Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Informatyki, Elektroniki i Telekomunikacji
Instytut Informatyki

Rozprawa doktorska

Ant Colony Optimization with Distributed Pheromones Matrix for Discrete Large-scale Optimization Problems

mgr inż. Mateusz Starzec

Promotor:
prof. dr hab. inż. Aleksander Byrski
Promotor pomocniczy:
dr inż. Kamil Piętak

Kraków 2021

To my beloved Wife,
for her endless love and support.

Abstract

Meta-heuristic optimization algorithms are an important starting point for efficient solving of NP-hard problems. Their universality facilitates application to a variety of optimization problems. However, the *no-free-lunch* theorem is a constant motivation for developing new metaheuristics and refining existing ones.

Ant Colony Optimization algorithms are one of the most popular and efficient optimization methods. They are especially useful for logistics and transport problems, which can be easily represented as a graph. For dozens of years, multiple research papers lead to more and more efficient variants of this algorithm. Some of them are still universal metaheuristics and others are more specialized variations for the specific types of problems. The optimization is based on wide exploration of the solution space. It enables the algorithm to collect internal knowledge, which steers it towards exploitation of the most prospective regions.

The exploration is driven by the group of agents (also known as ants), which iteratively and independently from each other create new feasible solutions. The model of independent agents encourages parallelization of the computations, which enables efficient utilization of modern computational clusters or even HPC environments.

Despite the popularity of parallelization of the Ant Colony Optimization, only a small part of the research considered a *master-slave* model with a single pheromone matrix shared in the whole cluster with numerous computational nodes. The majority of articles focused on sharing cluster resources by multiple ant colonies with independent matrices and limited knowledge exchange between them. The reason might be that the single matrix forces all computational nodes to synchronize after each algorithm's iteration, which leads to reduced scalability of the system, particularly in a large cluster with a big optimization problem to solve.

This extended summary presents a method of distributing and desynchronizing the access to the global knowledge in big computational clusters. The method enables running the computations using large groups of agents in the HPC environments preserving high scalability of the algorithm. As a result, the obtained results are better in comparison to the referential sequential algorithms.

The main part of this work is a design and an implementation of a desynchronized version of the ACO algorithm. A series of experiments proved high scalability of the system and a significant improvement of the convergence and the quality of the solutions. Using a computational cluster enables running a multitude of simultaneously operating agents, which translates into intensified exploration of the solution space and further into better final results of the algorithm. The proposed method does not make any additional assumptions about the problem to be solved, so it preserves the universality of the standard ACO algorithm. As part of work, some prospects of applying the mechanism of desynchronized access to global knowledge to other agent-based optimization algorithms and simulations were also proposed.

The presented thesis is based on 4 published articles that are the main contribution to the state of the art. At the beginning of the document, an extended review of the contributions is presented, depicting the background, pointing out the objectives, describing their implementation and discussing the obtained results. The aforementioned publications are attached at the end of the document and fully referenced in the extended review.

Streszczenie

Metaheurystyczne algorytmy optymalizacyjne stanowią punkt wyjściowy do efektywnego rozwiązywania problemów NP-trudnych. Uniwersalność tych metod pozwala dostosować je do różnorodnych przypadków użycia, jednak teoria *no-free-lunch* jest stałą motywacją do rozwoju istniejących i tworzenia nowych.

Algorytmy mrówkowe są jedną z popularnych i efektywnych metod optymalizacji, w szczególności problemów logistycznych i transportowych, które w bardzo intuicyjny sposób mogą być reprezentowane jako graf. Przez lata, wiele badań doprowadziło do powstania coraz bardziej wydajnych wariantów tej metaheurystyki, a także licznych jej specjalizacji dla węższych grup problemów. Podstawą działania algorytmu jest szeroka eksploracja przestrzeni rozwiązań, która na podstawie wiedzy zebranej w kolejnych iteracjach, zostaje ukierunkowana w sąsiedztwo najbardziej obiecujących możliwości, w celu jego eksploatacji.

Za eksplorację przestrzeni odpowiada grupa agentów, którzy w kolejnych iteracjach, niezależnie od siebie, tworzą nowe propozycje rozwiązań. Agentowy model algorytmu zachęca do zrównoleglenia obliczeń, aby lepiej wykorzystać nowoczesne klastry obliczeniowe lub środowiska HPC.

Pomimo dużego zainteresowania rozproszeniem algorytmu mrówkowego, tylko niewielka część opublikowanych prac skupia się na typowym podejściu *master-slave* z utrzymywaniem globalnej informacji w całym systemie. Zwykle skupiano się na podejściach dzielących zasoby obliczeniowe pomiędzy mniejsze kolonie z ograniczoną wymianą informacji pomiędzy nimi. Wynika to z rozmiaru współdzielonej wiedzy w postaci macierzy feromonów, której synchronizacja po każdej iteracji, w dużym klastrze, skutecznie ogranicza skalowalność algorytmu w klasycznej formie.

Niniejsza rozprawa prezentuje technikę rozproszenia i desynchronizacji dostępu do globalnej wiedzy w dużych klastrach obliczeniowych, która pozwala uruchomić obliczenia z dużymi grupami agentów w środowisku HPC przy zachowaniu wysokiej skalowalności algorytmu, a w efekcie poprawić jakość otrzymywanych rozwiązań w porównaniu do referencyjnych algorytmów sekwencyjnych.

W ramach pracy zaprojektowano oraz zaimplementowano zdesynchronizowaną wersję algorytmu mrówkowego. W serii eksperymentów wykazano wysoką skalowalność systemu oraz wyraźną poprawę jakości i szybkości zbiegania do optymalnych rozwiązań. Wykorzystanie klastra obliczeniowego pozwala na uruchomienie bardzo dużej liczby agentów działających jednocześnie, co przekłada się na zwiększenie eksploracji przestrzeni rozwiązań, a w efekcie na lepsze wyniki końcowe działania algorytmu. Zaproponowana metoda nie przyjmuje dodatkowych założeń dotyczących rozwiązywanego problemu w porównaniu do standardowego algorytmu mrówkowego, co zapewnia jej uniwersalność. Ponadto przedstawiono też możliwości przeniesienia mechanizmu desynchronizacji wiedzy globalnej do innych agentowych algorytmów optymalizacyjnych oraz symulacji.

Niniejsza praca naukowa oparta jest o 4 publikacje stanowiące główny wkład w aktualny stan wiedzy z rozważanej dziedziny. Na jej początku został zamieszczony rozbudowany przegląd wspomnianych artykułów, który opisuje motywację i tło badań, przedstawia cele i ich realizację oraz prezentuje uzyskane rezultaty wraz ze stosowną dyskusją wyników. Przegląd zawiera odniesienia do wspomnianych artykułów, zaś same artykuły zostały umieszczone w całości w drugiej części dokumentu.

Contents

Part I

Extended summary

Chapter 1

Introduction

Meta-heuristic optimization algorithms are a universal and efficient way of solving NP-hard problems. Since these algorithms do not take any problem specific assumptions, they are applicable to a wide variety of use-cases [42]. Moreover, the *no-free-lunch* theory is a motivation for extending the existing ones or even developing new algorithms [40], in order to improve solutions for particular types of problems.

Population based algorithms (e.g., evolutionary ones), like Particle Swarm Optimization [24], Ant Colony Optimization [13] or Social Cognitive Optimization [41], are proven to be effective and easy to implement in parallel platforms [7], which is important to efficiently utilize the modern multi-core processors. However, access to the shared global knowledge may limit the algorithm scalability in large clusters and High Performance Computing (HPC) environments [21]. The scalability is essential in the case of solving real life optimization problems, where solutions quality is as important as the computation time [37].

