# Comparison of Different Topologies for Island-Based Multi-Colony Ant Algorithms for the Minimum Weight Vertex Cover Problem

RAKA JOVANOVIC
Institute of Physics
Belgrade
Pregrevica 118, Zemun
SERBIA
rakabog@yahoo.com

MILAN TUBA
Faculty of Computer Science
Megatrend University Belgrade
Bulevar umetnosti 29
SERBIA
tubamilan@ptt.rs

DANA SIMIAN
Department of Computer Science
Lucian Blaga University of Sibiu
5-7 dr. I. Ratiu str.
ROMANIA
d_simian@yahoo.com

*Abstract:* - The aim of this paper is compare the effect of using different topologies or connections between separate colonies in island based parallel implementations of the Ant Colony Optimization applied to the Minimum Weight Vertex Cover Problem. We investigated the sequential Ant Colony Optimization algorithms applied to the Minimum Weight Vertex Cover Problem before. Parallelization of population based algorithms using the island model is of great importance because it often gives super linear increase in performance. We observe the behavior of different parallel algorithms corresponding to several topologies and communication rules like fully connected, replace worst, ring and independent parallel runs. We also propose a variation of the algorithm corresponding to the ring topology that maintains the diversity of the search, but still moves to areas with better solutions and gives slightly better results even on a single processor with threads.

*Key-Words:* - Ant colony optimization, Minimum weight vertex cover problem, Parallel computing, Combinatorial optimization, Evolutionary computing

## 1 Introduction

Most processors today have multiple cores and even for a single core multiple treads can be implemented. In general, a system of *n* parallel processors, each of speed *k*, is less efficient than one processor of speed *n*k*. However, such parallel system is usually much cheaper to build and because of that research in parallelization is of great importance. Parallelization of algorithms has proven to be very powerful method in the case of population based algorithms like Ant Colony Optimization (ACO) and Genetic algorithms [1].

The basic idea of ACO is to imitate the behavior of ants in a colony while gathering food. Each ant starts from the nest and walks toward food. It moves until an intersection where it decides which path it will take. In the beginning it looks like a random choice but after some time the majority of ants are using the optimal path. This is because the colony works as a group and not just as individual ants and this is achieved by using pheromone. Each ant deposits pheromone while walking which marks the route taken. The amount of pheromone indicates the usage of a certain route. Pheromone trail evaporates as time passes. Due to this a shorter path will have more of pheromone because it will have less time to evaporate before it is deposited again. The colony behaves intelligently because each ant chooses path that has more pheromone. There are many different ways of converting the presented behavior into a computational system, the most widespread is the one presented by Marco Dorigo and Luca Maria Gambardella [2].

Different parallelization approaches have been applied to ACO algorithms. It has been shown that the multi-colony model is more effective than the parallelization applied by assigning separate processes to ants belonging to a single colony. This is similar to the situation with genetic algorithms where the best application of parallelization is to create separate islands of populations and to implement some kind of communication between them. This approach gives even super-linear improvement to population based algorithms applied to certain problems [3]. Due to this fact parallelization of ACO has been successfully applied to a wide set of different problems like TSP [4], Quadratic Assignment Problems [5], Routing in MANETs (Mobile Ad Hoc Networks) [6], Task Scheduling [7], DNA Sequencing [8].

When working with multi-colony systems, the communication data is of great importance. Solutions, pheromone matrices, and parameters have all been tested as the type of information that will be exchanged between colonies [9], [10], [11]. The exchange of the best-so-far solution has been shown to be a good choice, which we use in our comparisons of different topologies.

The last step in application of parallel ACO is to define the methods of communication and interaction between colonies, and the corresponding algorithms. These algorithms are named by their corresponding topologies and the standard ones are:

- fully connected
- replace worst
- ring
- independent parallel runs

We compare the quality of the results acquired by these parallel algorithms with the results of the sequential implementation and our variation of the ring topology algorithm.

To illustrate these parallel implementations we use one of the classical problems of graph theory: the Minimum Vertex Cover Problem. The problem is defined for an undirected graph $G = (V, E)$. $V$ is the set of vertexes and $E$ is a set of edges. A vertex cover of a graph is set of vertexes $V' \subset V$ that has the property that for every edge $e(v_1,v_2) \in E$ at least one of $v_1,v_2$ is an element of $V'$. A minimal vertex cover is a vertex cover that has the minimum number of vertexes. In this paper we devote our attention to an extension of this problem named the Minimum Weight Vertex Cover Problem (MWVCP) in which weights are added to the vertexes. The solution is not the vertex cover with the minimum number of vertexes, but one with the minimum sum of weights.

It has been shown that this problem is NP-complete even when it is restricted to a unit-weighted planar graph with the maximum vertex degree of three [12]. In the same way as for many other NP-complete problems, finding the optimal solution is very time consuming and in larger problem cases even impossible in realistic time. Variety of different methods have been investigated for calculating near optimal solutions. The first is a greedy heuristic approach of collecting the vertex with the smallest ratio between its weight and degree [13], [14]. Genetic algorithms have also been used [15].

