IADIS International Journal on Computer Science and Information Systems
Vol. 15, No. 2, pp. 44-57
ISSN: 1646-3692

DESIGN OF COOPERATIVE OPENMP-BASED METAHEURISTIC APPROACH FOR MULTI-OBJECTIVE KNAPSACK PROBLEM

Imen Ben Mansour[1,2], Ines Alaya[1] and Moncef Tagina[1]
[1]*ENSI-COSMOS, University of Manouba, Manouba 2010,Tunisia.*
[2]*Esprit School Of Engineering, Tunis, Tunisia*

ABSTRACT

Parallelism arises as an attractive option when solving Multi-Objective optimization problems (MOPs). Moreover, it seems interesting when metaheuristics demand an intensive use of CPU or memory. In this paper, we propose a parallel implementation of a hybrid ant colony optimization metaheuristic for the multiobjective knapsack problem using the OpenMP framework called MHAC_OMP. The proposed approach combined a MultiObjective Ant Colony Optimization (MOACO) algorithm with Tchebycheff based Local Search (TLS) procedure. The idea behind MHAC_OMP is to evolve several independent MOACO in parallel. Each MOACO hold a local archive to maintain diversity. The parallelization is defined as assuming a shared-memory based on threads in which the initialization phase begins with a single thread called the master thread and executed sequentially. Afterward, a parallel region is defined where many threads are created, each one of them executing its own copy of the proposed ant colony algorithm independently. Experimental results show a significant efficiency of the solutions returned over the sequential implementation.

KEYWORDS

Parallel Metaheuristic, Threads, OpenMP, Ant Colony Optimization, Multiobjective Optimization, The Augmented Weighted Tchebycheff Method

1. INTRODUCTION

Several real-life optimization problems are modeled as a multiobjective combinatorial optimization problem, due to the multiple conflicting objectives and the different constraints that have to be respected simultaneously. Since such optimization problems cannot be tackled by exacts methods, metaheuristics represent a great alternative, they are used with less computational effort to find a good approximation of the optimal Pareto set.

Since introduced in 1992 by Marco Dorigo (Dorigo, 1992), ant colony optimization (ACO) algorithms have been successfully applied to many combinatorial mono and multi-objective optimization problems, ranging from traveling salesmen (TSP) (Angus, 2007; Dorigo & Gambardella, 1991) to knapsack problems (KP) (Alaya, Solnon, & Ghédira, 2007; Zouari, Alaya, & Tagina, 2017) and a lot of derived methods have been adapted to cooperative and parallel implementations.

ACO was proposed as a solution when suffering from limited diversification. Therefore, ACO is a powerful technique in exploration of the solution space. In addition to its significant performance improvement compared with other metaheuristic techniques, its multicore computing power encouraged the modification of the standard sequential form to be applied in a parallel framework.

In this paper, we propose an efficient and straightforward OpenMP multiobjective ACO (MOACO) implementation based on shared memory multiprocessing programming architecture. Here, OpenMP is implemented with its parallel regions. A shared memory model has been considered to get the benefit of creating a common space sharing pheromone matrix and the approximate Pareto set without the overhead of communication, especially when applying both the Construct Ant Solutions and Update Pheromone processes. The proposed approach is implemented to improve the approximation of the Pareto-optimal solutions not altering the MOACO algorithm behavior. The design of a parallel framework is motivated by, firstly, the need to improve the convergence and the diversity of the solutions, secondly to reduce computing efforts without altering its behavior. This paper proposes a solution with OpenMP to get the performance gain of parallel regions. These parallel regions provide parallelizing to the MOACO algorithm not only to speed-up the MOACO but also to improve its Pareto set's quality.

This paper is organized as the following: in Section 2, the related work to ACO and the research efforts towards its parallelization are presented. Section 3 introduces the studied problem: the multiobjective KP. Section 4 presents the sequential ACO algorithm mapped to the multiobjective knapsack problem. In section 4, the proposed ACO parallelization using OpenMP is introduced where its sub-sections show the analysis of different elements of OpenMP and its effects on performance. In section 5, results and performance evaluation are investigated using the multiobjective KP problem as an implementation of the parallel ACO algorithm. Finally, section 6 concludes the research and suggests the future work.

2. RELATED WORK

Many efforts have been devoted to adapting the ACO algorithm on several cooperative and parallel paradigms (Pedemonte, Nesmachnow, & Cancela, 2011). The first implementation of a parallel Ant System is attributed to (Bolondi & Bondanza, 1993), who presented a fine-grain implementation that uses several groups of ants executed a standard AS algorithm and placed one ant in each processor to solve the Traveling Salesman Problem (TSP).

