d1291	3.00%	292.95	**2.79%**	406.41
d1655	2.33%	183.73	**2.32%**	180.76
d2103	**1.15%**	69.42	1.25%	61.59
fl3796	**1.19%**	63.64	1.36%	50.33
fnl4461	**2.02%**	257.25	2.65%	264.08

The Strong Elitist (SE3-optRCl) version is the one with best solutions for the d1291 and d1655 instances. As the instance size increases above $2,103$ nodes, the Rank Based (RB3-optRCl) version shows the best solutions.

TABLE XIII
ITERATION TIME IN MS FOR MEDIUM SCALE INSTANCES

Instances	RB3-optRCl	SE3-optRCl	PD
d1291	**16.379**	18.800	12.88%
d1655	**20.513**	23.787	13.75%
d2103	**27.386**	36.433	24.83%
fl3796	**50.560**	52.035	2.83%
fnl4461	**86.168**	96.061	10.30%

In terms of iteration times, it was observed that the RB version (RB3-optRCl) is the winner for larger instances. Although for instances of less than $1,000$ nodes the iteration times did not differ greatly for each strategy (with candidate list and without candidate list) as is shown in Table X, for problems of more than $1,000$ nodes the difference in iteration times is in the range of 2.83% and 24.83% percentage difference (PD).

CONCLUSIONS AND FUTURE IMPROVEMENTS

Using multi-GPUs architectures, new ACO algorithms are developed to solve TSP improving the possibilities to test parameters and speedup strategies. Rank Based and Strong Elitist pheromone updating decreases the computational operations and decreases the time without sacrificing optimal search. To speed up optimal search, 3-opt strategy improves local searches. Moreover, restricting source solutions and using a candidate list, four quick algorithms were presented and tested on problems from 51 to $4,461$ nodes. The algorithms can be ported to other GPU programming languages since the main libraries are the random generator and sort libraries. For example, in the case of random generator, the GPU programming languages have libraries like RandomCL(OpenCL), rocRAND(HIP), and SYCL-PRNG.

First, it is noticed that the restricted version was 3x faster, locating most of the best optimal solutions for problems between 51 and 561 nodes. Next, the best choice for $\alpha = 0.3$ and $\beta = 5$ keeping the best solutions for the SE3-optR and RB3-optR is selected, which behave similarly. Notice that as α increases, it also increases the computational time. On the opposite, β has no main influence in computing time. Next, the ant number was increased to get better solutions, where $M = 4,096$ was chosen. The evaporation rate was studied for two different setups: RB3-optR with pa561 and RB-3optR with pcb442 showing that the best rate is $\rho = 0.01$ since better solutions are found. Finally, the best algorithm was RB3-optRCl, which shows low relative error and best average solutions, followed by SE3-optRCl. The standard deviation shows that ACO algorithms are similar and the behavior depends on the search space given by some kind of distance distribution of each problem. Finally, for problems between 1,291 and 4,461 nodes, the best algorithms show good performance.

Since GPUs are RAM-limited, large problems cannot be solved by using them. To solve large TSPLIB95 datasets (above 4461), multicore architectures are required. Currently, the authors are studying parallelization of C codes in shared-memory systems with large RAM allowing to test over 100,000 nodes as in [23].

ACKNOWLEDGEMENT

M. I Gonzalez-Flores thanks support from FONDECYT Postdoctoral (Chile) under project ANID 3220650. Powered@NLHPC: This research was partially supported by the supercomputing infrastructure of the NLHPC (ECM-02). Juan Aedo and Andrés I. Ávila thanks Dirección de Cooperación International at Universidad de La Frontera.

REFERENCES

[1] A. Brodtkorb, T. R. Hagen, C. Schulz, and G. Hasle, "GPU computing in discrete optimization. Part I: Introduction to the GPU", EURO journal on transportation and logistics, vol.2, pp. 129-157, May 2013.

[2] C. Schulz, G. Hasle, A. R. Brodtkorb, and T. R. Hagen, "GPU computing in discrete optimization. Part II: Survey focused on routing problems", EURO journal on transportation and logistics, vol. 2, pp. 159-186, May 2013.

[3] G. D. Guerrero, J. M. Cecilia, A. Llanes, J. M. García, M. Amos, and M. Ujaldón, "Comparative evaluation of platforms for parallel Ant Colony Optimization", The Journal of Supercomputing, vol. 69, pp. 318-329, March 2014.