The use of ant colony optimization gives very good results when used for the MWVCP, better that results acquired by genetic algorithms and local search methods like tabu search, and simulated annealing [16].

This paper is organized as follows. In Section 2 we present the implementation of ACO for the MWVCP. In Section 3 we discuss parallelization of the Ant Algorithms. In Section 4 different parallel topologies for ACO are presented. In the Section 5 we present our implementation of parallelization and in Section 6 conducted experiments and comparison of the effectiveness of these algorithms to the sequential one.

## 2 ACO for the MWVCP

The use of ACO has been proven to be effective on various types of problems from Economic Load Dispatch [17], Scheduling problems [18], Image processing [19], and also the MWVCP [16].

The MWVCP is in two main aspects different from most of the problems solved by using ACO. The solution of the problem is a subset of the graph vertexes set, instead of a permutation. The heuristic function is dynamic, while in most of other applications it is static. These two differences affect the basic algorithm in two directions. First, ants leave the pheromone on vertexes instead of on edges and second, we dynamically update the graph, and with it, the heuristic function. The first step in solving these problems is representing the problem in a way that makes dynamic calculation of the heuristic function simple.

Since ants in their search can move from a vertex to any other vertex, it is natural to use a fully connected graph $G_c(V,E_c)$ derived from $G$. In the article [16] it is proposed to add weights to edges in the new graph $G_c$. If an edge exists in $G$, it is given the weight 1, or 0 if it does not exist in the original graph. We have adopted this approach,

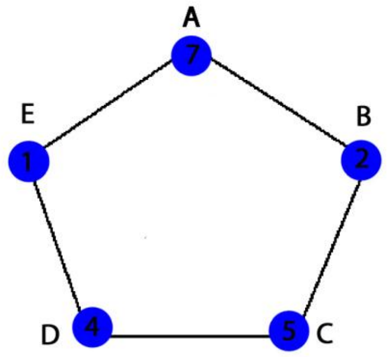

**Fig. 1** Original graph

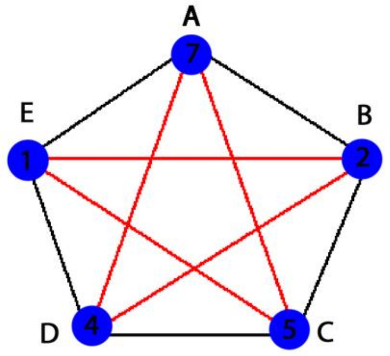

**Fig. 2** Fully connected graph

which is illustrated by Fig.1, the original graph and Fig 2, the derived graph. Lines colored in black represent edges with value 1, the red ones have the value 0.

As we mentioned before, we also have to update this graph as we add new vertexes to the result set. This is done using the following rule: when we add vertex *a* weights of all edges in $G_c$ that are connected to *a*, are set to 0. This is illustrated by Fig. 3.

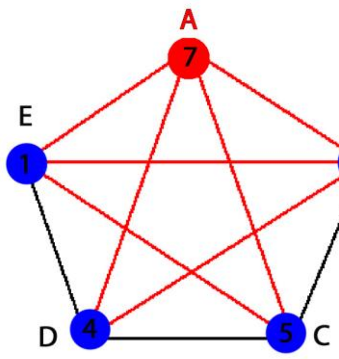

**Fig. 3** Adding a vertex to the solution set

Let us define $G_k$ $(V,E_{ck})$ as the state of the graph after *k* vertexes have been added to the solution set, and a corresponding function:

$$\psi_k(i,j) = Value(E_{ck}(i,j)) \qquad (1)$$

This update rule has two roles. First, we can dynamically evaluate the preference of vertexes with function $\psi_K$ and second, it gives us the information when all edges have been covered, or more precisely, if the total sum of edge weights in $G_k$ is 0, then all edges are covered. Now we can define a dynamic heuristic

$$\eta_{jk} = \frac{\sum_{(i,j)\in E_c} \psi_k(i,j)}{w(j)} \qquad (2)$$

In Equation 2 *w(j)* is the weight of a vertex. Using the heuristic defined with $\eta_{jk}$ in Equation 2 we can setup the state transition rule for ants:

$$p_j^k = \begin{cases} 1 & , q > q_0 \ \& \ j = \arg\max_{i\in A_k} \tau_i \eta_{ik}^{\alpha} \\ 0 & , q > q_0 \ \& \ j \neq \arg\max_{i\in A_k} \tau_i \eta_{ik}^{\alpha} \\ \dfrac{\tau_j \eta_{jk}^{\alpha}}{\sum_{i\in A_k} \tau_j \eta_{ik}^{\alpha}} & , q \leq q_0 \end{cases} \qquad (3)$$

In Equation 3 $q_0$ is the standard parameter that specifies the *exploitation/exploration* rate, and *q* is a random variable that decides the type of selection on each step. $A_k$ is a list of available vertexes. We point out that opposite to the TSP transition rule, it does not depend on the last selected vertex and that is why we have $\tau_i$ instead of $\tau_{ij}$.