Talbi et al. in (Talbi, Roux, Fonlupt, & Robillard, 2001) presented a method called ANTabu combining ACO and Tabu Search (TS), applied to solve the Quadratic Assignment Problem (QAP). The TS method was used as a local search to improve the solutions in each thread. The parallel version was compared against a sequential ACO, a parallel TS, genetic algorithms (GA) and variable neighborhood search. In (Lv, Xia, & Qian, 2006) Lv et al. proposed a parallel ACO

using P groups of ants, distributed in P processors, which shared one pheromone matrix in multiprocessing computer to solve TSPLIB instances with up to 15,915 cities. The algorithm achieved better solutions than sequential versions. Stützle in (Stützle & Hoos, 2000) studied the parallel independent execution of MAX − MIN Ant System (MMAS) with a local search for the TSP. The evaluation analyzed the benefits of using a parallel model with respect to a sequential ACO, comparing the quality of the solutions for TSPLIB instances with up to 1173 cities. The results showed that the parallel executions obtained better solutions than the sequential algorithm in all the studied instances. In (Bai, OuYang, Li, He, & Yu, 2009), the authors implemented parallel independent runs of MMAS on GPU. Each thread executed one ant and each thread block was used for independent execution. The main algorithm runs on GPU, while the CPU is only used to initialize the solutions and to control the iteration process. Regarding the quality of the solutions, the GPU-parallel implementation outperformed three sequential MMAS versions, while acceleration values between 2 and 32 were obtained. In (Abouelfarag, Aly, & Elbialy, 2015), the authors proposed a parallel ACO using the OpenMP framework. Parallel regions are used, to increase the overhead of creating and terminating threads and a shared memory model has been chosen to get the benefit of creating a common space sharing pheromone matrix without the overhead of communication.

Although several studies have been proposed for the parallel implementation of mono-objective ACO, as far as we know, few references can be found about parallel implementations of the multiobjective ACO (MOACO) (Falcón-Cardona, Leguizamón, Coello, & Tapia, 2020), even less with the openMP framework. In (Delisle, Krajecki, Gravel, & Gagné, 2001) a parallel implementation of an ant colony optimization metaheuristic with OpenMP for the solution of an industrial multiobjective scheduling problem in an aluminum casting center is introduced. The parallelization process consists to affect the generation and evaluation of each ant to a different processor. The pheromone matrix is in the shared memory and is updated by one ant, by an OpenMP thread, at a time, in a critical section of the parallel region. In (Mora, García-Sánchez, Merelo, & Castillo, 2013), the authors introduce an island-based model where the colonies communicate by migrating ants, following a neighborhood topology that fits the search space. Thus a parallel scheme will be implemented, taking a coarse-grained parallelization approach, at the colony level, so every computational node (processor) will contain a set of ants (colony).

3. MULTIOBJECTIVE KNAPSACK PROBLEM

The multiobjective multidimensional knapsack problem consists of finding a subset of items subject to a set of resource constraints while maximizing several objectives. Due to its NP-hard nature (Martello, 1990), several approaches has been proposed (Lust, & Teghem, 2012). Formally, MOMKP could be written as follows:

$$\text{Maximize} \quad \sum_{j=1}^{n} p_j^k x_j \qquad k = 1,\ldots, m \tag{1}$$

$$\text{Subject to} \quad \sum_{j=1}^{n} w_j^i x_j \leq b_i \qquad i = 1,\ldots, q \tag{2}$$

$$x_j \in \{0,1\} \qquad j = 1,\ldots, n$$

There are n items, for each item I_j is assigning a decision variable x_j equal to 1 if the item is selected, 0 otherwise. Each item I_j has a profit p_j^k relatively to the objective k and w_j^i consumed quantity relative to the resource i (like weight, volume...). The aim objective of the problem is to select a subset of items among the n items in order to maximize m objective functions while not exceeding q resource constraints b_i regarding the total quantity available for the resource i.