1.1 Motivation of the research

Ant Colony Optimization is a popular meta-heuristic algorithm for solving discrete optimization problems with a graph-like representation. It is based on a colony of agents (ants) repeatedly creating a multitude of feasible solutions, evaluating them and building a common global knowledge that is further used for subsequent iterations, which should ultimately lead to finding better solutions. Mostly independent work of multiple agents serves as a strong foundation for parallel implementation of the algorithm. However, the fact remains that ACO algorithms incorporate global knowledge in the shape of the pheromone matrix, which the agents need regular access to – both for reading (repeatedly in each step of the solution creation) and for writing (less frequently, however in the

majority of ACO variants at most once per iteration). This property poses a challenge when building distributed implementations of the algorithm with big clusters or even HPC environments in mind.

Successful application of ACO to multiple problems motivated researchers to explore the possibilities of its non-single-core implementation. A lot of studies have been conducted in the fields of both multi-core and GPU-based parallel implementations [8, 31, 43, 44]. Going a step further has lead to adapting the algorithm to a cluster environment. In this area, according to the taxonomy proposed in [27], research papers might be assigned to four categories based on two primary criteria: the number of colonies and the cooperation or lack of it among the parallel parts:

- *Master-slave model* – a single colony without cooperation, where the global information is processed by the master.
- *Cellular model* – a single colony with cooperation, where the colony is divided into small overlapping neighborhoods with their own pheromone matrices.
- *Parallel independent runs model* – multiple ACOs are executed concurrently and independently on several processors using an identical or different set of parameters.
- *Multi-colony model* – the solution space is explored by multiple colonies with their own pheromone matrices, exchanging information periodically.

As the size of the optimization problem increases, so does the dimensionality of the solution space and therefore its exploration becomes more and more difficult. The exploration in ACO algorithm is encouraged by a large size of the colony, so bigger problems require bigger colonies [19]. Thus, one of the objectives when aiming for an effective implementation of ACO is the ability to efficiently manage a large number of agents. This conclusion steers the research towards the master-slave model. However, this area of study has not been very popular and the effects so far, both regarding scalability and obtained optimization results, have not been satisfying [21]. Nevertheless, achieving good scalability of a distributed single colony and thereby being able to significantly increase its population size may become a basis for an efficient ACO implementation with capability of solving large-scale optimization problems.

1.2 Thesis of the dissertation

When implementing parallel and distributed algorithms achieving very good scalability is mostly hindered by sequential parts and any points of synchronization. In the case of creating an environment for execution of an ACO algorithm based on a large distributed colony, a crucial part is to design and implement a mechanism for efficient handling of repetitive access to pheromone matrix – both for reading and writing.

Therefore, the thesis of this work reads as follows:

Introducing a distributed pheromone matrix to the Ant Colony Optimization algorithm will enable a scalable distribution of the computations within a cluster or supercomputer environment, improving the quality of the obtained results in comparison to the referential sequential algorithms.

Taken all together, a distributed pheromone matrix should allow for high scalability of the algorithm, thereby enabling a significant increase of the size of the colony. This in turn should serve as a basis for ensuring proper exploration of the search space. Since the search space increases exponentially with the size of the optimization problem, standard sequential algorithms are not capable of providing proper exploration in reasonable time. Therefore, the proposed modification should be able to achieve comparatively better results.

1.3 Research methodology

In order to prove the thesis, the following research steps were undertaken:

- Designing novel variants of distributed Ant Colony Optimization algorithm using the actor model and implementing them with Scala and Akka.
- Introducing the concept of desynchronization into the developed system in order to remove the need for global synchronization.
- Experimental comparison of scalability of the aforementioned variants in HPC environment (Prometheus supercomputer — a peta-scale (2.4 PFlop) cluster located in the Academic Computer Center Cyfronet AGH in Krakow).
- Identification of the optimal meta-heuristic parameters for a large-sized ant colony.
- Comparison of the proposed algorithms with Max-Min Ant System on TSP instances from TSPLIB[1] in terms of solutions quality.

[1] `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`

1.4 Structure of the document

The dissertation itself consists of a *set of published and thematically related scientific papers*[2], being the main contribution to the state of the art:

- **M. Starzec**, G. Starzec, A. Byrski, W. Turek: *Distributed ant colony optimization based on actor model*. In: Parallel Computing, 2019 vol. 90 art. no. 102573, s. 1–9. (see Section 6).
- **M. Starzec**, G. Starzec, A. Byrski, W. Turek, M. Kisiel-Dorohinicki: *Distributed ant system for difficult transport problems*. In: Journal of Intelligent & Fuzzy Systems, 2019 vol. 37 iss. 6, s. 7347–7356. (see Section 7).
- **M. Starzec**, G. Starzec, A. Byrski, W. Turek, K. Piętak: *Desynchronization in distributed Ant Colony Optimization in HPC environment*. In: Future Generation Computer Systems, 2020 vol. 109, s. 125–133. (see Section 8).
- **M. Starzec**, G. Starzec, M. Paciorek: *Desynchronization of simulation and optimization algorithms in HPC Environment*. In: Computer Science 21(3) 2020: 307-322. (see Section 9).

The document starts with an extended summary of the contributions (Part I), giving an insight into the background, showing the aims and presenting the obtained results with appropriate discussion. At the end (Part II), the aforementioned publications are attached. They are all fully referenced in this extended summary.

The structure of the extended summary is as follows. Chapter 2 outlines the design of the modifications introduced to standard ACO algorithms. Chapter 3 presents the most significant results obtained and conclusions drawn from all conducted experiments. Chapter 4 delineates the potential of applying a similar concept to other algorithms. Chapter 5 summarizes the research carried out for the thesis.

[2]Art. 187. p. 3, Ust. z dn. 20. lipca 2018, Prawo o szkolnictwie wyższym i nauce, Dz. U. 2018 poz. 1668. `http://isap.sejm.gov.pl/isap.nsf/DocDetails.xsp?id=WDU20180001668`

Chapter 2

Desynchronized Ant Colony Optimization

Ant Colony Optimization is one of the algorithms inspired by the collective behavior of natural populations and mechanisms. The algorithm was presented for the first time as a way to solve the Traveling Salesman Problem (TSP) [13]. As a meta-heuristic algorithm, ACO was described a few years later [14].

A meta-heuristic algorithm does not take any assumptions regarding the optimization problem it solves. However, in the case of ACO, it requires the problem to be considered as a graph consisting of a finite set of *components* (vertices) connected by edges with assigned costs. The solution space consists of components paths that respect the posed restrictions and fulfill the requirements defined by the problem. The optimal solution is the path with the minimum cost, which is defined as the function of all of the costs of all connections belonging to the solution.

The algorithm iteratively creates a set of feasible solutions. Each solution is build by an agent, which step-by-step traverses a graph, starting from some initial node, searching for the optimal solution. Selection of each subsequent component in the path is guided by a probabilistic decision rule, which is influenced by a heuristic attractiveness of the component and the pheromone trails left on the edges by the previous generations of agents. In the basic ACO algorithm – Ant System (AS, [16]) – the probability of moving from component i to component j for agent k in iteration t is defined as follows:

$$p_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}(t)]^{\beta}}{\sum_{k \in allowed_k} [\tau_{ik}(t)]^{\alpha} \cdot [\eta_{ik}(t)]^{\beta}} & \text{if } j \in allowed_k \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where $\tau_{ij}(t)$ is the intensity of a pheromone trail on an edge $i \to j$ and $\eta_{ij}(t)$ is the heuristic attractiveness of the edge, which can be defined as the inverse distance between

cities in the case of Traveling Salesman Problem. The parameters that control the relative importance of trail versus visibility are α and β. Finally, $allowed_k$ is a set of possible moves for agent k.

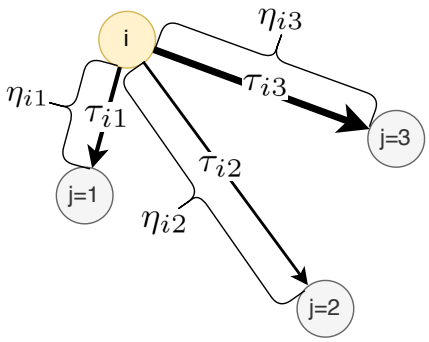


Figure 2.1: Ant's next move selection process.