To fully specify an Ant Colony System we still have to define the global (when an ant finishes its path) and a local (when an ant chooses a new vertex) update rules. The role of the global update rule is to make paths creating better solutions to become more desirable, or in other words, it intensifies exploitation.

$$\Delta\tau_i = \frac{1}{\sum_{j\in V'} w(j)} \quad , \forall i \in V' \qquad (4)$$

$$\tau_i = (1-p)\tau_i + \Delta\tau_i \qquad (5)$$

Equation 4 defines the global update rule. In it $\Delta\tau_i$ is a quality measure of solution subset *V'* that contains vertex *i*, and with it we define a global update rule in Equation 5. This measure is inverse proportional to the weight of a solution. Parameter *p* is used to set the influence of newly found solution on the pheromone trail.

The local update rule purpose is to shuffle solutions and to prevent all ants from using very strong vertexes. The idea is to make vertexes less desirable as more ants visit it. In this way, exploration is supported. The formula for the local update rule has the standard form

$$\tau_i = (1-\varphi)\tau_i + \varphi\tau_0 \qquad (6)$$

For the value of $t_0$ we take the quality measure of the solution acquired with the greedy algorithm when we select the vertex with the best ratio of vertex degree and weight. Parameter $\varphi$ is used to specify the strength of the local update rule. This implementation with other different variations of ACO are compared in [20, 21].

## 3 Parallel ACO

Due to the properties of ant based algorithms it is natural to use parallelization in its application. There are several different approaches to parallel implementations of ant algorithms that have been described in the literature. The most natural way extending ACO to parallel algorithms is connecting ants with processes. A very fine-grained parallelization, where every processor holds an individual ant was presented by Bolondi and Bondaza [22].

Characteristic of fine-grained approaches is that very few, often only one, individuals are assigned to one processors and these individuals are connected by a population structure. One of the main problems of this approach is the large overhead in communication that appears with the increase of the number of processors. Because of that this implementation does not scale very well with the growth of the number of processors. In the same thesis it is shown that when a coarser grid is used, better results are achieved.

Bullnheimer et al. discusses in their article [23] two approaches to this type of parallelization: one that is synchronous and a second one partially asynchronous. In the synchronous application they propose a straightforward strategy for the Ant System for computing the TSP tours in parallel. An initial process "master" starts a set of processes "workers" one for each ant. After distributing starting information about the problem, the distance matrix and the initial trail intensities to all the ants, every "worker" can start to draw up the path and compute the tour ant. When ants finish their path calculations the resulting path is sent from each worker back to the master. Afterwards the master updates the trail levels, and the global best solutions and new worker processes are started. In the asynchronous approach each worker holds a certain number of processes (ants) and performs independently of other workers a certain number of iterations of the sequential algorithm on them consecutively. After these local iterations are finished a global synchronization among all workers will be performed. The master will then globally update all the colony data. This approach reduced the communication overhead considerably and gave good and promising values. An interesting approach for multi-agent methods in a decentralized, asynchronous and parallel environment is given in article [24] which is more closely connected with its ant colony natural counterpart. In this approach the pheromone infrastructure, used for storing the pheromone matrix, for all the processes is divided into parts and control over each of them is given to separate processes.

The Bullnheimer asynchronous approach comes close to the, in majority of the cases, more efficient island model which was first used for genetic algorithms. The island models for genetic algorithms (GAs) have been intensively studied in [25]. Stutzle in his article [26] compared the solution quality of one long and several independent short runs of an ant algorithm. He has shown that when the running time of the sum of short ones is equal to the long run, under some conditions the short runs proved to give better results. In the island model application of parallel ACO each colony has a single processor dedicated to it. When applying this model of parallelization of great importance is the data that is going to be exchanged between processes. One of the possibilities is communicating the pheromone matrix between colonies, but this approach has the significant drawback of a large amount of data traveling between them. A better way, both in the sense of final result quality and data transfer quantity is exchanging the best-so-far solutions of colonies. The exchange between solutions is usually done either when colonies finish some fixed number of iterations or periodically every *t* time units.

## 4 Different Topologies for Parallel ACO Algorithms

Parallel algorithms are very important for population based optimization heuristics because they can give super-linear increase in efficiency. This level of improvement is accomplished with the use of multiple colonies. In our parallel implementation of ACO for the MWVCP we have adopted the island approach in which each colony has been given a separate process. The method of communication between colonies that we implemented is the exchange of the best-so-far solution found by each colony. We focus our analysis to the possible systems of communication and their effectiveness in the case of MWVCP. In the following we describe these communication methods which are slight variation of the topologies presented in article [4] and the principles of the communication on each of them:

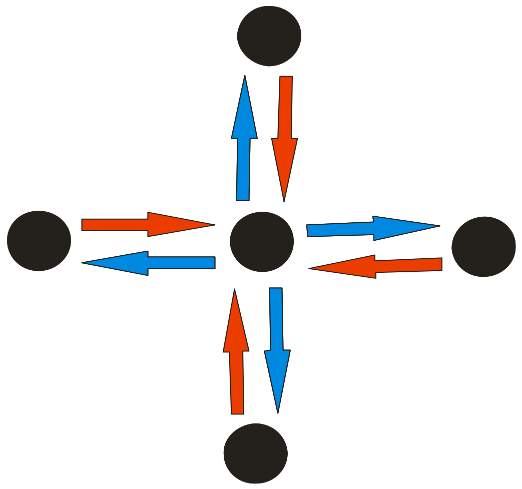

**Fig. 4** Communication in a fully connected topology. Red represents sent, blue returned (overwritten) best solution

**Fully connected.** In this case, *n* colonies with different random seeds are simulated and they communicate with each other with the goal of finding the good solutions. The interaction between colonies is done in the following fashion. The best-so-far solutions are collected from all the colonies. The best overall solution, or in a variation the best colony index, is found and it is sent to all *n* colonies which set their own best-so-far solution to it.

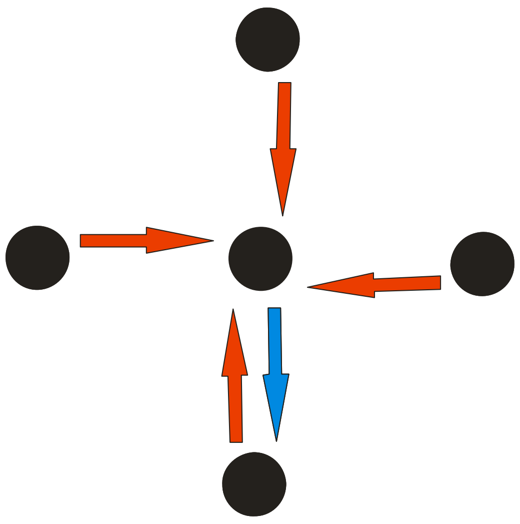

**Fig. 5** Communication in a replace worst topology. Red represents sent, blue returned (overwritten) best solution

**Replace worst.** In this case, we again search for the best overall solution for all *n* colonies, but we also find the colony with the worst solution. Instead of sending the best solution to all the colonies, it is only sent to the worst colony which sets a new best solution. This approach has an advantage compared to a fully connected topology of lower amount of communication between the colonies.

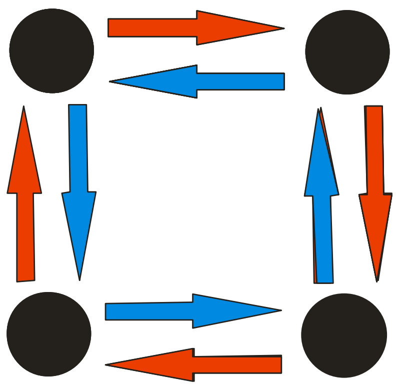

**Fig. 6** Communication in a ring topology. Red represents sent, blue returned (overwritten) best solution

**Ring.** This method of communication is inspired by the ring topology in which a colony only communicates with neighboring colonies. In a colony cluster with *n* colonies the *k* indexed colony will only give its best-so-far solution to the [*(k-1) mod n*] indexed colony, and receives it from [*(k+1) mod n*] indexed colony. This approach has greatly smaller level of communication then the two previously mentioned methods.

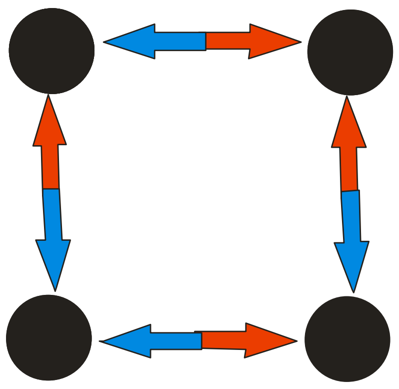

**Fig. 7** Communication in a ring switch topology. Red blue arrows represent the exchange of best solution between colonies

**Ring switch.** Is very similar to the ring method in the sense that each colony communicates only with its neighbors. A big difference to the three already mentioned methods is that colonies with low quality solutions do not overwrite them with a better solution but instead an exchange is done. In this way, the lower quality solutions are not lost, but are used in combination with pheromone matrixes from other colonies. In practice colony with index *k* only exchanges its solution with colony [*(k-1) mod n*].

**Independent parallel runs.** This implementation has no communication at all between colonies. It runs the same sequential ACO algorithm with different random seeds in *n* different processes. The solution it takes is the best solution of all the independent runs. This method has the advantage that no extra code is needed for the parallelization.

A very interesting adaptive approach to communication connections between colonies is given by Ling Chen and Chunfang Zhang [27] for parallel implementation of ACO. The connections are not fixed like in previously mentioned topologies but depend on the currently found best solutions for all colonies. They propose two methods for establishing connections between colonies that exchange data. In the first, the solutions found by colonies are sorted and ranked. The next step is exchanging solutions between colonies with the best and the worst rank, then the second best and the second worst, and so on. In the second model colonies with the most different solution exchange their solutions. In their implementation the time interval of data exchange is also adaptive to improve performance. In our tests we did not implement this adaptive algorithm due to the relative complexity but we believe that it is necessary to mention it in the context of topologies.