4. THE SEQUENTIAL MHAC

ACO metaheuristic is a cooperative population-based construction algorithm inspired from the behavior of real ants while searching for a food source. The colony of ants cooperates to build a set of solutions using an indirect form of communication, called pheromone, deposited by the member of the colony while building their solutions. The Tchebycheff-based Local Search (TLS) procedure is a simple heuristic algorithm used to find good-quality solutions or to improve results in a relatively short time. They provide an interesting alternative to classical evolutionary algorithms and often involve a small number of parameters.

```
MHAC()
{
 Initialize Pheromone Structure to τ_max
 A← Ø
While(run time limit not reached)
{
  Sol← Ø
  For each ant k
 { Construct Solution S^h;
    Sol ← Sol ∪ S^h; }
 Local Search (Sol);
A ← non-dominated (A ∪ Sol);
Update Pheromone Structure τ(I_j);
P ← non-dominated (A ∪ Sol);
}
}
```

Figure 1. The pseudo code of MHAC algorithm

Here, the hybrid approach called MHAC is proposed as a synergy of the multiobjective ACO algorithm with a Tchebycheff-based Local Search (TLS) procedure. In summary, the structure of the MHAC algorithm is given in Figure 1. The MOACO algorithm is used to favor the exploration in the search space while the TLS method tries to enhance the solutions built by ants. Since our algorithm follows the MMAS scheme (Stützle & Hoos, 2000) so, as a first step, the pheromone traces are initialized to an upper bound τ_max. Apart from that, mainly, the hybrid MOACO algorithm for MOMKP consists of three procedures described in the following sub-sections.

4.1 Construct Ants Solutions

To tailor the ACO for solving the Knapsack problem (KP), the problem must be reduced to a complete graph G =(V ,E) along which the ants move to build candidate solutions. Where V is the set of vertices and E is the set of edges. The vertices V acted as the set of items to be selected and E represents the connection between the vertices V i.e. path constructed by a given ant. Here, the pheromone trails are associated with the vertices of this graph. Each ant h constructs one feasible solution by applying repeatedly the state transition rule to select the most appropriate item I_j to be added to the solution S^h among a set of feasible items Feas (candidate vertices). This set is updated by including items not yet added and that doesn't violate any constraint. The transition rule p_{S^h} is directed by the pheromone value τ_{S^h} and the heuristic information η_{S^h}:

$$p_{S^h}(I_j) = \frac{\left[\tau_{S^h}(I_j)\right]^{\alpha}\left[\eta_{S^h}(I_j)\right]^{\beta}}{\sum_{I_l \in Feas}\left[\tau_{S^h}(I_l)\right]^{\alpha}\left[\tau_{S^h}(I_l)\right]^{\alpha}} \quad (3)$$

The heuristic information is used to guide the search process of artificial ants. In order to orientate ants to look in different regions of the non-dominated front, different configurations of the heuristic information matrix are executed, i.e. using different weight vectors $\lambda(g)=(\lambda_1(g),\ldots,\lambda_m(g))$. To generate the set of the weight vector, we use the Gradual weights generation method (Gw) (Mansour, Alaya, & Moncef, 2017a, 2017b, 2019). For that, η_{S^h} for a given ant h, is set as:

$$\eta_S^h(I_j) = \frac{\sum_{k=1}^{m}\lambda_k(g)p_j^k}{\sum_{i=1}^{q}\left(\frac{w_j^i}{R_S^h(i)}\right)} \quad (4)$$

where is the remaining amount of the resource i when an ant h is currently building its solution S^h. p^k_j and w^i_j are respectively the profit and the weight of the candidate item.

4.2 Local Search

This step is started after the solution construction phase and before the pheromone update. The local search procedure aims to improve the quality of solutions generated by the Ant Colony algorithm. The figure 2 presents the local search procedure used in the proposed MHAC algorithm. The fitness value of each solution in Sol is calculated before performing a local search step. This step consists of exploring the neighborhood of each solution S^h in Sol until we find a solution S^{h*} that is better than the worst solution w of Sol regarding the search direction λ under consideration. Then, S^{h*} is added to Sol and replaces the solution w. The neighborhood exploration process stops once the first improving neighbor is found.

In order to evaluate a solution S^h against the whole population, we use one of the most commonly-used scalarization approaches: the augmented weighted Tchebycheff method (Steuer & Choo, 1983):

$$A_{WT}(S^h|\lambda, z^*) = max_{k=1\ldots m}\{\lambda_k.|z_k^* - f_k(S^h)|\} + \varepsilon \sum_{k=1}^{m}|z_k^* - f_k(S^h)| \quad (5)$$

Where z^* is the ideal point, used as a reference point here, and updated during the execution of the algorithm; $z_k^* = max_{k=1\ldots m} f_k(S^h)$ and $\varepsilon \geq 0$ is usually chosen as a small positive number.

```
Local Search(Sol)
{
 For each solution S^h
 {
   Evaluate solution S^h;
   Repeat {
   Generate neighbor S^h* of S^h;
   Evaluate S^h*
   If S^h* is better than the worst solution in Sol w
   Replace w with S^h* in Sol
   }
   Until (S^h*≠ w or all neighbors are explored)
 }
}
```

