Vietnam J Comput Sci (2016) 3:103–117
DOI 10.1007/s40595-016-0059-z

REGULAR PAPER

Using hyper populated ant colonies for solving the TSP

Andrzej Siemiński[1]

Received: 19 October 2015 / Accepted: 18 February 2016 / Published online: 10 March 2016

Abstract The paper discusses the application of hyper populated ant colonies to the well-known traveling salesman problem (TSP). The ant colony optimization (ACO) approach offers reasonably good quality solutions for the TSP, but it suffers from its inherent non-determinism and as a consequence the processing time is unpredictable. The paper tries to mitigate the problem by a substantial increase in the number of used ants. This approach is called ant hyper population and it could be obtained by increasing the number of ants in a single colony assigning more than one colony to solve the same task or both. In all cases the level of non-determinism decreases and thus the number iterations could be reduced. Parallel implementation of the ACO makes it possible to reduce drastically the processing time. The paper compares two ways of implementation of the parallelism using the sockets or the RMI—remote method invocation mechanisms. The paper concentrates on the classical static version of the TSP, but preliminary experiments indicate that such an approach could be even more useful for dynamic TPSs.

Keywords Ant colony optimization · Traveling salesmen problem · Parallelization strategies for ACO · Ant colony community (ACC)

1 Introduction

The aim of the paper is to discuss the problem of optimizing the performance of the ant colony optimization (ACO) used for the traveling salesman problem (TSP). Metaheuristics such as the ACO solve a complex problem by iteratively improving candidate solutions. They do not guarantee the selection of an optimum or even a satisfactory near-optimal solution. In the case of the NP-hard or even NP-complete problems they are often the only available choice that we have.

Usually the quality of solutions is measured by exclusively by the length of the selected route. The ACO works in a non-deterministic way. Usually a predefined number of iterations are executed and the best found solution is selected. The quality of solutions improves with time, but the process is not uniform. A high quality solution could be found after a few thousands of iterations, but it is not uncommon that we can have it after just a few dozens of them. Therefore, it is so hard to tell when to stop the operation of the ACO. There is a great incentive to prolong the execution time. The reported experiment results look more impressive. This is certainly possible in an university environment. It may not be the case for the real life applications. The time needed to execute a great number of iterations may simply not be available. Moreover, the longer we run the ACO, the solution updates are the less and less frequent. This demerit is even more acute for the dynamic TSP. In that case the ACO may not be able to catch up with the changing environment.

The paper addresses the problem using a drastically increased number of ants. In what follows such ACOs are called hyper populated Ant Colonies. They come in two flavors. In the first one the ant number increases in just one colony and in the second we have a number of cooperating colonies—the so-called ant colony community (ACC). In both cases the results converge faster so the number of iterations could be limited. The main contribution of the paper is a detailed presentations of a model for the ACC and verification of its efficiency. The Socket mechanism was used to

✉ Andrzej Siemiński
Andrzej.Sieminski@pwr.edu.pl

[1] Faculty of Computer Science and Management, Wroclaw University of Technology, Wroclaw, Poland

implement the model and its advantages over the competing RMI mechanism are discussed. The complexity of interactions between individual ants or even ant colonies make a theoretical analysis extremely hard even with a number of simplifying assumptions. Therefore, this paper demonstrates empirically the performance and convergence aspects of the proposed model.

The paper is organized as follows. The second Section briefly introduces the TSP. The next one is devoted to the ACO—a metaheuristics commonly used to solve it. This section presents its general operational principles and the role played by its parameters. Attempts to optimize their values are also mentioned. It ends with the discussion of the stopping conditions. In the fourth Section we provide experiment results that justify the increase in the number of used ants. The prolonged execution time resulting from increasing of ant population could be mitigated by the parallel implementation of the ACO. The fifth Section contains the taxonomy of parallel ACOs and introduces the coarse-grained ACC proposed in the paper. The conducted experiments, their results with the criteria used to evaluate them are presented in the 6th Section. The paper concludes with the resume of research work done so far and the plans for future investigation.

2 Traveling salesman problem specification

The TSP could be stated in a remarkably simple way: given a list of cities and the distances separating them what is the shortest possible route that visits each city exactly once and returns to the origin city? For the first time the problem was stated as early as in 1800's. At that time it was treated as a recreational puzzle, papers printed graphs and prized best solution sent by readers. Nowadays it has many practical implications in areas as diverse as optimizing scheduling of a route of the drill machine used to drill holes in a printed circuit board or minimizing material wasted in the cutting-stock problem.

The number of all possible different routes for a graph with n nodes is equal to (n-1)!/2. The number is estimated by $\sqrt{2\pi n}(\frac{n}{e})^n$ and even for relatively small values of n like 50 the value is just staggering. It is equal approximately to 3,04141E+64 which is far exceeds the mass of an observable steady-state universe is 1,45E+53 kg. This clearly calls for heuristic solutions as the complete search is not feasible.

The TSP is now one of the established, classical problems of Artificial Intelligence and serves as a touchstone for many general heuristics devised for combinatorial optimization. in 1970's it was proved to be a NP-hard problem. A recent comparison of metaheuristics used for TSP could be found in [1]. The paper discusses: genetic algorithms, simulated annealing, tabu search, quantum annealing, particle swarm optimization, harmony search, a greedy 2-opt interchange algorithm and the last, but not the least the ACO.

In a classical statement of the problem the distances between nodes are symmetric and do not change. This seems to match the real life where the road structure remains relatively static. Having said that we should bear in mind that minimizing the distance is not what we have really interest in. In more practical objectives include, e.g., the traveling time which is subject to the ever changing road conditions and therefore is inherently dynamic. The Dynamic TSP was introduced for the first time by Psarafits [2] and various aspects of the DTSP are now the subject of intensive study [3–5].

3 Ant colony optimization

Scientists were for a long-time puzzled by way the in which tiny, week and blind creatures, e.g., the termites, were capable of building and operating extremely complex, city like structures such as termite nests. The explanation was proposed in late 50' of the previous centaury by the French biologist Pierre-Paul Grassé. He coined the term stigmergy to describe a mechanism of indirect coordination between agents or actions. In the real world it could produce complex, seemingly intelligent structures, without the need for any planning, control, or even direct communication between the agents. The agents have very little or no memory. They lack intelligence or even awareness of each other. What makes them so capable is the pheromone trail that is deposited in the environment. The extreme simplicity of ants combined with their apparent ability to produce complex structures make them very attractive for computer science.

The ACO technique was introduced by Dorigo in as early as in 1992 [6]. Until now he remains one of the key researchers in this area. His extensive and fairly recent account of the ACO state of the art is presented in [7]. An ant colony consists of ants which are extremely simple agents. All they can do is to move from one node to another laying a pheromone trail on their way. They are also capable of detecting their current position, remembering the nodes that were already visited and sensing the direct distances from its current position to other nodes as well as the amount of pheromone laid upon them. The colony works in iterations. At the start of each iteration the ants are placed randomly on the graph. Each ant works on its own, completing a route that connects all nodes. In each step of an iteration an ant, sensing the distance and pheromone levels placed on routes connecting nodes, selects the next node to visit. The iteration stops when all cities are visited by all ants. The pheromone matrix harvests the collective experience gained by the ants. This general idea has many variants such as ant systems, ant colony systems, Max–Min ant systems. They differ mainly in exact way pheromone is deposited and the role played

by the colony. In each case the colony remembers the BSF (Best So Far) route, the current iteration number and the iterations' best route. The colony is responsible also for global pheromone updating.