## 5 Implementation of Parallel ACO Algorithms

In this section, we analyze methods of communication for parallel algorithms presented in the previous section and their effectiveness in ACO for MWVCP. All of the topologies have been implemented using our framework from article [28, 29]. This framework is dot net based and is designed for creating windows applications. It is implemented as a plug-in system so similar research on parallelization can be conducted on other problems that could be solved by ACO just by creating a sequential version of the algorithm. We have created a plug-in for this system and used existing features to conduct our tests. The executable alpha version of this software (Figure 8) and accompanying Microsoft Visual Studio project can be downloaded from http://mail.phy.bg.ac.yu/~rakaj/home/. All of our test have been performed on an Intel(R) CoreTM(2) CPU 6400 @ 2.13 GHz with 4GB of RAM with Microsoft Windows XP Professional x64 Edition Version 2003 Service Pack 2.

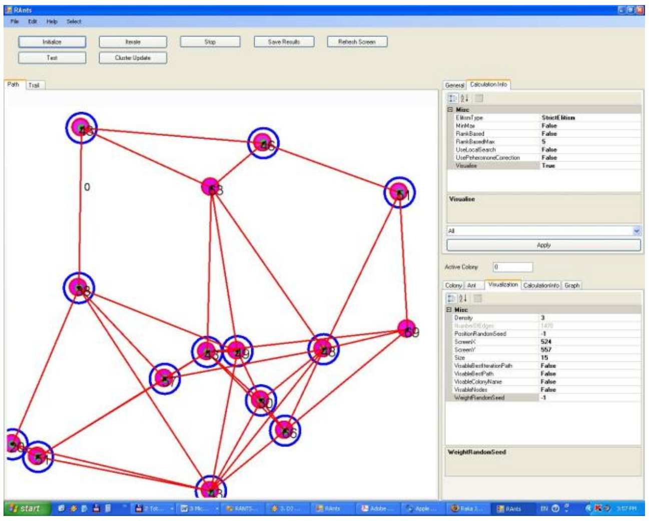

**Figure 8. Graf-Ant software with the plug-in for Minimal Weighted Vertex Covering Problem**

Parallelization has been implemented by creating different threads for each colony and one thread that is used as a colony cluster, a master class that executes the communication between different colony threads. This implementation is not a perfect representation of a true parallel execution of different topologies on multiprocessor machine or machines in a network. The main drawbacks are that communication between colonies is done without delay, there cannot be loss of data in communication between colonies or unexpected termination of execution of some colonies. Because of this, we focus our attention to the quality of the results these topologies give, rather than the speed.

Our software implementation was done in C#. The parallelization code was controlled by two main classes *RAntMWVCPColonyCluster*, and *RAntMWVCPColony*. RAnt*MWVCP*Colony is dedicated to a single colony and the execution of sequential ACO. *RAntMWVCPColonyCluster* is used for regulating the parallel runs of several colonies, in the sense of starting threads, collecting results, communicating data amongst colonies. We observe the pseudo-code of the method that initializes the all colonies.

```
void StartColonies(){

   ResetBestPath ();
     for (int i = 0; i < NumberOfColonies; i++)
{
        Colonies[i].InitializeProblem(P);
        StartThread(Colony[i].SimulateColony)
     }
    InitializeTimer(time, InformationExchange);
}
```

It first resets colony cluster data using the *ResetBestPath* method. Next, it initializes colony properties for the tested problem and starts a thread for each colony. When all the threads are running a periodic timer is created that initializes information exchange, executing the function given by the following pseudo-code

```
void InformationExchange(){

      GetBestPath();
      ActiveExchangeMethod();
      BestPath = Colonies[BestColony].Path;
      BestValue = Colonies[BestColony].Value;
      SaveClusterStatus();
}
```

This method calculates the best found solution and the colony to which it belongs, executes data exchange for the active topology, stores the new best solution, and leaves any needed log information.

## 6 Experimental Results for Parallel ACO Algorithms

We tested different sized problem instances with 50, 100 and 150 nodes. We also tested the effect of different sized colony clusters with 5 or 10 colonies working together. In all the cases each colony uses the Elitist Ant variation of ACO as presented in articles [16], [11]. In Tables 1, 2, 3, 4, 5, 6, all topologies have been given the same calculation time and the information exchange has been done periodically every *n* time periods. Tables 1 and 2 are for 50 node problems, Tables 3 and 4 for 100 node problems and Tables 5 and 6 for 150 nodes problems. Tables 1, 3 and 5 present results for 5 colony tests and Tables 2, 4 and 6 for 10 colony cases.

When comparing the sequential algorithm to the parallel versions, we used the standard approach of giving them the same time of execution. We compare the best solution and solution average of $F_t^k(c)$ running $k$ times with communication presented in the previous section, to $F_{tk}(c)$ running once for time $t*k$. $c$ is a problem instance. This puts the sequential algorithm in a partially disadvantaged position compared to the parallel algorithms because of the relatively long execution time. As it is mentioned in articles [4, 14], sequential algorithms perform better compared to parallel ones if calculation time is shorter. We can see the result for the sequential algorithm in the Tables 1, 3, 5 and compare them to results of clusters of 5 colonies. In all of our tests we simulated 5 separate runs for each parallel topology and the sequential algorithm. We compared the average and best found solution.