Figure 2. The the local search procedure

Since the 0/1 multi-objective knapsack problem is a constrained problem, all solutions should satisfy all resource constraints. Indeed, to generate a new neighbor of S^h, we have to use an efficient neighborhood structure for this problem. To this end, we use the same neighborhood function proposed in (Mansour, Alaya, & Moncef, 2017a). This function allows establishing an order between items according to their profitability. It has the goal of finding the item that minimizes an extraction ratio to permute it with items that maximize an insertion ratio in order to improve the quality of the obtained solutions, i.e. the neighbor in a small computational time.

A solution S^h to the MOMKP can be presented as a double list $I = (I_l^+, I_{\bar{l}}^-)$. The first list I_l^+ corresponds to the taken item (belonging to the solution) $I_l^+ = \{I_1^+, I_2^+, \ldots, I_T^+\}$, where T is the size of the list. The second list $I_{\bar{l}}^-$ is the list of the remaining items $I_{\bar{l}}^- = \{I_1^-, I_2^-, \ldots, I_{NT}^-\}$, where NT is the number of unselected items.

The transition from one solution S^h to $S^{h'}$ is referred as a move. Two neighborhood operators are performed in sequential order in this paper for the MOMKP:

$U(l^+)$: The extraction ratio is calculated for all items in the list I_l^+, which measures the utility value of each item. The lower this ratio is, the worst the item is. The figure 3 outlines the Extraction_Items algorithm.

```
Extraction_Items ()
{
 For l+ from 1 to T
 {
   Calculate U(l+) = (Σ_{k=1}^{m} λ_k(g) p_{l+}^k) / (Σ_{i=1}^{q} w_{l+}^i);
   Add item I_l^+ to list L_U
 }
 Sort L_U in ascending order
}
```

Figure 3. The Extraction_Items algorithm

$\overline{U}(\bar{l})$: The insertion ratio is calculated for all unselected items in list $I_{\bar{l}}^{-}$, the ratio measures the quality of the candidate item according to the solution S^h where the higher this ratio is, the better the item is. The figure 4 outlines the Insertion_Items algorithm.

```
Insertion_Items ()
{
 For  l̄  from 1 to NT
 {
   Calculate Ū(l̄) = (Σ_{k=1}^{m} λ_k(g) p_{l̄}^k) / (Σ_{i=1}^{q} w_{l̄}^i / R_i(s));
   Add item I_l̄^- to list L_Ū
 }
 Sort L_Ū in decreasing order
 S^h' ← Remove L_U(l+) from S^h
 While no constraint is violated and l̄ < NT do
 {
  S^h'← Add L_Ū(l̄) to solution S^h'
 }
}
```

Figure 4. The Insertion_Items algorithm

4.3 Pheromone Update

The pheromone update step is the most important phase. When all ants construct their solutions, the non-dominated solutions of the current generation are stored in the archive. The ants that are allowed to update the pheromone trails, are thus who have found the member of the current archive by laying an amount of pheromone on each item. The pheromone trail is updated as follows:

$$\tau(I_j) \leftarrow (1-\rho) * \tau(I_j) + \Delta\tau(I_j) \qquad (6)$$

with

$$\Delta\tau(I_j) = \begin{cases} |A| & if\ I_j \in S_{ND} \\ 0 & otherwise \end{cases} \qquad (7)$$

In order to give every generation of ants the same influence in a particular part of the Pareto front, ants that are authorized to update, lay the same amount of pheromone which is equal to the current archive set size.

5. THE PROPOSED PARALLEL MHAC_OMP

The Ant Colony Optimization is a potential candidate for parallelization for different reasons. One can cite: The large number of iterations required in updating pheromone trails, the computations needed for the single ant/ group of ants to construct solution(s) and the independent behavior of ants. Many strategies have been proposed to implement a parallel ACO. In this paper, we adopt a parallelization strategy based on multi-thread programming with multi-core processors. This section introduces an implementation for parallel MHAC using the OpenMP framework.

The idea is to execute several algorithms instead of a panmictic one, since the management of the many populations has a great impact on the efficiency of the approach and its ability to compute a sufficiently diverse and representative Pareto front. Also, we choose to parallel the places which consume most execution time in the sequential MHAC and to overcome the problem of communication overhead by using the OpenMP directives. Larger OpenMP parallel regions are used, because fragmented parallel regions would increase the overhead of creating and terminating threads.