3.1 Operation details

The distances between the cities (nodes) are represented by a square matrix of floating point numbers. The number of rows or columns is denoted by N. The element in row r and column s is denoted by $\delta(r, s)$ and is known also as a cost measure. The matrix is symmetric so $\delta(r, s) = \delta(s, r)$ and additionally for all nodes r $\delta(r, r) = 0$.

The pheromone levels are stored in an another matrix of floating point numbers which has the same size of the distances matrix. Its elements are denoted by $\tau(r, s)$ and are referred to as the desirability measure. The algorithm guaranties, that allays $\tau(r, s) > 0$. The ACO initializes all elements of the matrix are with the same value at the beginning of its work. Pheromone levels are preserved from one iteration to another.

There are three rules that define the operation of the ACO:

- the State Transition Rule that specifies the next node an ant selects, see Formulas 1 and 2;
- the Local Updating Rule which updates the pheromones as an ant moves from one node to another, see Formula 3;
- the Global Updating Rule which defines the way in which the pheromones are updated when all ants have constructed whole route, see Formula 5.

The operation of the ACO is controlled by five parameters. They are shown in the Table 1. The complexity of the ACO operation so great that it is impossible to provide an analytical way of selecting their optimal values. Therefore, they are usually chosen in an experimental manner. The table contains also their recommend used by many researches, e.g., by Chirico [8].

Table 1 ACO parameter description

Name	Description	Suggested value
N	Number of ants	Number of nodes
Q0	Probability of selecting exploitation over exploration	0.8
α	Aging factor used in the global updating rule	0.1
β	Moderating factor for the cost measure function	2.0
ρ	Aging factor in the local updating rule	0.1

An ant selects the next node using one of two possible operation modes: exploitation and exploration. In the exploitation mode an ant works in a deterministic manner. Staying in the r node an it selects the node t which maximizes the route quality function qf:

$$qf(r, t) = \tau(r, t) * \eta(r, t)^{\beta}. \tag{1}$$

The pheromone levels represent the collective knowledge of all ants. The influence of the distance depends of the parameter β. It is greater than 1. The distance between two nodes is always ≤ 1 therefore increasing β gives more prominence to pheromone level. Note, that to find the next node the values of the $qf(r, t)$ function for all possible (not yet visited) nodes have to be calculated.

By contrast, the exploration is a non-deterministic process. It selects the next t node with the probabilities defined by the Formula 2:

$$pr(r, t) = \frac{qf(r, t)}{\sum_{u \in A(r)} qf(r, u)} \tag{2}$$

The exploration algorithm prefers to choose nodes with the highest value of the qf function but it could, with a lesser probability, select any other available node. Their set is denoted by $A(r)$. Exploration is used to search for alternative solutions and to mitigate the danger of a colony being trapped in a local minimum. The parameter Q0 specifies the probability of selecting the exploitation mode of operation. The selection of operation mode is done each time a node is to be selected. As you can see the operation of the ACO is non-deterministic. A random number generator is used both for the selection of the operating mode and to select a node in the exploration mode. Usually the ants are implemented as threads and this is also introduces non-determinism.

Due to the sheer number of time-consuming floating point operations that are necessary to select the next node the ACO algorithm is relatively slow.

The pheromone levels are changed applying local and global updating rules. The local updating rule is used on the fly by each ant as it moves from one node to another. The global updating is done by the colony after an iteration step was completed. Let BSF (Best So Far) denote the best path found so far by any ant and L(BSF) denote its length. The Formula F3 defines the local updating rule:

$$\tau(r, s) = (1 - \rho) * \tau(r, s) + \rho * \Delta(r, s) \tag{3}$$

where:

$$\Delta(r, s) = \begin{cases} L(BSF)^{-1} & \text{if}(r, s) \in \text{global-best-tour} \\ 0 & \text{otherwise} \end{cases}. \tag{4}$$

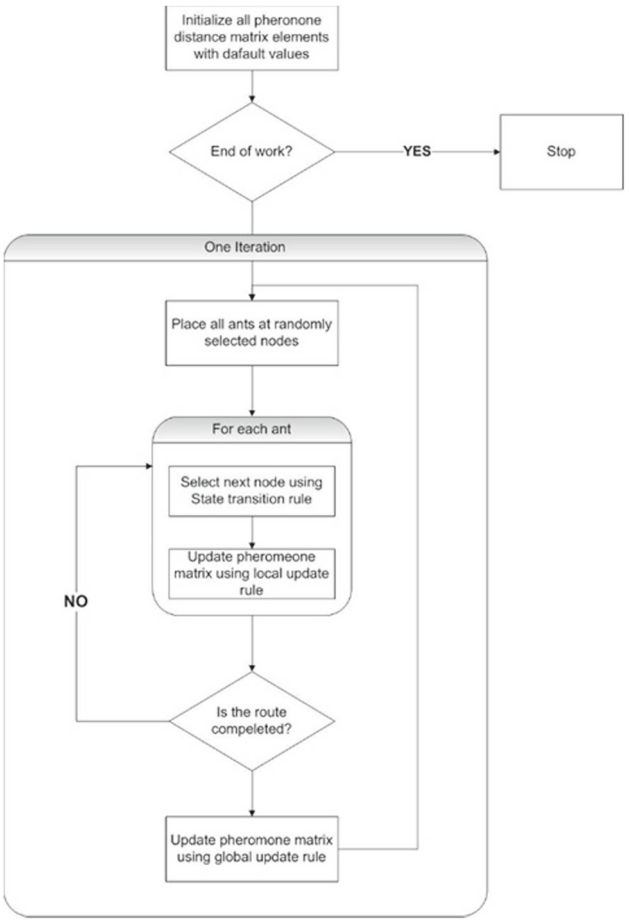

Fig. 1 Schema of the ACO operation

Taking a route that does not belong to the BSF results in decreasing the level of its pheromone. This simulates the process of pheromone evaporation. Otherwise the range of modification depends on the quality of the BSF solution. The shorter it is, the more impact it has on the resulting pheromone level. The evaporating intensity is controlled by the parameter ρ.

The global updating function is evoked after each iteration, It changes the pheromone level on all routs in the graph. The Formula 5 specifies the level modification:

$$\tau(r,s) = (1-\alpha) * \tau(r,s) + \alpha * \Delta(r,s). \tag{5}$$

The Formula 5 is very much like the Formula 3, the only difference is that it evaporation intensity is controlled by still another parameter α. For $\alpha = 1$ the ACO has no memory of previous results. With consecutive numbers of iterations the value of $\Delta(r,s)$ increases and therefore ACO favors exploration at start of its work and later the found solution become more and more stable as the exploitation becomes more prominent.