**Table 1**. Number of nodes 50, Number of edges 209, Number of Colonies **5**, greedy algorithm solution value 2038, Calculation Time 1 minutes, and communication every 6 sec

| **Topology** | **Best Value** | **Average** |
|---|---|---|
| Fully Connected | 1712 | 1738.8 |
| Replace Worst | 1701 | 1725.0 |
| Ring. | 1660 | 1729.2 |
| Ring Switch | 1667 | **1704.6** |
| Ind. Parallel Runs | 1695 | 1719.2 |
| Sequential | 1730 | 1749.2 |

We first observe the solution quality for the smaller case with 50 nodes, Table 1. We wish to point out that all the parallel implementation gave better quality solutions then the sequential algorithm. The main reason for this is that the sequential algorithm started stagnating relatively early in the solution search process. If we used shorter execution time, the difference between the sequential and the parallel algorithms would have been smaller.

**Table 2**. Number of nodes 50, Number of edges 209, Number of Colonies **10**, greedy algorithm solution value 2038, Calculation Time 1 minutes, and communication every 6 sec

| **Topology** | **Best Value** | **Average** |
|---|---|---|
| Fully Connected | 1735 | 1747.0 |
| Replace Worst | 1701 | 1730.6 |
| Ring. | 1672 | 1722.6 |
| Ring Switch | 1673 | **1712.4** |
| Ind. Parallel Runs | 1699 | 1719.8 |

In the small problem case, the fully connected approach gave poor results compared to other topologies. This can be explained by the fact that the search started focusing on some bad initial solution, in some runs. On the other hand, focusing all the colonies on one good solution was not needed because of the relatively small solution space. We can notice that keeping the diversity of the search in the small problem case was of great importance as the results indicate that the bigger the diversity of the search, the better the final solution was.

**Table 3**. Number of nodes 100, Number of edges 450, Number of Colonies **5**, greedy algorithm solution value 4548, Calculation Time 2 minutes, and communication every **10** sec

| **Topology** | **Best Value** | **Average** |
|---|---|---|
| Fully Connected | 3470 | 3521.6 |
| Replace Worst | 3493 | 3519.0 |
| Ring. | 3464 | 3516.8 |
| Ring Switch | 3464 | **3503.6** |
| Ind. Parallel Runs | 3503 | 3540.0 |
| Sequential | 3542 | 3583.4 |

In the case of the bigger problem, the focusing of the search gave better results because the focused area was big enough for different colonies not to search over the same regions. All the topologies that used intensified searches near good solutions gave similar results. Due to the larger solution space, the parallel independent runs lost its advantage to these methods but still gave good results.

**Table 4**. Number of nodes 100, Number of edges 450, Number of Colonies **10**, greedy algorithm solution value 4548, Calculation Time 2 minutes, and communication every **20** sec

| Topology | Best Value | Average |
|---|---|---|
| Fully Connected | 3462 | 3502.8 |
| Replace Worst | 3499 | 3524.8 |
| Ring. | 3499 | 3507.8 |
| Ring Switch | 3460 | **3491.0** |
| Ind. Parallel Runs | 3493 | 3505.8 |

Our second set of tests where on testing the effect of increasing the number of colonies from 5 to 10 and using the same calculation time. In the small problem case (Tables 1, 2) the increase was a bad step and did not improve the quality of results. This can be explained by the relatively small solution space and because of the fact that colonies would be exploring the same areas. In the case of medium sized problems (Tables 3, 4) results where similar, but slightly better with a higher number of colonies.

**Table 5**. Number of nodes 150, Number of edges 450, Number of Colonies **5**, greedy algorithm solution value 6782, Calculation Time 4 minutes, and communication every 30 sec

| Topology | Best Value | Average |
|---|---|---|
| Fully Connected | 5672 | 5763.2 |
| Replace Worst | 5634 | 5735.6 |
| Ring. | 5601 | **5726.0** |
| Ring Switch | 5643 | 5738.8 |
| Ind. Parallel Runs | 5746 | 5735.6 |
| Sequential | 5788 | 5852.6 |

Finally, in large problem cases (Tables 5, 6) the increase of the number of cooperating colonies significantly worsened the solution quality even getting it near to the sequential algorithm. In this case, we believe that the problem was that none of the colonies had sufficient time for improving solutions with intensive search near good solutions. Instead, just a wide range of areas was poorly tested. This indicates that depending of the problem in question, there is an optimal proportion between the number of colonies and the time or equivalent number of iterations dedicated to each colony.