The proposed framework begins as a single execution thread called the master thread. When the thread meets the parallel sections, it creates a team of threads consisting of the initial thread itself and the other threads and becomes the main thread in the team. All the members of the team execute a copy of the MHAC algorithm and update their local archive called A and then collaborate to the update of the global Archive named P. At the end of the parallel sections, the further execution of the code is performed only by the master thread. Therefore, the main thread is responsible for the final update the global archive holding all the non-dominated solutions found so far.

Figure 5. shows the parallel programming method of the proposed Algorithm, Multiobjective Hybrid Ant Colony based on OpenMP (MHAC_OMP). The algorithm is divided into four sections relatively to the “sections” of OpenMP. Each section is assigned to one thread. Once all threads finish the execution of MHAC_OMP, the shared global archive P is updated by all the threads and returned by the master thread. In summary, the structure of the MHAC_OMP algorithm is given in Figure 6.

```
Initialize Global Archive P
Initialize pheromone trails;
#pragma omp parallel sections shared (P, τ(I_j))
 {
   #pragma omp section
   MHAC_OMP();
   #pragma omp section
   MHAC_OMP();
   #pragma omp section
   MHAC_OMP();
   #pragma omp section
    MHAC_OMP();
 }
Update and return (P);
```

Figure 5. Parallel Programming method of MHAC_OMP

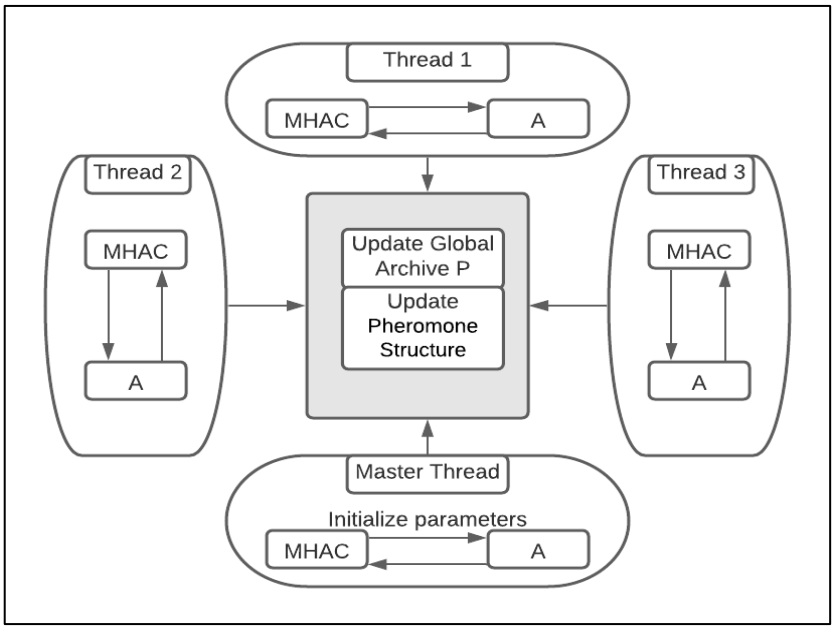

Figure 6. The general schema of MHAC_OMP with 4 threads. The gray zone corresponds to the common space between threads sharing pheromone matrix and Pareto set. The arrows correspond the sending of non-dominated solutions

6. EXPERIMENTS AND RESULTS

In this section, we assess the performance of the MHAC_OMP. The platform for conducting the experiments is a IntelCore i5 2.60 GHz laptop with 6 GB RAM. The program was implemented using C++ language with OpenMP 4.0. We examine the performance of MHAC_OMP on nine instances of MOMKP (Zitzler & Thiele, 1999) with two sequential MOACO algorithms: Gw-ACO (Mansour, Alaya, & Moncef, 2019) and the sequential version of MHAC_OMP called here MHAC.

To evaluate the efficiency of MHAC_OMP, we use as metrics of performance the hypervolume difference (HypD) (Zitzler & Thiele, 1999). The HypD value has to be as close as possible to zero to prove the efficiency of the algorithm. To compare the approaches behaviors in a graphical way, we use also the summary attainment surface (Da Fonseca, Fonseca, & Hall, 2001). We perform the non-parametric Mann-Whitney statistical test described in (Knowles, Thiele, & Zitzler, 2005.) in order to verify if the difference between the tested algorithms is statistically significant with a confidence level greater than 95% (p-value ≤ 0.05). The bold values in the tables denote the best results found in the considered instance by the corresponding algorithm.