The initial values of the pheromone level for all paths (τ_0) of a network with n nodes are not parameterized and are calculated using the Formula 6:

$$\tau_0 = \frac{2n}{\sum \delta(r,s)}. \tag{6}$$

The operation of an ant colony is shown on the Fig. 1. The end of work condition usually tests the total number of completed iterations or checks if no shortening of the BSF route has been reported over a predefined number of iterations.

3.2 ACO Optimization

Although the operation of each ant is simple their interplay is complex. Therefore, the selection of the values for the parameters from the Table 1 is not possible in any analytic manner. The paper [9] reports that their strikingly different values could lead to solutions of similar quality. What makes the process even more complex is the non-deterministic character of the ACO. The solutions found for the same static graph and the same set of parameter values could differ even after many thousands of iterations.

The recommended parameter values have round values like 0.8 or 2.0. This gives rise to an assumption that they are not optimal. Various attempts to identify values offering better results were described, e.g., in [10] or [11]. They include algorithms inspired by evolutionary programming (EP), simulated annealing (SA) or a statistical analysis of a large collection of gathered results. The results are not quite satisfactory. It possible to find values that lead to better results than the results achieved with recommended values, but it was not possible to correlate them to the properties of the environment. In each particular case has to be treated individually and the optimization of parameters values requires time-consuming experiments.

The route length is not the only quality measure. The other one is the execution time. In the study reported in this paper we examine the way in which the number of used ants and their organization impacts the processing time and route length.

3.3 Stopping Problem

No matter what the parameter values are used the processing has to be stopped at same point of time. When to stop is a question that is rarely raised in the discussion of the ACO. The frequency of changes of the BSF route lessens as the number of iterations increases. This means that the computational effort needed to execute subsequent iterations is less and less profitable. This is clearly visible on the Fig. 2 which illustrates the typical performance of an ACO.

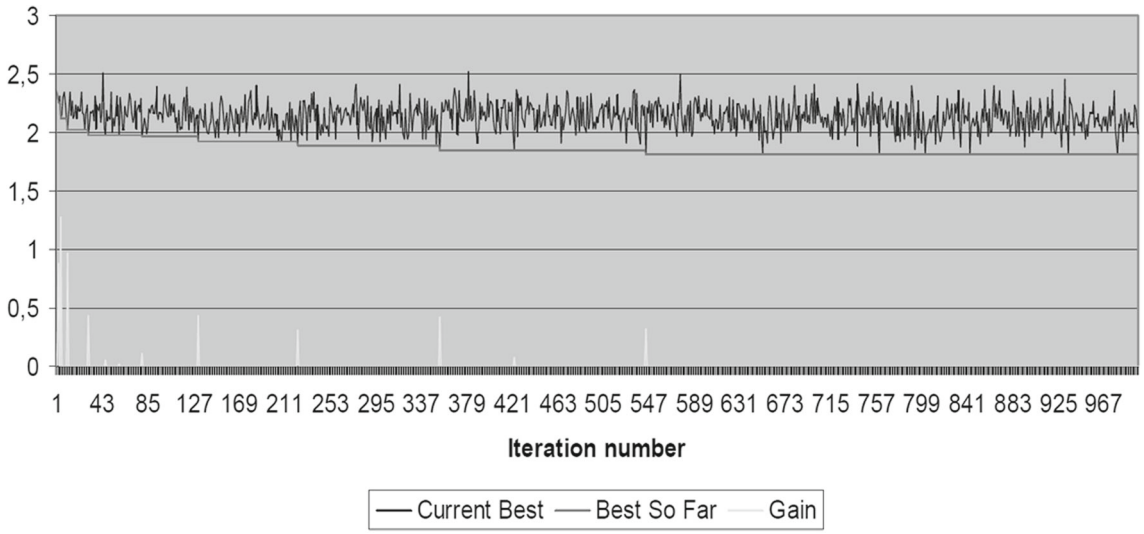

Fig. 2 The performance of a standard version of ACO [18]

Fig. 3 The cumulative gain in consecutive iterations for colonies with the number of ants in the range from 30 to 1000 [18]

The jigsaw upper line represents the best solution for consecutive iterations. There seems to be no clear pattern in its behavior. This makes extremely hard to predict the quality of the solution which the next iteration delivers. The lower, while spikes mark iterations for which there was a change in the BSF route length and their height is proportional to the shortening of L(BSF). They are more predictable. With the increasing number of iterations the spikes are less and less frequent. Note also that in the above example almost half of the processing time was not productive at all. There was no spike and hence no shortening of the BSF path after the iteration number 547.

To have a deeper insight into the performance of ACO we introduce the Cumulative Gain defied by the Formula 7. It describes the how much the solutions improve.

$$CG(i) = \left(\sum_{r=0}^{R} (\text{len}(r, 0) - \text{len}(r, i)) \right) / R \tag{7}$$

where R is the number of runs, i is the iteration number, len(r, i) is the BSF route length for the iteration i in the test run number r.

The Fig. 3 shows what impact the different sizes of an Ant Colony have on the Cumulative Gain.

The cumulative gain is the measure of the shortening of route not of its length. Therefore, the best performer (disregarding complexity of operation) is the colony with 1000 ants represented by the lower, dotted line. The sooner the values converge the better. Over populating the colonies offers a reasonable solution to the stopping problem. The routes length converge faster and there is much less incentive to extend the

Table 2 Observed average BSF route for varying colony populations, number of iterations, number of runs 30

Ant colony size	BSF fixed size	BSF normalized
30	1.82 [1000]	1.80 [1666]
50	1.81 [1000]	1.81 [1000]
80	1.80 [1000]	1.80 [625]
120	1.80 [1000]	1.85 [417]
150	1.79 [1000]	1.80 [333]
1000	1.78 [1000]	1.78 [50]

number of iterations. Note that in the above example for the 1000 ant colony there is hardly any improvement after the 320 iteration.

4 Increasing the number of ants

Many experiments have proven that the standard values of parameters provide a stable and reasonable good performance over a wide spectrum of input data. The recommended number of ants is equal to the number of nodes. For that reason the solution proposed in the paper keeps their standard values and studies the impact that the changes in the number of ants have on the performance of the ACO. The Fig. 1 suggests that the increase in the population size could be offset by the decrease in the number of iterations that are necessary to obtain stable and reasonably good results. In particular we are interested in using ant numbers that substantially exceed the recommended values. There are two ways in which the number is increased: single colony and multi-colony approaches.

Experiments conducted on a matrix with 50 nodes are reported in the Table 2 confirm that it is really possible to benefit from that phenomena. The Table shows average values of the BSF for different sizes of colony and varying number of iterations. The number of iterations is shown in square brackets. The iteration numbers in the BSF normalized column ware selected in such a way as to preserve the same the computational complexity of each raw.