**Table 6**. Number of nodes 150, Number of edges 450, Number of Colonies **10**, greedy algorithm solution value 6782, Calculation Time 4 minutes, and communication every 30 sec

| Topology | Best Value | Average |
|---|---|---|
| Fully Connected | 5779 | 5823.0 |
| Replace Worst | 5800 | 5836.6 |
| Ring. | 5786 | 5821.4 |
| Ring Switch | 5765 | 5804.0 |
| Ind. Parallel Runs | 5754 | **5788.2** |

In our tests, the ring switch algorithm we proposed has calculated, on average, the best results. The ring switch algorithm has performed somewhat worse in the case from Table 5. We expect that this is a consequence of a relatively small number of tests conducted and consider it a statistical error. The good performing of ring switch can be explained by qualities of this algorithm. First, diversity of the search is not quickly lost because no solutions found so far are overwritten and disregarded. This is its advantage to algorithms that focus the search near good solutions like fully connected, replace worst and ring. The effect of the exchange still moves the search of the colony cluster in a good direction and is not kept in the same areas as in parallel independent runs. We can observe what happens in each of the two colonies *A* and *B* affected by the exchange. Let us say that the colony *A* has had a better best-so-far solution than colony *B*. After the exchange, *B* will have a solution better than before and in the worst case, in later iterations after enough pheromone has been deposited, search the same area as colony *A* had before the exchange. In the case of colony *A* that has gotten a solution worse than the one it head before, it is possible for it to end up searching the same space as colony *B* before the exchange, after a high enough number of iterations. This is, however, not a very likely consequence of the exchange for this colony. The individual ants search paths do not directly depend on best-so-far solution but from the pheromone trail. This trail will slowly change from the good solution trail (from colony *A*) to a worse trail (colony *B*) and in this period it is highly likely for some ant to find a solution that has better quality than colony *B* had at the beginning.

## 7 Conclusion

We used our previously developed framework [26, 27] to create software for conducting tests. We compared the effect of different parallel algorithms for the MWVCP. We have confirmed that, similar to the case of the TSP, the simple use of parallel independent runs is a good approach. In small problem cases it was even better that other, more complicated topologies like fully connected, replace worst and the ring. In larger problem cases, this advantage has been lost, but the results were still of good quality. We also introduced a variation of the algorithm corresponding to the ring topology. In this variation instead of overwriting lower quality solutions an exchange was conducted between neighboring colonies. This proved to be a good choice because the diversity did not quickly disappear and the search of the colony cluster was moving towards areas with better solutions. In our tests, ring and ring switch algorithms gave the best results with ring switch being slightly better.

We implemented the parallelization through the use of threads on a Windows platform. Even in the case of parallelization simulated by Windows on a single processor the results were better than when using a sequential algorithm.

In further research, we wish to adopt and implement the suspicion path removal hybridization used on the TSP to this problem. We will also extend our ant colony framework with the Adaptive Parallel Ant Colony Algorithm, and compare with our previous results.

*References:*

[1] R.Tanese, Parallel genetic algorithms for a hypercube. *Proceedings of the second international conference on Genetic Algorithms and their Applications,* Hillsdale, NJ, Lawrence Erlbaum Associates, Inc, 1987, pp. 177–183

[2] Dorigo M, Gambardella LM: Ant colonies for the traveling salesman problem. *BioSystems* Vol. 43 No.2 ,1997, pp.73-81.

[3] Thomas Stützle, Parallelization strategies for Ant Colony Optimization, *Parallel Problem Solving from Nature - PPSN V*, Springer Berlin / Heidelberg, 1998, pp. 722-731

[4] Max Manfrin, Mauro Birattari, Thomas Stützle and Marco Dorigo, Parallel Ant Colony Optimization for the Traveling Salesman Problem, *Ant Colony Optimization and Swarm Intelligence*, Springer Berlin/Heidelberg (2006), pp. 224-234

[5] Shigeyoshi Tsutsui, Parallel Ant Colony Optimization for the Quadratic Assignment Problems with Symmetric Multi Processing, Proceedings of the 6th international conference on Ant Colony Optimization and Swarm Intelligence, Springer-Verlag Berlin, Heidelberg, pp. 363 - 370, 2008

[6] Mohammad Towhidul Islam, Parimala Thulasiraman, Ruppa K. Thulasiram, A Parallel Ant Colony Optimization Algorithm for All-Pair Routing in MANETs, Parallel and Distributed Processing Symposium, International, pp.259a, 2003

[7] T. Vetri Selvan, P. Chitra, P. Venkatesh, Parallel Implementation of Task Scheduling using Ant Colony Optimization, International Journal of Recent Trends in Engineering, Vol. 1, No. 1, pp. 339-343, 2009

[8] Xie Hongwei, Luo Yanhua, "Parallel ACO for DNA Sequencing by Hybridization," csie, vol. 4, pp.602-606, 2009 WRI World Congress on Computer Science and Information Engineering, 2009

[9] Bullnheimer, B., Kotsis, G., Strauß, C, Parallelization strategies for the Ant System. *High Performance Algorithms and Software in Nonlinear Optimization.* Kluwer Academic Publishers, Norwell, MA (1998) pp. 87–100