6.1 Parameterization of MHAC_OMP

Metaheuristic algorithms and parallel approaches require a crucial decision about the values of numerous parameters. The solution quality and speed may be affected by the parameter settings. Accordingly, we have been striven to determine an appropriate set of parameter values for the MD-HACO. The number of threads is fixed to 4, the significance weights for pheromone trail α set to 1, the significance weights for heuristic information β set to 10, pheromone evaporation rate ρ set to 0.90, the number of ants equal to 10, the lower bound of pheromone τ_{min} is 1 and the upper bound τ_{max} is 5.

Each algorithm runs 30 times and each time we run 100 seconds if the number of objectives if 2, 300 seconds if is equal to 3 and 500 seconds with 4 objectives. We use for each algorithm the same computation time budget. This choice was motivated by the fact that measuring the speed up using the time taken by parallel and sequential algorithms to find good solutions is a more significant measure of performance than simply comparing the time taken to run a certain number of generations.

6.2 Experimental Results

Table 1 reports the average values obtained by the hypervolume difference metric when comparing MHAC_OMP, Gw-ACO and MHAC. By analyzing the table it is clear that MHAC_OMP achieves the best results. In fact, MHAC_OMP is more effective than Gw-ACO. Moreover, this parallel approach returns better values than the sequential one on all tested instances.

Table 2 gives the p-values of the Mann-Whitney statistical test of MHAC_OMP against Gw-ACO and MHAC. Indeed, this table confirms the results obtained in Table 1. The returned values highlight that MHAC_OMP strongly outperforms Gw-ACO on all tested instances. When comparing the p-values obtained by MHAC_OMP against MHAC, one can say that the parallel approach statistically outperforms the sequential one in almost all instances. In fact,

MHAC_OMP is significantly better than MHAC for 7 out of the 9 instances. Moreover, it seems that the results of the parallel algorithm increase according to the size of the problem. Clearly, the larger the instance is, the better the results are.

Table 1. Average values of the hypervolume difference metric of MHAC_OMP, Gw-ACO and MHAC

Instance	MHAC_OMP	Gw-ACO	MHAC
250_2	**1.50E-01**	4.29E-01	2.32E-01
500_2	**1.39E-01**	4.28E-01	1.92E-01
750_2	**1.28E-01**	4.20E-01	1.45E-01
250_3	**1.69E-01**	5.41E-01	1.70E-01
500_3	**1.80E-01**	5.49E-01	2.02E-01
750_3	**1.45E-01**	5.72E-01	1.81E-01
250_4	**1.73E-01**	5.59E-01	2.12E-01
500_4	**1.49E-01**	5.23E-01	1.77E-01
750_4	**1.49E-01**	5.60E-01	1.76E-01

Table 2. The p-value of the Mann-Whitney statistical test of MHAC_OMP compared to Gw-ACO and MHAC

Instance	Gw-ACO	MHAC
250_2	≤ **0.05**	≤ **0.05**
500_2	≤ **0.05**	≤ **0.05**
750_2	≤ **0.05**	0.38
250_3	≤ **0.05**	0.5
500_3	≤ **0.05**	≤ **0.05**
750_3	≤ **0.05**	≤ **0.05**
250_4	≤ **0.05**	≤ **0.05**
500_4	≤ **0.05**	≤ **0.05**
750_4	≤ **0.05**	≤ **0.05**

Figure 7 plots the median attainment surface of the approximation sets of MHAC_OMP, Gw-ACO and MHAC on bi-objective instances. The figure shows that MHAC_OMP produces a very well-distributed Pareto front. It is evident from the figure 7, that almost all the final solutions obtained by MHAC_OMP dominate those obtained by Gw-ACO and there are no solutions returned by Gw-ACO that dominate anyone returned by MHAC_OMP and this is for all the bi-objective tested instances.

From the same figure, one can see that the surfaces of MHAC_OMP and those of MHAC are difficult to distinguish visually, but it is clear that MHAC_OMP returns an important number of solutions than MHAC and that the parallel approach provides a better distribution i.e. the surfaces of MHAC_OMP cover all the extreme ends of the Pareto front.

7. CONCLUSION

In this paper, we have proposed a parallel Multiobjective Ant Colony Optimization approach based on OpenMP to solve the Multiobjective Knapsack Problem. The nature of the ACO and the functionality offered by OpenMP made the transition from sequential to parallel easier and straightforward, while some modifications had to be made to the algorithm to obtain the level of efficiency that we achieved in comparison with other approaches. Ongoing work follows two main directions: the study of the further properties of OMP with MOACO and the development of further applications to multiobjective combinatorial optimization problems.