The over populating of a single ant colony does not offer a significant reduction in route length. The middle column contains the route lengths obtained after 1000 iterations for all colonies. The slight decrease in route length, e.g., for the colony of size 1000 does not compensate the 20-fold increase in processing time. Observe, however, that the last column with the normalized values for the BSF. Increasing if the ant's number is compensated by lowering the number of iterations. For the hugely overpopulated colony with 1000 ants the number of iterations is as small 50. The decrease in the BSF is not impressive, but the small number of iterations paves the way for parallel implementation of the ACO.

In the multi-colony approach the ants are distributed over more than one colony. Their number in a single colony could also exceed regular values as well. Using many colonies working in parallel brings up two problems:

- The necessary communication overhead must not diminish the advantages of speed up due to parallelism.
- The lack or reduction of cooperation of ants from different colonies must have not much effect upon the quality of the resulting solution.

The first problem could be analyzed theoretically. Let us assume that the colony is implemented by a number of node computers and one host computer that coordinates their work. The operation of a parallel colony is characterized by the following factors:

- n – the number of ants.
- k – the number of node computers.
- m – the number of chunks of data.
- t_i – time required to process one chunk of data by one ant.
- t_d – time necessary to transmit data to and from the host and a node.
- t_c - time for the establishing the initial connection between the host and a node.

A parallel implementation reduces the processing time only if the number of node computers satisfies the inequity:

$$k > \frac{nt_i}{nt_i - t_d - \frac{t_c}{m}} \tag{8}$$

and

$$nt_i - t_d - \frac{t_c}{m} > 0 \tag{9}$$

For a continues mode of operation which is typical for dynamic environments the value of t_c/m is negligible. What is really required is that the local one node processing time is less than the time necessary to transmit the necessary data.

The study on the second phenomena is for more complex and is delayed until the Sect. 6. This Section describes the details of the proposed approach and interprets the obtained experimental results. The following Sect. 5 describes related work on parallel implementations of the ACO.

5 Parallel implementations of ACO

No matter what is the number of used ants, the ACO metaheuristic needs relatively long time to provide a solution. Therefore, the first attempts to shorten the processing time

by parallel implementation of the ACO were presented just a few years after the its introduction. The ACO depends on cooperation of individual ants. The main problem is: in what way to preserve some level of their cooperation in a parallel environment.

5.1 Performance measures

The efficient, algorithmic solutions for many complex real life problems are not known. Solving NP-hard optimization problems is compute-intensive. Therefore, often we have to resolve to parallel implementations. The reasons for that are twofold [12]. First they take benefit of using several computing elements to speed up the processing. Second we may introduce new exploration patterns that are not workable for the sequential implementations. In parallel algorithms that require a close cooperation between individual agents there is a need for specially designed hardware that can support it. Therefore, the parallel ACO is mainly implemented on traditional supercomputers, clusters of workstations, multicore processors and grid environments and recently graphics processing units [13].

The most common metrics used to evaluate the performance of parallel algorithms are the speed up (s_m) and the efficiency (e_m). The speed up indicates how much faster a parallel algorithm is than its corresponding sequential algorithm. In the case of non-deterministic implementations the mean values are used, see Formula 10 and

$$S_m = \frac{T_1}{T_m} \tag{10}$$

or for non-deterministic case

$$S_m = \frac{E\,[T_1]}{E\,[T_m]} \tag{11}$$

where T_1 and T_m denote the execution time of sequential algorithm and its parallel version using m processors.

The efficiency is the normalized version of the speed up and is introduced to enable the comparison of implementations using non-identical computing platforms.

$$e_m = \frac{S_m}{m}. \tag{12}$$

There are two factors that usually restrict the value of e_m to values less than 1. First is the well-known Amdahl's law [14] which limits the performance of any parallel application by the sequential part of the code. The second is the communication time overhead which could be quite considerable even on a specialized hardware. There were, however, reports, that for some specific problems, taking into advantage specialized hardware architecture and dedicated algorithm design it is possible to achieve the values of $e_m > 1$ [15].

In the paper [12] we have a comprehensive, up to date taxonomy of parallel ACO. The taxonomy consists of two broad categories: Master-slave model and Cellular model.

5.2 Taxonomy of parallel ACO

5.2.1 Master-slave model

This is a strictly a hierarchical parallel model in which a master process manages the global information including, e.g., the pheromone matrix or best-so-far solution. It also controls the slave processes that are responsible performing the actual search of the solution space. The model has three subcategories. The classification criterion is the granularity level that is the amount of work performed by each slave process.

- Coarse-grain master-slave model. The master manages the pheromone matrix and the interaction with the slaves is based on complete solutions. Each slave has one or more ants, and they compute complete solutions which are then communicated back to the master. The master can receive just one or many solutions from a slave. It selects the best solution or merges them.
- Medium-grain master-slave model. The key feature is the domain decomposition. The slave processes solve work on each sub-problem independently. The master process is responsible for managing the overall problem information and constructing a complete solution from the partial solutions reported by the slaves.
- Fine-grain master-slave model. The model requires the parallel evaluation of solution elements. The slaves perform minimum granularity tasks, e.g., selecting the next node. It is characterized by the frequent communications between the master and the slaves.

5.2.2 Cooperative models

In this group the colonies cooperate directly without the need of a master.

- Cellular model. The model uses follows the guidelines specified by the diffusion model employed in cellular evolutionary algorithm. A single colony is structured in small neighborhoods. Each one has its own pheromone matrix. The trail pheromone update in each matrix considers only the solutions constructed by the ants in its neighborhood. The model uses overlapping neighborhoods. This makes it possible to spread gradually high quality solutions from the place of their origin to other neighborhoods.

Table 3 Characteristics of the models in the new taxonomy [12]

Model	Population organization	# Colonies	# Pheromone matrices	Communication frequency
Coarse-grain master-slave	Hierarchical, non-cooperative	One	One	Medium
Medium-grain master-slave	Hierarchical, non-cooperative	One	One	Medium-high
Fine-grain master-slave	Hierarchical	One	One	High
Cellular	Structured, cooperative	One	Many	Medium
Parallel independent runs	Distributed, non-cooperative	Several	Several	Zero
Multi-colony	Distributed, cooperative	Several	Several	Low
Hybrids	Hierarchical	D/P	D/P	D/P

- Parallel independent runs model. The cooperation between colonies is not required. Several sequential ACOs are concurrently executed on a set of processors. The individual ACO can use identical or different parameters. The executions are completely independent.
- Multi-colony model. Several colonies explore in this model the search space using their own pheromone matrices. The colonies periodically exchange information.
- Hybrid models. Some papers describe algorithms that feature characteristics from more than one parallel model. This may include approaches that combine master-slave models or that introduce hierarchical structure into the basic model.

The basic features of the taxonomy are summarized in the Table 3. The D/P abbreviation stands for depending on proposal.

5.3 Examples of fine-grain implementations of ACO

The approach taken in this study is an example of coarse-grain master-slave model. We think, however, that is instructive to confront it with two fine-grain approaches. This makes apparent the consequences of using a particular model.