Fig. 2.1 presents the decision parameters for transition from node i to one of three j candidates. $\tau_{ij}(t)$ – intensity of the pheromone trail – is represented as the width of the arrow, $\eta_{ij}(t)$ depends on the length of the arrow, the inverse of which might be treated as the attractiveness.

In the classic AS version, the algorithm updates the pheromone trails based on the created solutions set after each iteration. The update is performed according to the following formula:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{2.2}$$

where $\rho \in [0, 1)$ is a pheromone evaporation coefficient and m is the number of agents. The pheromone update value for each agent is defined as follows:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & \text{if } k\text{-th agent uses edge } (i,j) \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

where L_k is the cost of the solution of the k-th agent and Q is a configurable constant (often with a value of 1).

The presented algorithm was extended by many modifications improving its performance and efficacy, e.g.:

- Elitist Ant System (EAS, [16]) – only some agents with the best solutions update the pheromone trails. It modifies Eq. 2.2 to:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k'=1}^{m_{best}} \Delta\tau_{ij}^{k'} \tag{2.4}$$

where m_{best} is the number of the selected best solutions and k' is the k'-th best solution.

- Rank-based Ant System (ASRank, [5]) – the pheromone trail update is weighted by a rank in the solutions ranking. It modifies Eq. 2.2 to:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k'=1}^{m_{best}} (\Delta\tau_{ij}^{k'} \cdot (1 - \frac{k'-1}{m_{best}})) \tag{2.5}$$

  where m_{best} is the number of the selected best solutions, k' is the k'-th best solution.

- Max-Min Ant System (MMAS, [36]) – introduced bounds of minimum (τ_{min}) and maximum (τ_{max}) value of the pheromone trail. It modifies Eq. 2.2 to:

$$\tau_{ij}(t+1) = max(\tau_{min}, min(\tau_{max}, (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}^{best})) \tag{2.6}$$

  where $\Delta\tau_{ij}^{best}$ is the pheromone update value incidental to the best solution from iteration.

- Ant Colony System (ACS, [15]) – introduced *pseudo-random-proportional rule* that modifies ant's choice of next move: if the value of a random variable q uniformly distributed over $[0, 1]$ is not greater than a q_0 constant, then the component that maximizes the product $\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}$ is chosen from the feasible components (exploitation); otherwise, Eq 2.1 is used (biased exploration). Additionally, right after choosing the next edge, its pheromone is updated according to the following equation (similar to Eq. 2.2):

$$\tau_{ij} = (1-\phi)\tau_{ij}(t) + \phi\tau_0 \tag{2.7}$$

  where $\phi \in (0, 1]$ is the local pheromone decay coefficient, and τ_0 is the initial value of the pheromone.

This chapter describes the most common ways of parallelization and distribution of Ant Colony Optimization algorithms. Then, it introduces the idea of a distributed pheromone matrix and desynchronized access to it. The proposed modifications aim to enable high scalability of the algorithm, that will lead to efficient utilization of big clusters and HPC environments. In order to simplify the system architecture and focus on the inter-process communication, it will be described using the actor model. The high scalability will also require additional consideration regarding efficient utilization of the numerous ant colony by proper settings of the meta parameters.

2.1 Parallel and distributed ACO

The independent agents, building feasible solutions in each iteration, make the algorithm interesting to be considered in parallel and distributed environments. However, the need of constant access to the global knowledge, represented as pheromone matrix, makes the distribution of agents nontrivial. There are a few widely used strategies of using computational clusters to run distributed ACO algorithms, each having some pros and cons [27].

The simplest way to utilize a huge cluster is to run multiple independent colonies [2, 3, 28] and collect the results from them (see Fig. 2.2). It makes the system almost infinitely scalable, since there is no communication among colonies during the calculations. It only requires distributing the initialization data and collecting the results at the end. The model of independent runs may also explore meta-parameters space in order to fit them to a specific problem instance. It may be also used in combination with any other model as a framework running the other models in independents runs. On the other hand, the lack of communication among colonies may lead to a situation, where most or even all colonies are stuck in local optima and are unable to explore the solution space properly. The model can be scaled easily, but it obtains similar results to the sequential algorithms and it is outperformed by other models which employ communication among colonies [27].

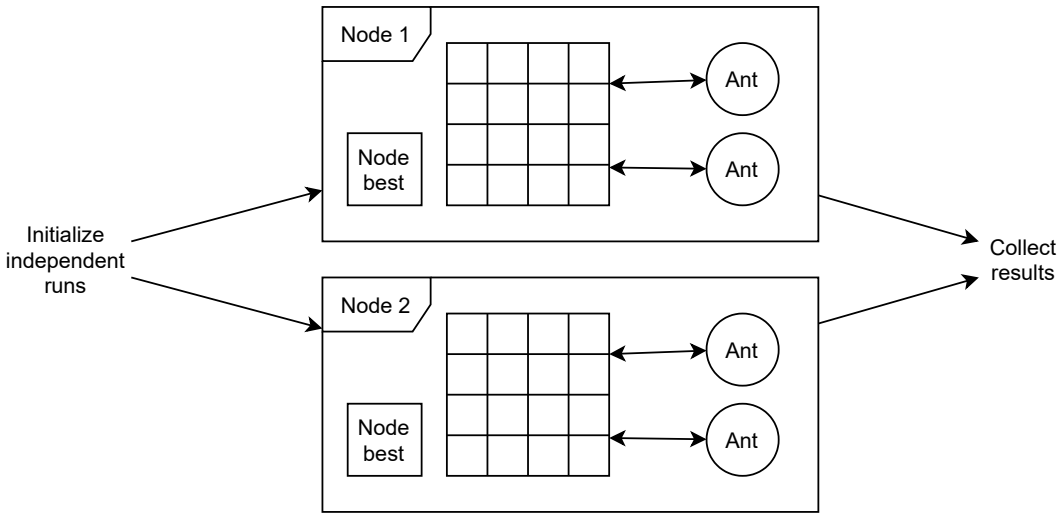


Figure 2.2: The parallel independent runs model.

In order to avoid the aforementioned problem caused by the lack of communication, one may introduce sparse communication in order to exchange some knowledge among colonies [17, 23, 29, 39] (see Fig. 2.3). It might be the best solution found in each colony broadcast to all other colonies, however, there is a lot of research regarding the best strategies of data exchange. The knowledge exchange may help the colonies avoid stagnation in local minima without noticeable impact on the system scalability. However, the more data is exchanged,

the better the results are but it comes at a cost of worse scalability. In huge clusters, the data exchange may be organized in some kind of neighborhood communication instead of broadcasts in order to avoid scalability issues.

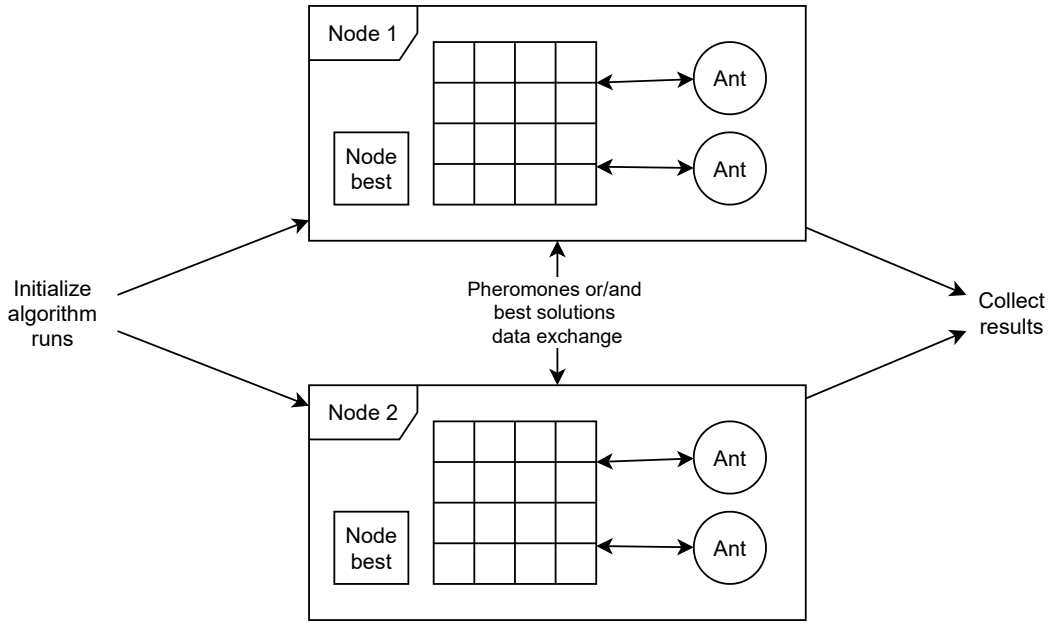


Figure 2.3: The multi-colony model.