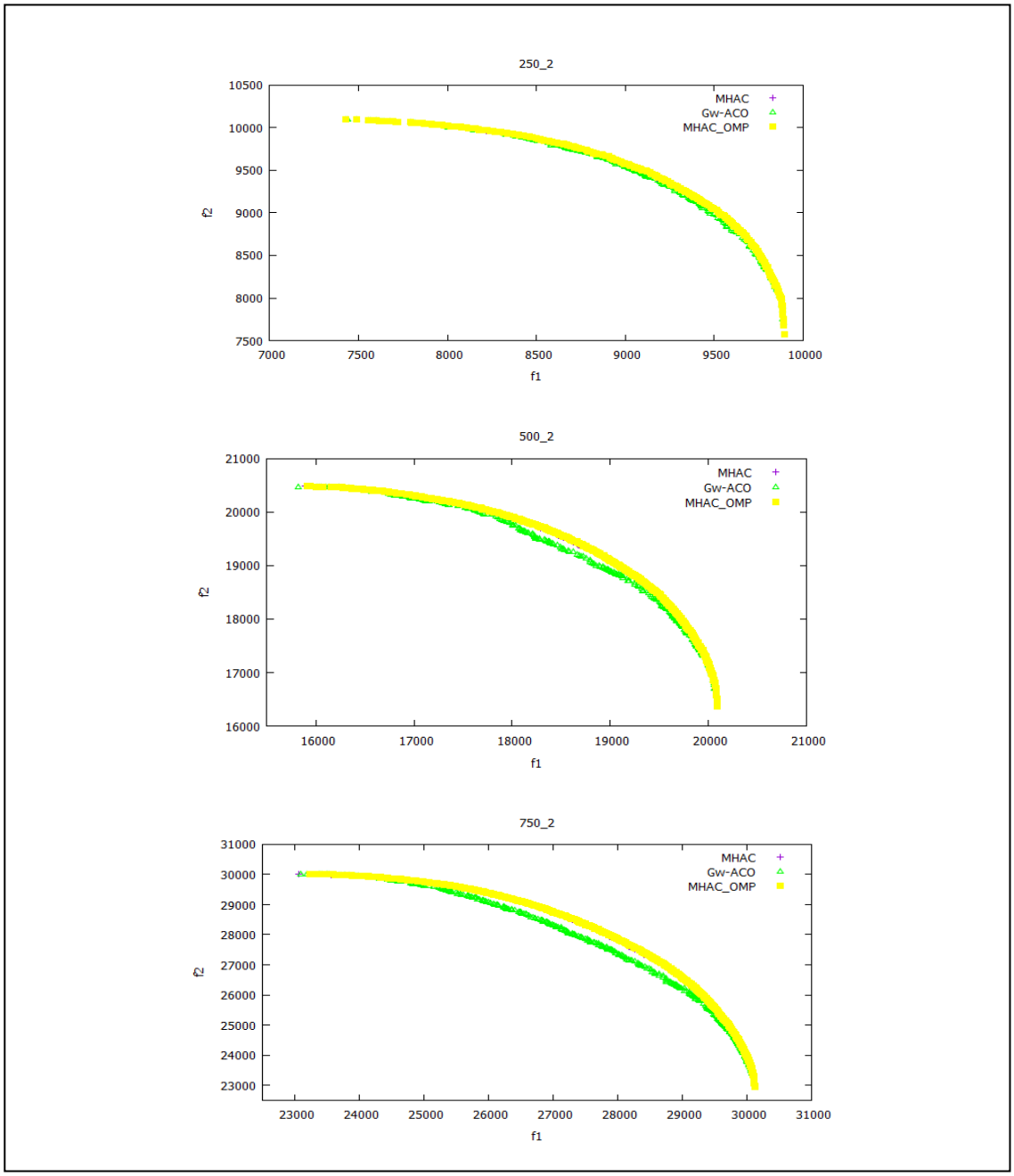

Figure 7. The median attainment surface obtained by MHAC_OMP, Gw-ACO and MHAC on biobjective instances

REFERENCES

Abouelfarag, A. A., Aly, W. M., & Elbialy, A. G., 2015. Performance analysis and tuning for parallelization of ant colony optimization by using OpenMP. *In IFIP International Conference on Computer Information Systems and Industrial Management*. Springer, Cham. 73-85.