In the implementation proposed by Randall and Lewis [16] each ant is assigned to a separate processor. It is therefore extremely fine-grained implementation. The client works in a loop in which it receives the pheromone matrix form the server, selects the next node and passes back the choice to the server. This involves massive data exchange. The problem of it is not only just amount of the data, but also frequency of data synchronization. The ants are allowed to modify the pheromone matrix at the same time. In traditional approaches the synchronization of matrix access by different threads is done, e.g., by the standard lock mechanism of Java. The locks are part of the core of the JVM and therefore they are very efficient. The synchronization of different processes if much more time consuming. There are two drawbacks of the solution. The number of ants is limited by the available hardware. In the experiment their number was in the range from 2 to 8.

Table 4 Comparing the efficiency (*Sm*) of message passing and sharing of memory for parallel ACO's [17]

Node number	Number of processors						
	Message passing						
	2	3	4	5	6	7	8
318	0.60	0.48	0.36	0.32	0.27	0.22	0.20
442	0.71	0.54	0.48	0.44	0.38	0.33	0.29
657	0.83	0.65	0.58	0.58	0.54	0.47	0.41
	Shared memory						
318	0.83	0.80	0.77	0.73	0.69	0.66	0.60
442	0.86	0.82	0.81	0.80	0.76	0.75	0.69
657	0.87	0.87	0.85	0.82	0.80	0.77	0.77

All papers advocate more numerous ant colonies and some of them describe even the advantages of overpopulating of the colonies. The second is long time necessary to synchronize numerous modifications of the pheromone matrix. To communicate the processors use the messages.

The solution proposed by Delisle et al. [17] eliminates some on the above deficiencies by relaxing the onerous pheromone matrix update. In this solution each client looks for the route separately updating the pheromones levels locally. Only after completing the route it is passed to the server. The server is responsible for finding the best solution, preforming the global pheromone level update and sending the results to the clients. Each client can host several ants. Best results, both in the terms of speed up and the solution quality were obtained with the value of 40—the maximal number of ants being allocated to a processor. The communication is based on sheared memory model. The Table 4 compares the efficiency measured by S_m of the two approaches.

The results speak clearly for the second approach. In all reported cases the efficiency lowers with increasing number of processors and increases with the number of nodes. The shared memory approach is superior to message passing on every instance. The difference is remarkably high for the lowest number of nodes and eight processers. Note also that

the efficiency raises significantly with the increase of node number.

The communication overhead and need for synchronization of pheromone matrix could severely limit the benefits of parallelization. In the first from above examples the size of transferred data is small, but it is very frequent. The second approach makes the transfer less frequent, but the size of data increases. It is well known that memory sharing is the fastest way of communication between processes. The size of data is not critical factor. The impact of synchronizing read-write access to common memory has more severe impact.

The low level approaches described above have two deficiencies. The necessity of frequent updates diminishes the speed up factor and they require dedicated hardware. The hardware configurations are hardly scalable and are not popular. On the other hand they duplicate or closely mimic the operation of a single colony. Therefore, they can directly exploit the extensive research work on the ACO area.

5.4 Coarse-grain implementation

In a contrast to the fine-grained models the coarse-grain models communicate less frequently. They can be implemented on a number of different types of standard computers. This is the approach taken in this paper. It has two advantages. The structure of the implementation is flexible, could adopt itself to needs or changing environment. Computers with different processing power could cooperate easily. It even enables us to harvest the spare computer power which is available for free on almost all computers. Up to date multicore processors have processing power exceeding that of mainframes from mid-90's. Computers working in a typical network have processor utilization less than 5 % most all the time. The rest is consumed by the idle process of operating system and thus is available for free.

The implementation uses the Sockets mechanism. It keeps permanent connection between two processes until one of them closes it or stops operation. They communicate over network addresses. This ensures a great deal of flexibility as we are not constrained by the physical structure of a computer. The processes could be run on one computer using local host address or over a network. The socket stream-based mode of operation used in the experiments ensures reliable transfer of data on the physical level.

In a previous work the communication between the server and clients used the Java Remote Method Invocation which is a relatively high level mechanism [18]. It offers the developer many advantages. Once the connection has been established the code for handling local and remote objects is almost identical. This gives the developer full compiler support. The implementation and debugging of network programs are not much different from traditional programming. The complexity of organizing the data flow is handled by the compiler.

Table 5 Time necessary to perform basic operations for the RMI and sockets

Operation	RMI network	Sockets	
		Local server	Network server
Initialization if the connection	1.30 s	0.75 s	0.45 s
Passing parameter (one double value)	0.01 s	0.01 s	0.23 s
Passing distance matrix (50 Nodes)	1.60 s	0.03 s	0,26 s

Using the RMI is therefore most beneficial when the interaction pattern is complex.

The RMI enable us to have a parallel implementation with just one remote colony object that resides on a server and hosts the distance and pheromone matrixes. The individual ants could be located on client computers. Once the connection between processes has been established the source code for such an implementations differs not much from its original, sequential version. The RMI could be easily used for fine-grained parallelization approaches.

The RMI approach looks attractive at first, but it is less useful when it comes to an actual implementation. The time necessary for a remote procedure call to fetch a double value is approximately equal to 0.39 ms even when the client and the server reside on the same machine. In contrast fetching the same value from a local object is equal to 0,001785 ms so it is two orders of magnitude faster. Calling a methods with a single double parameter to on network object is much slower—it takes 10 ms, see Table 5. The passing of a pheromone matrix takes almost 2 s.

The RMI mechanism makes it possible to have a straightforward implementation of a fine-grained parallel ant colony, but the performance of such a solution is poor. The access to pheromone matrix is many orders of magnitude slower than for traditional thread-based sequential implementations. For the coarse-grained parallel implementation the communication burden is not prohibitive. Transmitting a whole matrix takes a few seconds, but a colony needs a few minute to process it.

In the case of the coarse-grained parallel implementation of the ACC the data flow pattern is straightforward and it could be implemented without using remote objects. All we need is pure data transfer. That functionality is offered by sockets. The low level socket mechanism provides a connection-oriented service. The protocol used for transmission is the TCP. The Community uses stream-based sockets to establish a connection between two processes. While it is in place, data flow between the processes in continuous streams.

Unlike the RMI the sockets do not provide remote objects and offers only means for data transfer. The programmer

Table 6 Time necessary for the sockets to transfer a given number of floating point numbers (in milliseconds)

Connection type	Number of doubles	Mean time	Std. dev.	Rate
Local	100	11.13	6.48	8.98
	1200	34.99	7.054	34.31
	4900	136.31	12.35	35.95
	19800	511.37	96.89	38.72
Network	100	228.75	12.55	0.44
	1200	261.27	39.37	4.59
	4900	342.85	31.13	14.29
	19800	726.55	330.26	27.25

is solely responsible for ensuring the correctness of data flow that is making sure that both the server and the client expect the same type of data. As the real life communication between a server and a client requires many types of messages being transferred and this may sound prohibitive. Fortunately enough the JVM supports the transfer of any serializable object what makes the task less daunting. Using serialized objects by the socket mechanism does not mean that we have access to objects methods. Only the data are transferred.