Another step to exchange more data between computation nodes while keeping the scalability high is to organize the nodes into overlapping neighborhoods. In the *cellular* model, each node has its own pheromone matrix in order to provide fast data reads, however, the ants solutions are broadcast to all neighbor matrices in order to diffuse the knowledge across the whole cluster as a part of pheromone trails updates [26] (see Fig. 2.4). The model was not deeply explored so far, nevertheless, it provided promising results.

It is also possible to share a single pheromone matrix in the whole system in order to maximize the knowledge exchange among ants (see. Fig. 2.5). On the other hand, it forces a single node to keep the data in memory, allow all ants in the cluster to read it (which introduces a lot of inter-node communication) and collect all solutions from the cluster to handle pheromones update procedure. That greatly reduces system scalability, since the master node is overloaded, accessing data through network is slow and the other nodes need to synchronize after each iteration and wait for pheromones update. However, the *master-slave* model is quite popular on small clusters, because it is easy to implement and provides good results [9, 12, 18, 25].

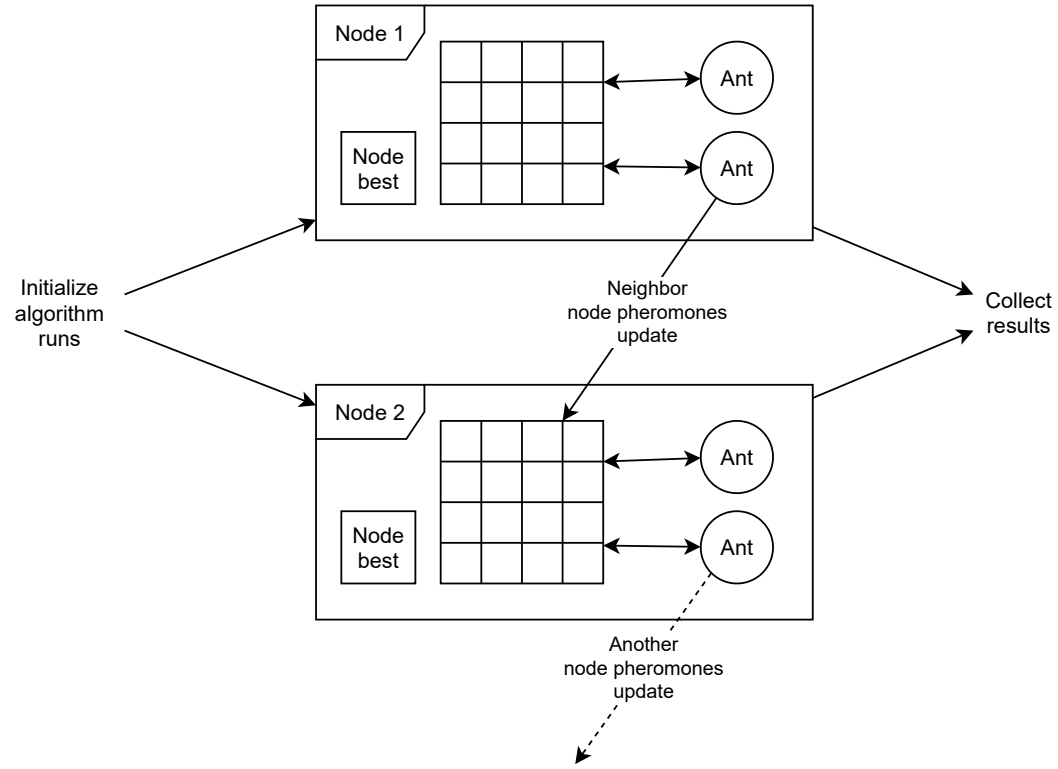


Figure 2.4: The cellular model.

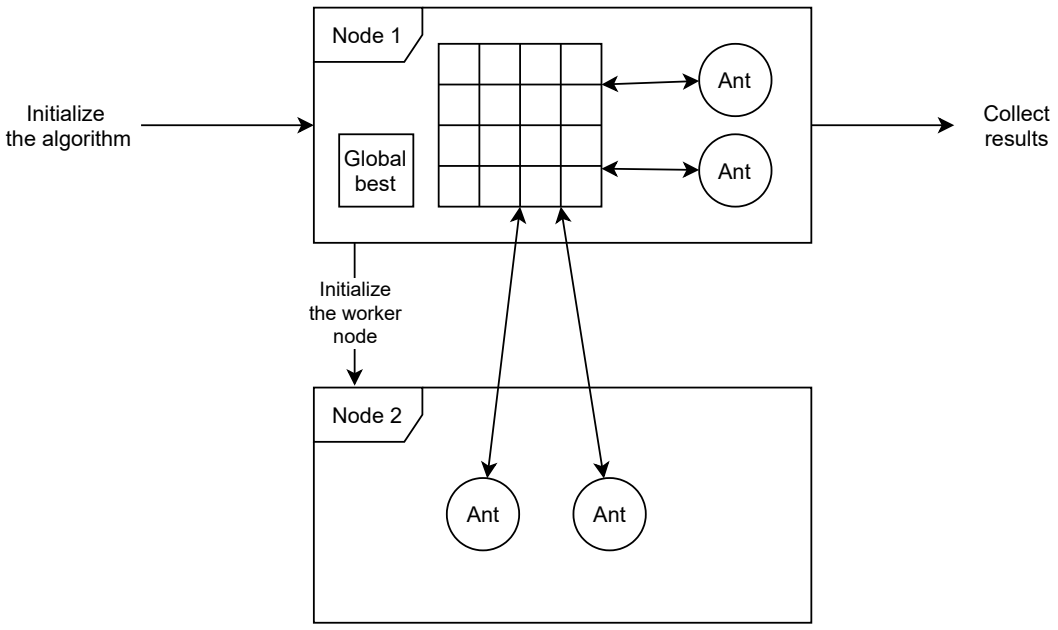


Figure 2.5: The master-slave model.

The presented models of parallel and distributed version of ACO algorithm try to reach a compromise between good scalability and effective joint search of the solution space. The main problem when distributing ACO is choosing a way to deal with access to common knowledge in the form of the pheromone matrix. Some models head towards high scalability giving up common knowledge by splitting it into multiple, independent pieces. In this work, a novel approach to distributing ACO is presented. It utilizes a single distributed pheromone matrix for the whole ants colony, providing global common knowledge. Still, its distribution across nodes and employing a desynchronization mechanism gives a chance for preserving good scalability.

2.2 Ant Colony Optimization based on Actor Model

The Ant Colony Optimization algorithm is based on agents, which create feasible solutions independently, synchronizing only at the end of each iteration. The actor model is a way to efficiently distribute computational load across multi-core processors or even computational clusters. The model was proposed in 1973 by Carl Hewitt [1, 20]. It introduces the message passing concurrency model and defines the concurrent processes as the basic units of parallel computations, which are internally sequential and process the incoming messages one-by-one. The mechanism of asynchronous messages allows the actors to communicate and as a result they can change their state, send other messages or create new actors.