Alaya, I., Solnon, C., & Ghédira, K., 2007. Ant Colony Optimization for Multi-objective Optimization Problems. *19th IEEE International Conference on Tools with Artificial Intelligence* (ICTAI'07), 450-457.

Angus, D, 2007. Crowding population-based ant colony optimisation for the multi-objective travelling salesman problem. *IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM)*,333–340.

Bai, H., OuYang, D., Li, X., He, L., & Yu, H., 2009. Max-min ant system on gpu with cuda. Proceedings *of the 2009 Fourth International Conference on Innovative Computing, Information and Control, IEEE Computer Society*. 801–804.

Bolondi, M., & Bondanza, M., 1993. Parallelizzazione di un algoritmo per la risoluzione del problema del commesso viaggiatore (Doctoral dissertation, Master's thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy).

Delisle, P., Krajecki, M., Gravel, M., & Gagné, C., 2001. Parallel implementation of an ant colony optimization metaheuristic with OpenMP. *Proceedings of the 3rd European Workshop on OpenMP (EWOMP'01)*, Barcelona, Spain.

Dorigo M., 1992. *Optimization, Learning and Natural Algorithms*. PhD thesis. Politecnico di Milano, Italy.

Dorigo, M., & Gambardella, L. M., 1997. Ant colony system: A cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions*, 1(1), 53-66.

Falcón-Cardona, J. G., Leguizamón, G., Coello, C. A. C., & Tapia, M. G. C. .,2020 Multi-Objective Ant Colony Optimization: An Updated Taxonomy and Review of Approaches.

Da Fonseca, VG., Fonseca, CM., & Hall, AO., 2001. Inferential performance assessment of stochastic optimisers and the attainment function. *1st International Conference on Evolutionary Multi-criterion Optimization (EMO 2001)*. Lecture Note in Computer Science. 1993,213–225. Springer, Berlin.

Knowles, D [Joshua], Thiele, L., & Zitzler, E., 2005. A tutorial on the performance assessment of stochastive multiobjective optimizers. *Technical report TIK-Report* No. 214, Computer Engineering and Networks Laboratory, ETH Zurich.

Lust, T., & Teghem, J., 2012. The multiobjective multidimensional knapsack problem: a survey and a new approach. *International Transactions in Operational Research* 19 (4), 495-520.

Lv, Q., Xia X., & Qian, P., 2006. A parallel ACO approach based on one pheromone matrix. *Proceedings of the 5th International Workshop on Ant Colony Optimization and Swarm Intelligence*. Lecture Notes in Computer Science. 4150. 332–339.

Mansour, I. B., Alaya, I., & Moncef, T., 2017a. A Min-Max Tchebycheff Based Local Search Approach for MOMKP. In ICSOFT (pp. 140-150).

Mansour, I. B., Alaya, I., & Moncef, T., 2017b. Chebyshev-based iterated local search for multi-objective optimization. *In 2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*. 163-170.

Mansour, I. B., Alaya, I., & Moncef, T., 2019. A gradual weight-based ant colony approach for solving the multiobjective multidimensional knapsack problem. *Evolutionary Intelligence*, 12(2), 253-272.

Martello S., 1990. Knapsack problems: algorithms and computer implementations. *Wiley-Interscience series in discrete mathematics and optimization*.

Mora, AM., García-Sánchez, P., Merelo, JJ., & Castillo, PA., 2013. Pareto-based multi-colony multi-objective ant colony optimization algorithms: an island model proposal. *Soft Computing* 17.7. 1175-1207.

Pedemonte, M., Nesmachnow, S., & Cancela, H., 2011. A survey on parallel ant colony optimization. *Applied Soft Computing*. 11, 5181-5197.

Steuer, R. E., & Choo, E. U., 1983. An interactive weighted Tchebycheff procedure for multiple objective programming. *Mathematical programming*, 26(3), 326-344.

Stützle T., 1998. Parallelization strategies for ant colony optimization. *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, Lecture Notes in Computer Science. 1498 722–731.

Stützle, T., & Hoos, H. H. 2000. MAX–MIN ant system. *Future generation computer systems*, 16(8), 889-914.

Talbi, EG., Roux, O., Fonlupt, C., & Robillard, D., 2001. Parallel ant colonies for the quadratic assignement problem, *Future Generation Computer Systems* 17 (4) 441–449.

Zitzler, E., & Thiele, L., 1999. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*. 3(4): 257–271.

Zouari, W., Alaya, I., & Moncef, T., 2017. A hybrid ant colony algorithm with a local search for the strongly correlated knapsack problem. *In 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*. 527-533.