[4] B. A. Menezes, H. Kuchen, H. A. A. Neto, and F. B. de Lima Neto, "Parallelization strategies for GPU-based ant colony optimization solving the traveling salesman problem". In 2019 IEEE Congress on Evolutionary Computation (CEC) (pp. 3094-3101). IEEE.

[5] K. Saleem, and I. Ahmad, "Ant Colony Optimization ACO Based Autonomous Secure Routing Protocol for Mobile Surveillance Systems", Drones, vol. 6(11), pp. 6110351, November 2022.

[6] Y. Li, H. Soleimani, and M. Zohal, "An improved ant colony optimization algorithm for the multi-depot green vehicle routing problem with multiple objectives", Journal of cleaner production, vol. 227, pp. 1161-1172, August 2019.

[7] R. Mariescu-Istodor and P. Fränti, "Solving the Large-Scale TSP Problem in 1 h: Santa Claus Challenge 2020", Frontiers in Robotics and AI, vol. 8, pp. 689908, October 2021.

[8] M. Dorigo and L. M. Gambardella, "Ant colonies for the travelling salesman problem", Biosystems, vol. 43, pp. 73-81, July 1997.

[9] M. Pedemonte, S. Nesmachnow and H. Cancela, "A survey on parallel ant colony optimization", Applied Soft Computing, vol. 11, pp. 5181-5197, December 2011.

[10] A. Uchida, Y. Ito, and K. Nakano, "An efficient GPU implementation of ant colony optimization for the traveling salesman problem". In 2012 Third International Conference on Networking and Computing (pp. 94-102). IEEE.

[11] J. M. Cecilia, J. M. García, J A. Nisbet, M. Amos, and M. Ujaldón, "Enhancing data parallelism for ant colony optimization on GPUs", Journal of Parallel and Distributed Computing, vol. 73, pp. 42-51, January 2013.

[12] A. Delévacq, P. Delisle, M. Gravel, and M. Krajecki, "Parallel ant colony optimization on graphics processing units", Journal of Parallel and Distributed Computing, vol. 73, pp. 52-61, January 2013.

[13] R. Skinderowicz, "The GPU-based parallel ant colony system", Journal of Parallel and Distributed Computing, vol. 98, pp. 48-60, December 2016.

[14] A. Llanes, J. M Cecilia, A. Sánchez, J. M García, M. Amos, and M. Ujaldón, "Dynamic load balancing on heterogeneous clusters for parallel ant colony optimization" Cluster Computing, vol. 19, pp. 1-11, January 2016.

[15] Y. Zhou, F. He, and Y. Qiu, "Dynamic strategy based parallel ant colony optimization on GPUs for TSPs", Science China Information Sciences, vol. 60, pp. 1-3, February 2017.

[16] J. M. Cecilia, A. Llanes, J, L. Abellan, J. Gomez-Luna, L. W. Chang, and W. M. W. Hwu, "High-throughput ant colony optimization on graphics processing units", Journal of Parallel and Distributed Computing, vol. 113, pp. 261-274, March 2018.

[17] R. Skinderowicz, "Implementing a GPU-based parallel MAX–MIN Ant System", Future Generation Computer Systems, vol. 106, pp. 277-295, May 2020.

[18] Z. B. Huang, G. T. Fu, T. H. Fa, D. Y. Dong, P. Bai, and C. Xiao, "High performance ant colony system based on GPU warp specialization with a static–dynamic balanced candidate set strategy", Future Generation Computer Systems, vol. 125, pp. 136-150, December 2021.

[19] M. Dorigo, T. Stützle, "The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances", F. Glover, G. A. Kochenberger, (eds) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 57, pp. 250-285, Springer, Boston, MA, 2003.

[20] S. Lin, and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem", Operations research, vol. 21 , pp. 498-516, 1973.

[21] N. Rokbani, P. Kromer, I. Twir, A. M. Alimi, "A new hybrid gravitational particle swarm optimisation-ACO with local search mechanism, PSOGSA-ACO-Ls for TSP", International Journal of Intelligent Engineering Informatics, vol. 7, pp. 384–398, August 2019.

[22] Ş. Gülcü, M. Mahi, Ö.K. Baykan and H. Kodaz, "A parallel cooperative hybrid method based on ant colony optimization and 3-Opt algorithm for solving traveling salesman problem". Soft Comput 22, 1669–1685 (2018).

[23] R. Skinderowicz, "Improving Ant Colony Optimization efficiency for solving large TSP instances", Applied Soft Computing, vol. 120, pp. 108653, May 2022.

[24] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," IEEE Transactions on Evolutionary Computation, vol. 1, pp. 53-66, April 1997.