This section focuses on leveraging the actor model in order to implement a scalable and reliable system by using a novel and easy-to-apply technology (like the actor model delivered by Scala and Akka [4]) ([32] and Section 6). The model is supposed to ease understanding the parallelism of the system [6, 22, 38] and focusing on inter-node and inter-process communication, since these are the major factors reducing system scalability.

The system elements are modelled by several types of actors:

- *Computation Master* – a singleton deployed on the master node, sets up the *Computation Local* actors on every node and manages the whole computation process.
- *Computation Local* – one actor per node, manages all other actors on its node in order to handle cluster statistics and failures.
- *Pheromone Manager* – one actor per node, provides pheromone trails data to *Ant*s and handle pheromones updates; internally, it communicates with the managers on other nodes to collect data from the distributed pheromone matrix and may additionally cache its remote parts.

- *Problem Description Manager* – one actor per node, provides possible moves to *Ant*s, based on the problem restrictions.
- *Ant Manager* – one actor per node, manages *Ant*s life cycle.
- *Ant* – multiple actors distributed across all nodes, iteratively constructs feasible solutions based on the data provided by local *Problem Description Manager* and *Pheromone Manager*.

Both *Pheromone Manager*s and *Problem Description Manager*s might internally create other actors in order to handle requests in parallel to improve utilization of multi-core processors.

The model's behaviour does not diverge from the standard Ant System, since it still forces the synchronization of all nodes at the end of each iteration in order to update the pheromone matrix. Fig. 2.6 makes it easier to spot the remote communication that may affect the system scalability in the case of big clusters. The actors are organized in a way that allows only *Pheromone Manager*s to communicate with actors on other nodes, except the system initialization and statistics collection. It blocks *Ant* processes when they ask for pheromone values from the remote parts of the matrix until the data is loaded, but caching enables reuse of the remote values for multiple requests from different agents in the same iteration.

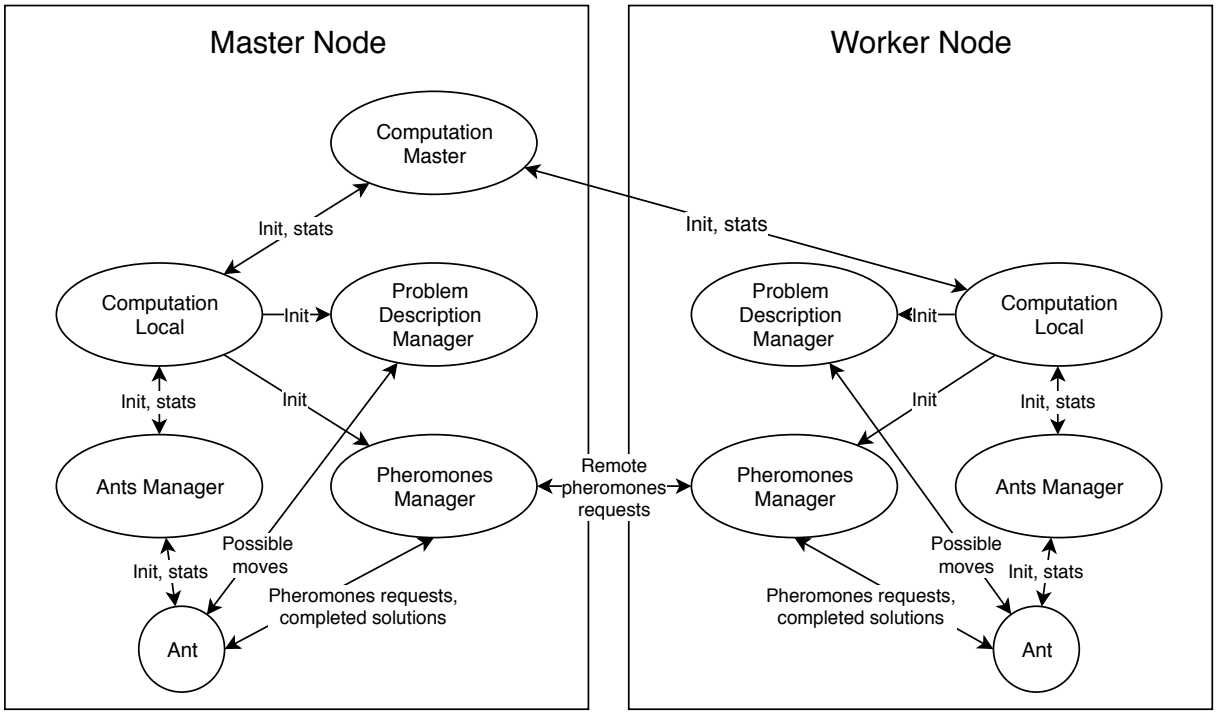


Figure 2.6: Relationships and data passed between actors.

2.3 Distributed pheromone matrix

The model of the system, presented in the previous section, assumes that the information from the pheromone matrix might be distributed across the nodes. This section goes into detail about this topic as it is crucial for the system scalability ([32] and Section 6, [33] and Section 7).

The standard *master-slave* architecture enables parallelization of agents constructing feasible solutions, however, it is limited by performance of the master node, which has to provide pheromone trails data to all other nodes. Fig. 2.7 presents the idea of the distributed pheromone matrix. It splits the matrix into parts and distributes them evenly between the nodes. Therefore, the single point providing all the data is eliminated in favour of decentralized reads and updates at the end of each iteration.

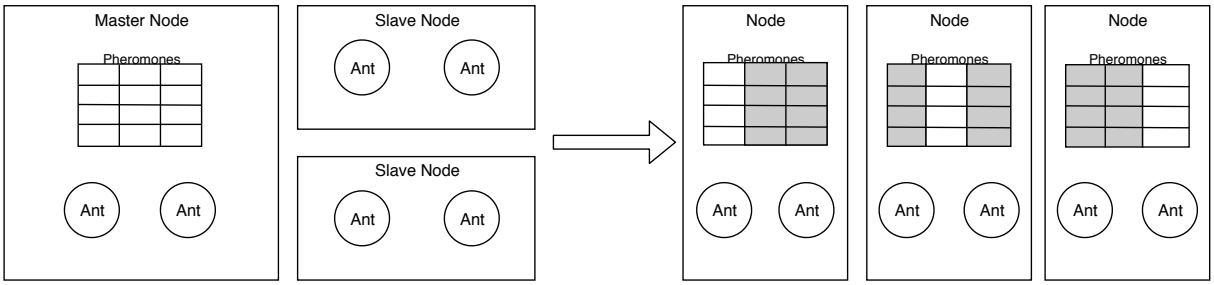


Figure 2.7: Migration from the master-slave model to the distributed pheromone matrix.

The columns of the matrix assigned to node n (enumerated from 1) are described by Eq. 2.9 and Eq. 2.10.

$$Cols = ceil(\frac{MatrixSize}{NodesCount}) \tag{2.8}$$

$$FirstCol(n) = Cols * (n - 1) \tag{2.9}$$

$$LastCol(n) = min(Cols * (n - 1), MatrixSize - 1) \tag{2.10}$$

Since there are multiple agents constructing feasible solutions on each node, the values read from the remote nodes should be cached and reused until the end of iteration, if the operational memory can fit them. Otherwise the cache may use *Least Recently Used* strategy so as to reduce the memory usage. The remote parts are loaded on-demand in order to disperse network traffic created by the requests and reduce duration of inter-iterations pause. It also avoids loading unnecessary parts of the matrix in the case of problems, which do not require all graph nodes to be used in a feasible solution.

The number of remote pheromone requests outgoing from a single node in each iteration depends on the cache size and may vary as described by Eq. 2.11, where $Cols$ is the number of matrix columns on each node (Eq. 2.8) and m_n is the ants count on the node. The bounds may be lower if the problem solution does not require all components to be used.

$$(Nodes - 1) * Cols <= Requests <= (Nodes - 1) * Cols * m_n \tag{2.11}$$

The updates of distributed pheromone matrix require broadcasting the solutions, which will be strengthened in the trails. However, the time required for transmission is compensated by the utilization of the whole cluster's computational power during the update in the case of big matrices. The size of the broadcast messages can be reduced by sending a map from used edges to a sequence of the solutions costs instead of the whole solution paths, since the used edges may repeat between the solutions. It also allows to limit the information sent to each manager only to his part of the matrix.