[10] Middendorf, M., Reischle, F., Schmeck, H.: Multi colony ant algorithms. *Journal of Heuristics* Vol. No.3, 2002, pp, 305–320

[11] Benkner, S., Doerner, K.F., Hartl, R.F., Kiechle, G., Lucka, M.: Communication strategies for parallel cooperative ant colony optimization on clusters and grids. *Complimentary Proceedings of PARA'04 Workshop on State-of-the-Art in Scientific Computing*, June 20-23, 2004, Lyngby, Denmark 2005, pp. 3 - 12

[12] Karp, R.M.. Reducibility Among Combinatorial Problems. In R.E. Miller and J.W. Theater, *Complexity of Computer Computations,* New York: Plenum Press, 1972

[13] Chvatal, V.. A Greedy-Heuristic for the Set Cover Problem. *Mathematics of Operations Research,* Vol.4, 1979, pp. 233–235.
[14] Clarkson, K.L. A Modification of the Greedy Algorithm for Vertex Cover. *Information Processing Letters,* Vol. 16, 1983, pp. 23–25.
[15] Ashok Kumar Gupta, Alok Singh, A Hybrid Heuristic for the Minimum Weight Vertex Cover Problem, *Asia-Pacific Journal of Operational Research,* 2006, vol. 23, No 2, pp 273-285
[16] Shyong Jian Shyu, Peng-Yeng Yin, Bertrand M.T. Lin, An Ant Colony Optimization Algorithm for the Minimum Weight Vertex Cover Problem, *Annals of Operations Research,* Vol. 131, 2004, pp. 283–304,
[17] Vlachos Aristidis, An Ant Colony Optimization (ACO) algorithm solution to Economic Load Dispatch (ELD) problem. *WSEAS Transactions On Systems,* Vol 5, No 8, pp. 1763 – 1771, 2006
[18] Kolahan, F., Abachizadeh, M., Soheili, S, A comparison between Ant colony and Tabu search algorithms for job shop scheduling with sequence-dependent setups, *WSEAS Transactions on Systems,* Vol. 12, pp. 2819- 2824, 2006
[19] Mastorakis, N.E., Zhuang, X, Image processing with the artificial swarm intelligence, *WSEAS Transactions on Computers*, Vol 4, No. 4, pp. 333-341, 2005
[20] Raka Jovanovic, Milan Tuba: A Comparative Assesment of Ant Colony Optimization Algorithms for the Minimum Weight Vertex Cover Problem, in Recent Advances in Artificial Intelligence, Knowledge Engineering and Data Bases, A Series of Reference Books and Textbooks, Artificial Intelligence Series, WSEAS Press, Cambridge, United Kingdom, 2009, pp. 490-494
[21] Milan Tuba, Raka Jovanovic: An Analysis of Different Variations of Ant Colony Optimization to the Minimum Weight Vertex Cover Problem, WSEAS Transactions on Information Science and Applications, Issue 6, Volume 6, June 2009, pp. 936-945
[22] M. Bolondi, M. Bondaza: Parallelizzazione di un algoritmo per la risoluzione del problema del comesso viaggiatore; Master's thesis, Politecnico di Milano, 1993
[23] B. Bullnheimer, G. Kotsis, C. Strauß, Parallelization strategies for the ant system. Report Series SFB "Adaptive Information Systems and Modeling in Economics and Management Science", Nr. 8, 1997
[24] E. Ridge, D. Kudenko, D. Kazakov, E. Curry, Moving Nature-Inspired Algorithms to Parallel, Asynchronous and Decentralized Environments, Proceeding of the 2005 conference on Self-Organization and Autonomic Informatics, IOS Press, Amsterdam, The Netherlands, pp. 35--49, 2005
[25] E. Cantu-Paz, Efficient and Accurate Parallel Genetic Algorithm. Kluwer Academic Publishers, Boston, 2000
[26] T. Stutzle: Parallelization strategies for ant colony optimization; in: A. E. Eiben, T. Back, M. Schonauer, H.-P. Schwefel (Eds.), Parallel Problem Solving from Nature - PPSN V, Springer-Verlag, LNCS 1498, 722-731, 1998.
[27] Ling Chen, Chunfang Zhang , Adaptive Parallel Ant Colony Algorithm, Advances in Natural Computation, Springer Berlin / Heidelberg, pp. 1239-1249, 2005
[28] Raka Jovanovic, Milan Tuba, Dana Simian: Developing an Object-Oriented Framework for Solving Problems Using Ant Colony Optimization, in *Computers and Simulation in Modern Science*, Volume II, A Series of Reference Books and Textbooks, Mathematics and Computers in Science and Engineering, WSEAS Press, Bucharest, 2008, pp. 94-99
[29] Raka Jovanovic, Milan Tuba, Dana Simian, An Object-Oriented Framework with Corresponding Graphical User Interface for Developing Ant Colony Optimization Based Algorithms, *WSEAS Transactions on Computers*, Vol. 7, No. 12, 2008, pp. 1948 - 1957

**Acknowledgment:** This research is supported by Projects 144007 and 141031, Ministry of Science, Republic of Serbia.