The data in the Table 5 show the difference in performance between the RMI an Sockets. The RMI is optimized for parameter passing what is common for method evocation, but are less efficient for passing large amounts of data.

The time that is necessary for initializing connection is shorter than for the RMI, but this is not crucial as it happens only once. The sockets mechanism is optimized for transferring large blocks of data over the network as is clearly visible from the measurements that are shown in the Table 6. The transfer rate steadily increases with the size of data being transmitted and for the largest block there is not much difference between local and network transfer.

In the traditional client—server mode of operation thin clients call a server for performing complex calculations or to obtain data from a centralized database. In the parallel implementation of the ACO the configuration is reversed. The bulk of computations is done by the clients. The obtained results are passed to the server which is responsible for evaluating the individual solutions obtained by the clients, selecting solutions to propagate and sending tasks to servers. The kind of task depends on the granularity level of parallelization.

The server is a multi-threaded process with each thread being attached to a unique client. The number of threads is limited only by the memory available on the server. For a computer with 12 Gb of memory is no limit to their number from the practical point of view. Usually a client colony works on a dedicated computer as a separate process. It is also possible to locate a few of clients running on a single machine.

5.5 Presentation of ant colony community

The ACC consists of a number of colonies controlled by a server. The initial version of the ACC was presented in [18]. The older version of the ACC used the RMI mechanism and its structure was rigid. In an apparent contrast to its predecessor the structure of the ACC presented here uses Sockets and its structure is highly flexible. It could change at runtime.

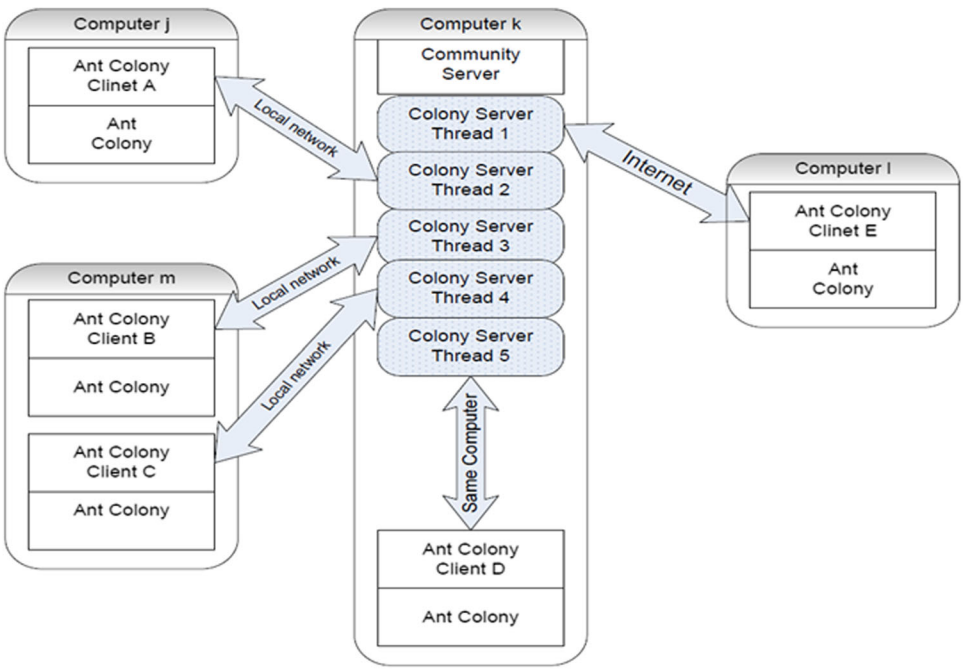

Fig. 4 An example of an ant colony community

Table 7 The message passing between a server and a client colony

Community server operation	Ant colony client operation	Remarks
Stop if work is accomplished		The stopping criteria are described in Sect. 3.3
	← Register client	A separate thread is created to handle the client. Register data include: client identifier and location, current time. They are stored in a server's registry
Initialization data→		Colony operational parameters distance matrix specification
Loop data →	Using data from server the client creates an ant colony and starts its operation	Community best so far solution [distance matrix][pheromone matrix]
	←Found solution	Distance of the best path, sequence of visited nodes, [pheromone] matrix
Result Integration		

The colonies could run a single computer, many computers in a local network or on computers on internet servers. It is possible also to have any combination of this locations. An exemplary structure with four computers and five Ant Colonies is presented on the Fig. 4.

The CS (community server) awaits for calls from ACCs (ant colony clients). The CS after receiving a request from the ACC, registers it in its repository and creates a separate thread for handling data transmission. In the next step the CS passes parameters and data to the appropriate ACC. Depending on the type of Community Server the data can contain distance matrix, pheromone matrix as well as previously found routes. After that, the CS waits for the a solution from any ACC. The ACC in turn after receiving data from the CS a colony creates a local ant colony and feeds it with the data from the server and then starts it operation. An Ant Colony produces solution and passes it to ACC which transmits it further to the Community server.

The descried above structure is relatively simple and at the same time robust and flexible. The complex task of finding solutions is separated from the rest of the implementation. Replacing one type of an ant colony by another is relatively simple. The communication process is initiated by an Ant Colony Client and the Community adopts itself automatically to different processing power or connection transfer rate. The "slower" clients, hosted, e.g., on a remote Internet server are just less frequently assigned tasks by the Colony Server. Even dropping out of a computer from the Community does not disintegrate its operations. The processing just slows down as more tasks are allocated to the remaining clients. The ACCs are separate processes. To obtain reliable measurements of the execution time they were started by a cron-like utility during the tests. A single computer can host many ACCs. We have found that running more than six ACCs on a computer without a SSD drains the recourses of a typical personal computer.

The processing ends when the stopping condition has been met. This happens when the server has already received a specified number of solutions or the predefined processing time has elapsed. The stopping criteria and their substantiation are described in the Sect. 3.3. In either case the server breaks connection to all clients what eventually leads to killing all processes that are executed on the client side.

The Table 7 shows the operation synchronization of the Community Server and Client. The elements enclosed in [ ] are optional. The operations written in bold are performed in a loop, separate for each client.

6 Analysis of ACC operation

To evaluate the performance of the proposed solution a number of experiments were performed.

6.1 Experimental setup

The experiments were run on a network of four computers with codenames from Ca to Cd. All of them have minimum 4 Gb of memory. They were equipped with different processors. The Ca computer was the fastest one. It had Intel i7-4700MQ 2.4 GHz 2.4 Ghz. The Cc with processor Inter Core2Quad D6600 computer was the slowest one. The difference in computing power had an impact on the structure of the communities.