2.4 Desynchronization of pheromone matrix

Even with the pheromone matrix distributed across the whole cluster, the behaviour of the algorithm still requires data synchronization at the end of each iteration. It forces all *Ant*s to finish their solutions, then stop and wait for the others and for the exchange of the solutions data between the nodes to complete. The bigger the problem size and the nodes count are, the longer the pause between iterations becomes, which greatly reduces the system scalability.

The concept of desynchronization may be applied in order to reduce the pause wasting computational resources ([34] and Section 8). When an agent finishes a feasible solution, it should start constructing another one without waiting for anything. It might cause the usage of the pheromone values from the previous iteration until the distributed updates finish. Fig. 2.8 presents an example of agents communication in the case of two nodes, each with a single ant. It shows that the *Ant*s are able to load last known pheromone trails whenever they need and the updates are applied independently as soon as they are available.

This solution actually simplifies the internal logic of the *Pheromone Matrix Manager* (compare Alg. 1 and Alg. 2), because when he receives a request for the pheromone trails values it can simply return the current value from the matrix. In the standard version (see Alg. 1), the request has to contain the *Ant*'s current iteration number and the manager has to queue requests which refer to the next iteration values, before the matrix

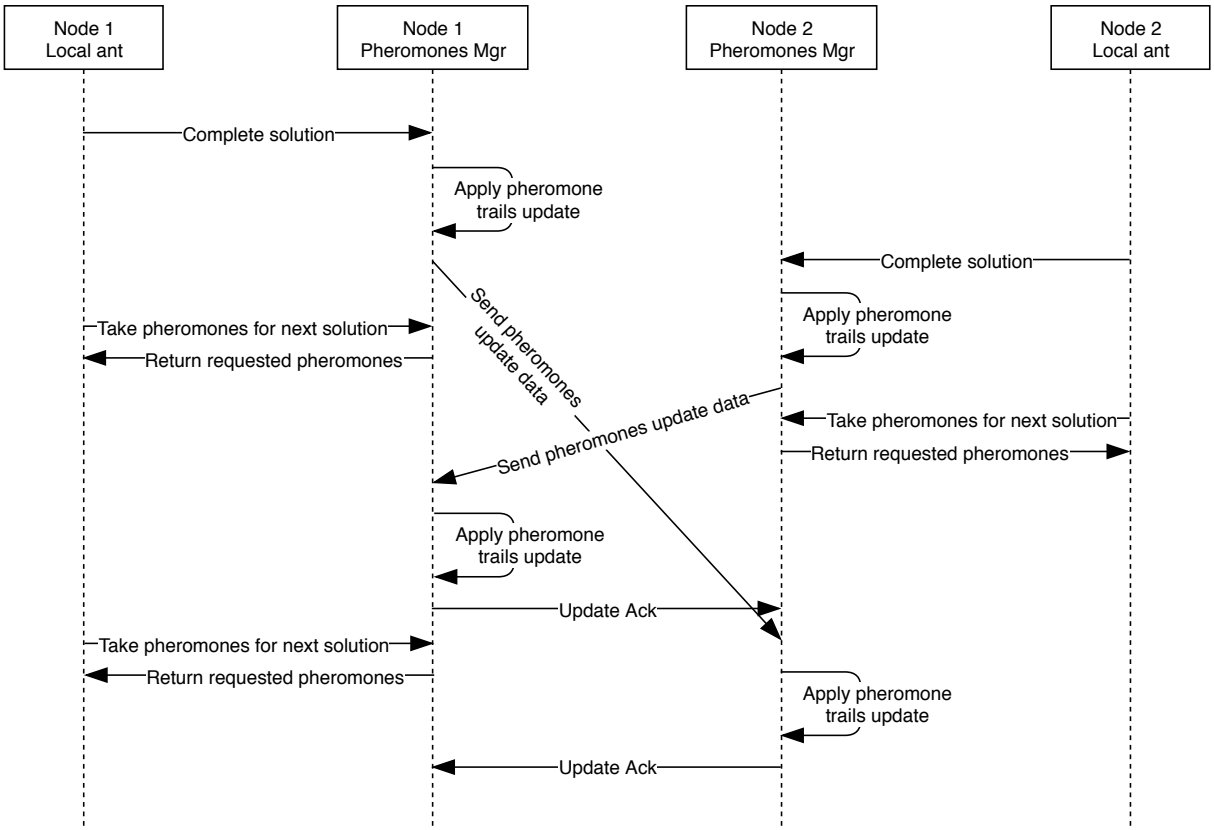


Figure 2.8: Desynchronized pheromone matrix updates – messages sequence example.

update is finished. Loosing this restriction allows the system to maximize computation resources utilization at the cost of a minor inconsistency in the algorithm behaviour when the updates of the pheromone trails are delayed. On the other hand, the Ant Colony Optimization, similarly to other optimization algorithms, is randomized by design and the results are not expected to be deterministic, so the update delays may be considered as another random factor in the algorithm.

Since there is no single event starting the pheromone matrix update, the mechanism of caching remote values also has to be modified in the desynchronized version. When a *Pheromone Matrix Manager* collects a batch of feasible solutions from the *Ant*s on his node, he broadcasts it to the other managers and starts updating his part of the matrix. At the same time, he clears the whole local cache and the other managers clear only the values received from the broadcasting manager. The number of solutions contained in a single batch might be configurable, but in the experiments it was usually equal to the number of *Ant*s on each node.

Algorithm 1 Ant request handling – standard

```
1: procedure HandleAntRequest(iteration)
2:     if isMatrixUpdated(iteration) then SendResponse()
3:     else AddRequestToQueue(iteration)
4: procedure HandleLocalSolution
5:     AddSolutionToLocalBatch()
6:     if batchSize == expectedBatchSize then
7:         SendUpdatesToRemoteManagers()
8:         HandleUpdateBatch(localBatch)
9:         ClearLocalBatch()
10: procedure HandleUpdateBatch(batch)
11:     AddBatchToQueue()
12:     if batchesCount == nodesCount then
13:         UpdateMatrix()
14:         ClearBatchesQueue()
15: procedure UpdateMatrix(iteration)
16:     ApplyUpdateAndEvaporation()
17:     HandleAllQueuedRequests(iteration)
18:     ClearRemotePheromonesCache()
```

Algorithm 2 Ant request handling – desynchronized

```
1: procedure HandleAntRequest
2:     SendResponse()
3:     if requested value is remote and invalidated then
4:         SendCurrentValueRequest()
5: procedure HandleLocalSolution
6:     AddSolutionToLocalBatch()
7:     if batchSize == expectedBatchSize then
8:         SendUpdatesToRemoteManagers()
9:         HandleUpdateBatch(localBatch)
10:        ClearLocalBatch()
11:        InvalidateRemotePheromonesCache()
12: procedure HandleUpdateBatch(batch)
13:     ApplyUpdateAndEvaporation()
14:     InvalidateSenderValuesInPheromonesCache()
```

The proposed changes remove the global iteration counter and introduce some delays, affecting pheromone values used in each iteration counted from a single *Ant* perspective. Eq. 2.12 describes the possible options on each pheromone value read.

$$\tau_{ij}(t) = \begin{cases} \tau_{ij}(0) + \sum \Delta\tau_{ij} + \sum \Delta\rho & \text{if edge } (i,j) \text{ is in the local matrix part} \\ Cache_{ij}(s), s < t & \text{if edge } (i,j) \text{ is in a remote matrix part} \end{cases} \tag{2.12}$$

where t is the time of receiving request for the pheromone value of the ij edge, $\Delta\tau_{ij}$ is the pheromone value update received before t and $\Delta\rho$ is a pheromone evaporation update after each local batch pheromones update, $Cache_{ij}$ is a remote value cached at time s.

2.5 Efficient use of a huge ants colony

This section outlines measures introduced to the algorithm in order to maximize the profit gained from significantly increasing the size of the colony ([33] and Section 7). In order to efficiently utilize hundreds of nodes with multi-core processors, the colony should have at least as many *Ant* actors as the number of cores in the cluster. The standard implementations of Ant Colony Optimization algorithms usually run dozens of agents, hence, it is also important to take advantage of numerosity of created feasible solutions.

The standard Ant System allows all feasible solutions to leave pheromone trails [16], however, the improved variants (like Elitist AS, MMAS) suggest using only the best solutions subset (from a current iteration or the best found so far) as it improves the final results [16, 36]. In the case of a huge colony, using only a few best solutions might be a waste of the collected knowledge, therefore, the system should be configurable in this part and allow more solutions update the trails.

The configuration of the algorithm, used in the experiments in order to test different scenarios, includes:

- N – a number of ants in the system.
- ρ – a pheromone evaporation coefficient.
- Q – a pheromone update unit.
- α – an importance of the pheromones.

- β – an importance of the desirability.
- IBU – iteration best updates; a number of the best ants from the current iteration updating the pheromone values after each iteration. Their update is not weighted by the ranking position.
- GBU – global best updates; a multiplier for the pheromones update based on the global best solution after each iteration.
- $\tau(0)$ – an initial amount of the pheromones on every edge. Usually it is equal to the τ_{Max} value.
- τ_{Max} – a maximum value of the pheromone on an edge.
- τ_{Min} – a minimum value of the pheromone on an edge.

Eq. 2.3 uses the Q parameter and the solution cost to calculate pheromone update value. Since the solution cost might be any real value, the value of Q is tightly connected with the problem. To avoid this inconvenience, the algorithm normalizes the solution using the value of the current global best solution. With a minimum and maximum limit of the pheromone values, it allows to prepare sensible configuration values without a need to adjust them to given problem instance.

The order of magnitude of $(IBU + GBU)\frac{Q}{Fitness}$ and $\rho\tau_{Max}$ values should be similar, since it allows to keep the trail of the current best solutions close to τ_{Max}.

Chapter 3

Overview of experimental results

The experiments focus on two main areas of study. First of all, the scalability of the created implementations should be investigated, since this property, according to the thesis, will allow to use larger colonies and obtain better results in comparison to standard sequential ACO algorithms. And that is the second field of experiments, where the objective is to prove the improved efficacy of the system manifesting itself in better final optimization results.

This chapter summarizes the most important experimental results and conclusions from the publications. Some selected graphs from the publications are included below for illustrative purposes.

The experiments were conducted using the Prometheus[1] supercomputer — a peta-scale (2.4 PFlop) cluster located in the Academic Computer Center Cyfronet AGH in Krakow, Poland. As of November 2020, Prometheus was ranked 324th in the TOP500 fastest supercomputer list. Prometheus is a cluster of HP Apollo 8000 nodes, each with 24 Xeon E5-2680v3 CPUs working at a 2.5GHz frequency and connected via an InfiniBand FDR network.

3.1 Scalability testing

Despite the multitude of parallel Ant Colony Optimization algorithms, there have not been many implementations of a single ant colony in a cluster environment. These few found that actually took up this problem did not manage to prove spectacular scalability. For example, Ilie and Bădică et al. [21] presented an agent-based approach to ACO modeling and they reported very good scalability up to 7 nodes on the *gr666* problem from TSPLIB.

[1]Thanks to PlGrid project support `http://www.plgrid.pl`.

The results obtained were comparable to the standard Ant System in terms of the final solutions quality. Unfortunately, there were no results for large-scale TSP instances nor larger clusters.

In this work, the first step towards achieving high scalability was the distribution of the pheromone matrix. With this innovation, it was possible to spread and parallelize the sequential and centralized part of the algorithm – updating the pheromone matrix at the end of each iteration (see Section 2.3). When testing the scalability of the solution it was applied to four TSPLIP problems with increasing size (666, 1432, 2392, and 3038 components) ([32] and Section 6). The size of the colony was constant and equal to 10,000. The agents were evenly distributed across an increasing number of nodes (from 1 up to 30). As a result, the speedup was calculated as the duration of a single iteration on one node divided by the duration of a single iteration on multiple nodes – see Fig. 3.1. Each configuration was repeated 5 times.

Figure 3.1: The system scalability with 10k ants on a different number of nodes.

From the chart it is clear that the scalability of the system depends on the size of the problem. All problems scale well up to 5 nodes, but the smaller the problem is, the faster its speedup curve starts to flatten, i.e. only bigger problems can still take advantage of additional nodes when there are already more than 15.

Another possibility of investigating the scalability of the system in case of a large, distributed colony of agents is testing its weak scalability. For this purpose, again two problems from TSPLIB were used and run on a varying number of nodes (from 1 up to 20) but this time the number of agents on each node was constant so effectively the size

of a colony varied from 500 to 10,000 ([32] and Section 6). Again, each configuration was repeated 5 times, and the standard deviation was negligible and is practically invisible on the chart.

It should be noted that this is a test of weak scalability in the context of the number of agents, but not entirely in the context of pheromone matrix distribution. Although the number of ants increases linearly with each additional node, the pheromone matrix total size stays the same and the matrix is divided into smaller pieces.

Fig. 3.2 presents how the duration of a single iteration changed with the size of the colony. Ideally the duration should not increase at all. Here good weak scalability was observed that nonetheless motivated further work on this topic, especially that this experiment also exhibited the deterioration of scalability with the number of nodes reaching 20.

Figure 3.2: Single iteration duration with different ant numbers distributed as groups of 500 ants per node.

In order to further improve the scalability of the system, the desynchronization of the pheromone matrix was introduced – the boundaries between consecutive iterations were blurred as the reading and the updating of the pheromone values were decoupled as described in Chapter 2.4.

Since the experiments use the number of iterations for measuring the scalability of the system, the current iteration number $EstIt$ has to be estimated based on the total number of solutions built so far S and the number of all agents K: $EstIt = \frac{S}{K}$.

The scalability and performance of the distributed and the desynchronized versions of the algorithm were compared by testing weak scalability ([34] and Section 8). Tables 3.1 and 3.2 present the average duration of a single iteration when running both versions on varying number of nodes, with 25 agents operating on each node. For the distributed version the maximum number of nodes was 100, since for more nodes it was not able to reach sensible iteration time. The desynchronized version was tested on up to 400 nodes. Every test case was repeated 10 times.

Nodes	1	2	5	10	20	50	100	200	300	400
Distributed ACO	16.77	15.74	12.65	12.24	11.63	13.9	39.93	-	-	-
Desynchronized ACO	16.75	17.72	15.55	15.58	15.94	17.93	19.34	20.72	21.35	22.00

Table 3.1: Iteration time (s) on pr2392 from TSPLIB.

Nodes	1	2	5	10	20	50	100	200	300	400
Distributed ACO	-	2.13	6.22	10.33	28.98	43.31	62.04	-	-	-
Desynchronized ACO	-	1.89	5.38	10.75	21.01	46.71	86.64	161.67	235.41	304.58

Table 3.2: Speedup on pr2392 from TSPLIB.

When running on smaller clusters, the distributed version is faster but on 100 nodes it is significantly outperformed by the desynchronized version. The speedup of the distributed version is also better, but the desynchronized version maintains the efficiency over 75% up to 400 nodes. Speedup and efficiency were calculated as follows:

$$Speedup_n = \frac{t_n}{t_1} * n \tag{3.1}$$

$$Efficiency_n = \frac{Speedup_n}{n} \cdot 100\% \tag{3.2}$$

where n is a number of nodes and t_n is an average iteration duration on n nodes.