Each colony runs as a separate process. All of the computers were powerful enough to host many of them. The data in the Table 8 show how efficient it could be. The table shows the time used to run various number of an ACO tasks simul-

Table 8 The efficiency of hosting many colonies on a single computer with a SSD drive

Number of colonies	Total time	Time per colony
1	81	81.0
2	83	41.5
3	86	28.7
4	89	22.2
5	100	20.0
6	115	19.8

Table 9 Structure of ant communities

Code	Number of ant colonies	Computer/colony number
A	1	Cb /1
B	3	Ca/3
C	7	Ca/4; Cb/3
D	12	Ca/6; Cb/4; Cd/2

taneously on the computer Cc. Running simultaneously five colonies has increased the processing time by mere 25 % from 81 to 100 seconds. This means that the time per colony has dropped fourfold. For greater number colonies the time per colony increases due to memory swapping and frequent context switching. The data reported in the Table 8 refer to a computer equipped with a SSD drive. For computers with traditional hard disk drives the time per colony starts to increase for smaller number of colonies.

During the tests reported in this table the computer run only ant colony applications. The processor power was fully utilized. This kind of performance could not be sustained if a computer runs any other resource demanding a or an interactive application. This is not a serious limitation as the processor workload of computers in a typical network exceeds single digit values only occasionally. The ant colony application could be started if the processor workload does not exceed a predefined value.

It turned out that assigning many colonies to slower computers, although technically possible has resulted in a notable drop in quality of obtained results. The allocation of colonies to computers was done according to the rule of a thumb and was not the result of any optimization process. The aim of the study was to measure the relationship between the number of Ant Colonies and the achieved route length and not to optimize the Ant Communities structure. The structure of the used communities used in the experiment is presented in the next Table 9. The last column shows the computers and the maximal number of ant colonies that they could host. The Cd computer was used as a server. The Cb computer, ranked in the middle according to processor power was used for reference purposes.

Table 10 Ranking of communities, an example

Community	Len($C_x T_k$)					PRV(C_x, T_k)					RV(C_x)
	1	2	3	4	5	1	2	3	4	5	
C1	2.4	2.1	3.0	2.3	2.1	1	2	0	0	2	5
C2	2.5	1.9	2.2	2.1	2.7	0	3	2	2	1	8
C3	2.1	2.4	1.9	2.2	1.8	3	0	3	1	3	10
C4	2.3	2.2	2.6	1.9	2.8	2	1	1	3	0	7

In the study only one server was used, but it is possible to build a more complex, multilayer structure in which lower layer server pass found solutions to a server located higher in the structure hierarchy.

6.2 Evaluating the efficiency of the TSP ACO

The TSP had started as a recreational puzzle and the solutions were assessed according to one criteria: the length of route. This simplistic approach is not sufficient when we try to evaluate ACO algorithms used for solving the problem. There are several reasons for that:

- As stated in the Sect. 3.1 the exploitation mode of work uses random function to select the next node. The selection of the mode of work is also controlled by a random function. As the result the ACO is non-deterministic by nature. The same algorithm working with identical set of parameters and processing the same distance matrix produces usually different results every time it is activated. We must therefore consider not a single result, but their arithmetic mean, other statistical measures could be also considered.
- A single solution, even a very good one, is not a decisive factor in evaluating a Community. It is very much possible that the next found solution will be not as good.
- Measuring the performance using mean values is not entirely justified due to the dispersion of results obtained in consecutive runs.
- For parallel implementations the computational complexity should be augmented or even replaced by duration of execution.
- It is important not only what solution was found, but also how may iterations we executed. It is true for both parallel and sequential implementations. For the dynamic TSP good quality solutions must be found quickly enough to catch up with the changing distances matrix.

Bearing all this in mind all these above factors we have decided to propose a more complex schema for ranking Ant Colony Communities. The process respects the following principles:

- To evaluate a community we run test it several times using the same distance matrix.
- Instead of using mean of route lengths we used ranks of route lengths.
- Two criteria for stopping the run are used:
  - Equal complexity—the computational complexity measured by the number of node selection operations is the same for all Colonies. The time needed to find a solution could be different and depends on the architecture of the Community.
  - Equal time—the clock time given to all Colonies is the same. The complexity of operation could be different and it is the sum of the complexities of individual Colonies that make up a Community.

Let Len(C_{xk}) denote the length of the best route length found by the Community x in the k-th run.

The process of ranking the communities starts the calculation of PRV($C_x T_k$)—Partial Ranking Value for each test run and each community, see the Formula 13. To rank the Communities we use the PR(C_x) which is sum their ranking values of their tasks, see the Formula 14.

$$\mathrm{PRV}(C_x T_k) = \sum_{i=1}^{N} \left\{ \frac{1 : \text{if} \mathrm{Len}(C_x T_k) < \mathrm{Len}(C_i T_k)}{0 : \text{otherwize}} \right\} \quad (13)$$

where N is the number of communities.

$$\mathrm{RV}(C_x) = \sum_{i=1}^{M} \mathrm{PRV}(C_x T_i) \quad (14)$$

where M is the number of test runs.

The process of ranking of Communities is illustrated by the Table 10.

The equal complexity criteria do not need much justification. It is well established in the computer science. It does not mean that it should be the only one used. Two Communities could have similar ranking values, but could differ substantially in their processing time. In this case the physical structure of a community does not have here any importance.

The equal time criteria are used to select a Community that is most likely to find the best solution in a given period of time. On many occasions the processing time is more important than the difference in route length. For Dynamic TSPs a Community has to find solutions fast enough to adopt to changes in the route matrix. The proposed solution scales very easily and so adding more Colonies could sufficiently seed up the processing.

6.3 The experiment results

During the experiments we ranked Ant Communities using both using the Equal Distances criteria and Equal Time criterion. The Communities used in an experiment are identified by their code (upper case letter) that refers to their structure (see Table 9) and an optional index that differentiates between distinct set of parameters: ant number and iteration number.

6.3.1 Equal complexity criterion

The reference community (code name A) consisted of one colony with standard set of operational parameters with 50 ants and iteration number of 800. All the other communities have preserved the level of computational complexity: the lower values of iteration number were compensated by the increase in the number of colonies and the number of ants in a colony. The actual number of used colonies could be lower that the maximal possible values.

The Table 11 shows the ranking of Communities according the Equal Complexity criterion. The good performance of the Standard Colony comes not as a surprise. Its parameters were carefully chosen after running many experiments and are not manually selected as in the case of the other Communities. Judging the performance of the Standard Community presented in the first row we should not forget that it is much slower than the rest of colonies. The winner, although not a clear one, is the Community B which doubles the number of colonies and keeps relatively large number of iterations. It looks like the iteration number close to 400 is required to achieve acceptable results. Further decreasing of the number of iterations could not be offset by the increase of number of Ants.

Table 11 Ranking of communities according the equal complexity criterion

Comm. code	Colony num.	Ant num.	Iter. #	1	2	3	4	5	6	7	9	8	10	RV
A	1	50	800	5	5	3	5	3	4	0	1	4	1	31
B	2	50	400	1	0	5	3	5	5	4	4	1	5	33
C	2	100	200	0	3	1	0	1	1	3	2	3	0	14
D_1	4	50	200	4	1	2	1	0	2	1	3	5	3	22
D_2	8	50	100	2	2	0	2	4	3	2	5	2	1	23
D_3	8	75	67	3	4	3	4	2	0	5	0	0	4	25

Table 12 Ranking of Communities according the Equal Time criterion

Comm. code	Iter. no.	Ant no.	Colony no.	1	2	3	4	5	6	7	8	9	10	RV
A	800	50	1	0	1	3	0	0	3	0	1	0	0	8
B	350	50	3	1	3	4	3	3	1	4	4	2	4	29
C	350	50	7	5	5	5	5	5	5	5	2	4	4	45
D_1	100	50	12	3	4	1	4	4	3	3	3	4	3	32
D_2	50	100	12	4	2	0	1	2	0	1	4	3	2	19
D_3	100	100	12	2	0	1	2	1	2	2	0	1	1	12

6.3.2 Equal time criterion

In the Equal Time test the structure of the Communities has a great impact on the achieved results. As a reference value we have used the time span necessary for the Cb computer to complete standard run (800 iterations, 50 ants). During the test all colonies were activated at the same time and were allowed to run for the mentioned above time span. After that they were stopped and the best found solution was recorded. Running more colonies on a single computer slows the execution and therefore the number of iterations was lowered from 400 to 350. This was to enable an Ant colony to complete a task within the allowed time period. That percussion measure was needed for the test runs with relative large number of iterations.

The allocation of colonies to computers was done according to the rule of a thumb and was not the result of any optimization process. The aim of the study was to measure the relationship between the number of Ant Colonies and the achieved route length and not to optimize the Ant Communities structure.

The achieved results are presented in the Table 12. The performance looks strikingly different from the previous test. The Standard A community is this time the looser. The difference between the B and C is not a surprise, increasing almost twice the number of Colonies has to result in better solutions. What is, however, worth noting is the performance of the D_1 community. It has a relatively small number of ants, very small amount of iterations and still it the occupies the second position in the ranking. This means that each test takes short time to accomplish. This makes the community a good choice for dynamic TSP.

7 Conclusions and future work

The paper presents initial studies in the performance of Ant Communities. An Ant Community is a coarse-grained parallel implementation of the ACO algorithm. An Ant community has a very flexible structure and the server could coordinate the work of practically any number of individual colonies located upon one or many servers. It uses low level Socket mechanism and is implemented in Java. Both of the features guarantee a fast data transfer. The TSP is a computational demanding process and completion of a typical task could well take almost a minute. Therefore, it allows us to locate many Colonies on one computer or spread them over local or global network and the change the amount of data being transferred between the server and client computers.

The experiments were done with a coarse-grained parallel implementation. The flexible structure of the Ant Community and relatively short transmission overhead allows us to manipulate the granularity level. Further study on that area are necessary. The obtained results show that the Communities could improve the basic, static performance of the TSP ACO. They also provide many clues that they will be much more useful for the dynamic version of the TSP. The tests described in [18], although were obtained using a different approach and implementation technology, also support such a claim.

The ACC was used to coordinate the work of standard Ant Colonies. It is, however, possible to apply the ACC to other related problems, in particular more specific versions of the ACO. Any success on that area would be a proof of the validity of the Ant Colony Communities idea.

References

1. Antosiewicz, M., Koloch, G., Kaminski, B.: Choice of best possible metaheuristic algorithm for the travelling salesman problem with limited computational time: quality, uncertainty and speed. J Theor Appl Comput Sci **7**(1), 46–55 (2013)
2. Psarafits, H.N.: Dynamic Vehicle Routing: Status and Prospects. National technical annals of operations research. University of Athens, Greece (1995)
3. Sieminski, A.: Using ACS for Dynamic Traveling Salesman Problem, New Re-search in Multimedia and Internet Systems. Advances in intelligent systems and computing. Springer, Berlin (2015)

4. Schaefer, R., et al.: Ant Colony Optimization with Immigrants Schemes. In: Schaefer, R., Cotta, C., Kolodziej, J., Rudolph, G. (eds.) PPSN XI, Part II, LNCS 6239, pp. 371–380. Springer, Berlin (2010)
5. Song Y., Qin Y.: Dynamic TSP Optimization Base on Elastic Adjustment. In: IEEE Fifth International Conference on Natural Computation, pp. 205–210 (2009)
6. Dorigo M.,: Optimization, learning and natural algorithms, PhD thesis, Politecnico di Mila-no, Italie (1992)
7. Dorigo M., Stuetzle T.: Ant colony optimization: overview and recent advances, IRIDIA—Technical Report Series, Technical Report No. TR/IRIDIA/2009-013 (2009)
8. Chirico U.: A Java Framework for Ant Colony Systems. Technical report, Siemens Informatica S.p.A (2004)
9. Sieminski, A.: TSP/ACO parameter Optimization. Information Systems Architecture and Technology; System Analysis Approach to the Design, Control and Decision Support, pp. 151–161. Oficyna Wydawnicza Politechniki Wroclawskiej, Wroclaw (2011)
10. Gaertner, D., Clark, K. L.: On optimal parameters for ant colony optimization algorithms. In: IC-AI, pp. 83–89 (2005)
11. Sieminski A.: Ant colony optimization parameter evaluation. In: Multimedia and internet systems: theory and practice. Advances in Intelligent Systems and Computing, ISSN 2194-5357, vol. 183, pp. 143–153
12. Pedemonte, M., Nesmachnow, S., Cancela, H.: A survey on parallel ant colony optimization. Appl. Soft Comput. **11**, 5181–5197 (2011)
13. Delévacq, A., Delisle, P., Gravel, M., Michaël Krajecki, M.: Parallel ant colony optimization on graphics processing units. J. Parallel Distrib. Comput. **73**, 52–61 (2013)
14. G. Amdahl: Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the spring joint computer conference, ACM, New York, USA, pp. 483–485 (1967)
15. Alba, E., Tomassini, M.: Parallelism and evolutionary algorithms. IEEE Trans. Evol. Comput. **6**(5), 443–462 (2002)
16. Randall, M., Lewis, A.: A parallel implementation of ant colony optimization. J. Parallel Distrib. Comput. **62**, 1421–1432 (2002)
17. Delisle, P., Gravel, M., Krajecki, M., Gagne, C., Price, W.: Comparing parallelization of an ACO: message passing vs. shared memory, hybrid metaheuristics. Proc. Lect. Notes Comput. Sci. **2005**(3636), 1–11 (2005)
18. Sieminski, A.: Potentials of Hyper Populated Ant Colonies, 7^{th} Asian Conference, ACIIDS 2015. Lecture Notes in Artificial Intelligence **9011**, 408–417 (2015)
19. Michel, R., Middendorf, M.: An ant system for the shortest common supersequence problem. In: Corne, D., Dorigo, M., Glover, F. (eds.) New ideas in optimization, pp. 51–61. McGraw-Hill Ltd., UK (1999)
20. Gharehchopogh, F.S., Maleki I., Farahmandian M.: New approach for solving dynamic travelling salesman problem with hybrid genetic algorithms and ant colony optimization. Int. J. Comput. Appl. **53**(1), 39–44 (2012)