The system shows super-linear scalability below 50 nodes. It is a result of the distributed pheromone matrix update, which allows to utilize the whole cluster for the update process.

The presented experimental results confirm the positive influence of the distributed pheromone matrix on the system scalability. The desynchronized access to the matrix improved it even further, enabling the system to efficiently utilize HPC environments and run huge ant colonies.

3.2 Quality of solutions

Although scalability is an important property of the distributed system, what is even more important is its efficacy, i.e., its ability to successfully solve optimization problems and obtain satisfying results in limited time if necessary. Therefore, the second part of the experiments focuses on the achieved optimization results.

First of all, an empirical proof of equivalence of two Ant System versions – a simple sequential implementation and the distributed one based on the actor model – will be presented ([32] and Section 6). The main objective is to demonstrate that the distributed version is compliant with the original ACO algorithm. For this experiment a small problem from TSPLIB was solved 20 times by both implementations using 500 ants (in the distributed version the colony was split in half on two nodes). Fig. 3.3 presents the cost of the best solution found so far by both implementations with a standard deviation across repetitions. As one can see, the first phase of the algorithm (the exploration phase) shows a higher variation in the results, which diminishes in the second phase of the algorithm (the exploitation phase). However, the key observation here is that both implementations exhibit extremely similar behavior as the chart lines practically overlay each other.

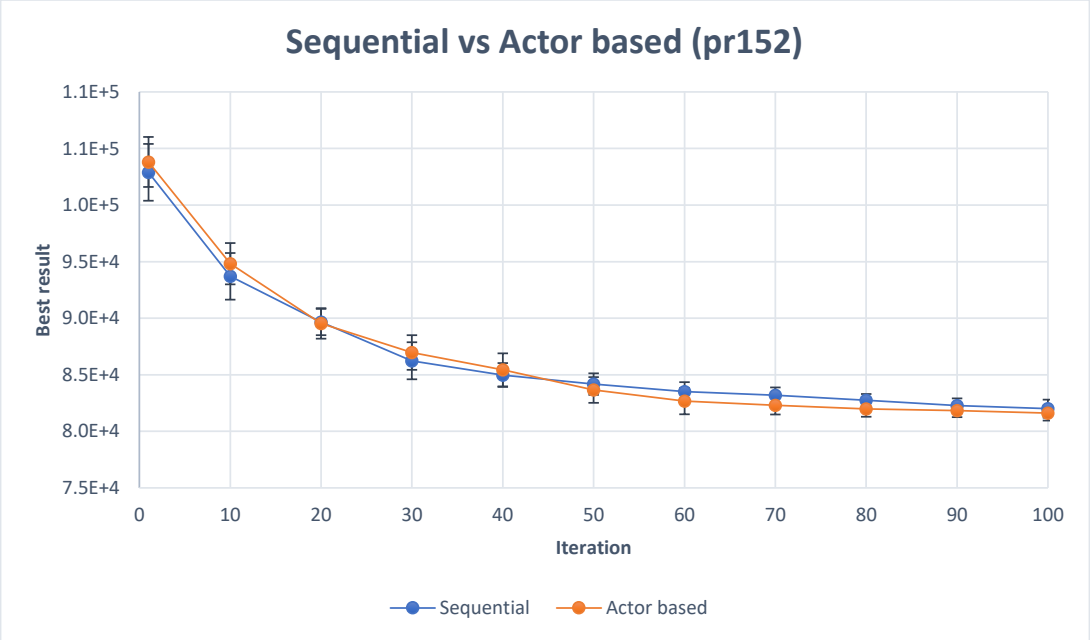


Figure 3.3: The cost of the best solution found so far for subsequent iterations by the sequential and the distributed implementation of ACO.

Distributing and desynchronizing the ACO algorithm allowed for enlarging the colony. Typically, MMAS algorithm uses only a global best and/or iteration best solution. Using such approach means that most results of computational work is dropped and the profit from running a large colony is limited. In order to better leverage enhanced exploration, the proposed algorithm updates the pheromone matrix based on multiple global and iteration best solutions. The experimental results showed that increasing total number of agents and properly adjusted percentage of ants leaving the pheromone trails (around $10\% - 20\%$) improve the results quality ([33] and Section 7).

Figures 3.4 and 3.5 present the cost of the global solution versus iteration number for the distributed and the desynchronized version of the ACO algorithm respectively; for varying size of the colony ([33] and Section 7, [34] and Section 8). The problem solved was a pr2392 from TSPLIB with 10 repetitions for each configuration. Note that in the first chart we can see the first 100 iterations whereas in the second one the are 200 iterations. The charts prove that the higher the number of agents is, the better the final result is. Since the desynchronized version scales better, it was possible to run a colony of 10k agents, and it achieved a better result than the largest distributed colony (2.5k agents). Moreover, the higher number of agents enables reaching the same solution quality with less iterations, which might be important in case of optimization with limited computation time.

In case of the desynchronized version of algorithm, it falls into stagnation after 140 iterations and is unable to improve the results any further. This problem could be evaded by adjusting meta-heuristic parameters, however, the experiments were conducted with analogous values in order to directly demonstrate the influence of the desynchronization and the number of agents.

However, as shown for a smaller problem, the profit gained from increased exploration of a large colony has its limits ([33] and Section 7). After reaching a saturation point, adding more agents does not further improve obtained results. This means that proposed algorithm might be especially valuable for large-scale optimization problems.

At the same time, it has been shown that using weak scalability enables achieving better results within the same time (but using more resources) ([34] and Section 8). In the experiment, the number of agents on a single node was constant. As already mentioned, adding more nodes (and thereby increasing the size of the colony) resulted in better final results and results in consecutive iterations. Even more interestingly, it also lead to better results obtained within a specific time. The profit from expended exploration of a larger colony outweighed the fact that using additional nodes increased the duration of a single iteration.

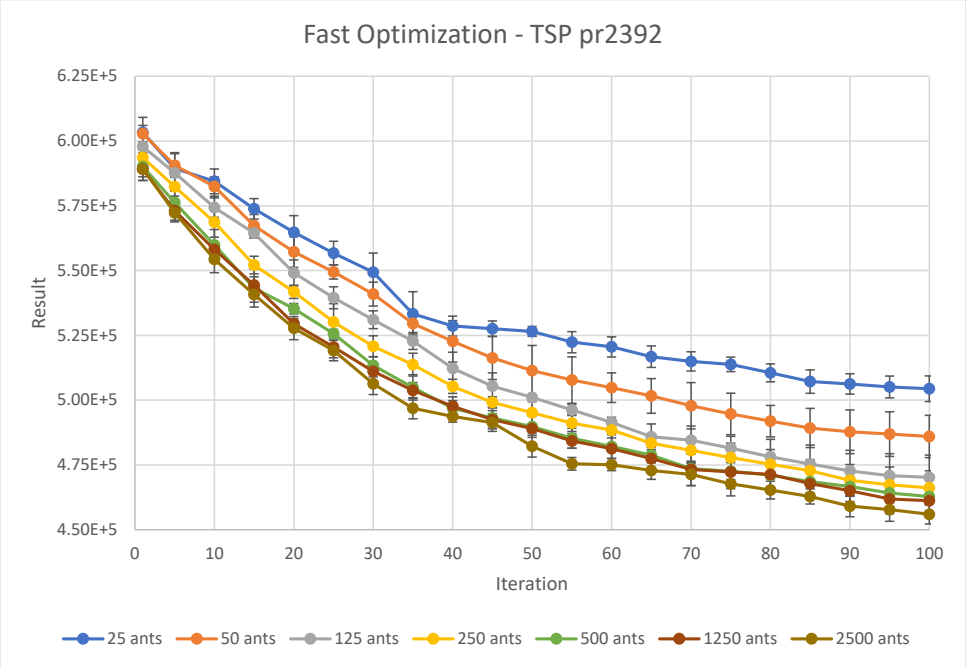


Figure 3.4: Best solution over iterations with varying number of agents – Distributed AS.

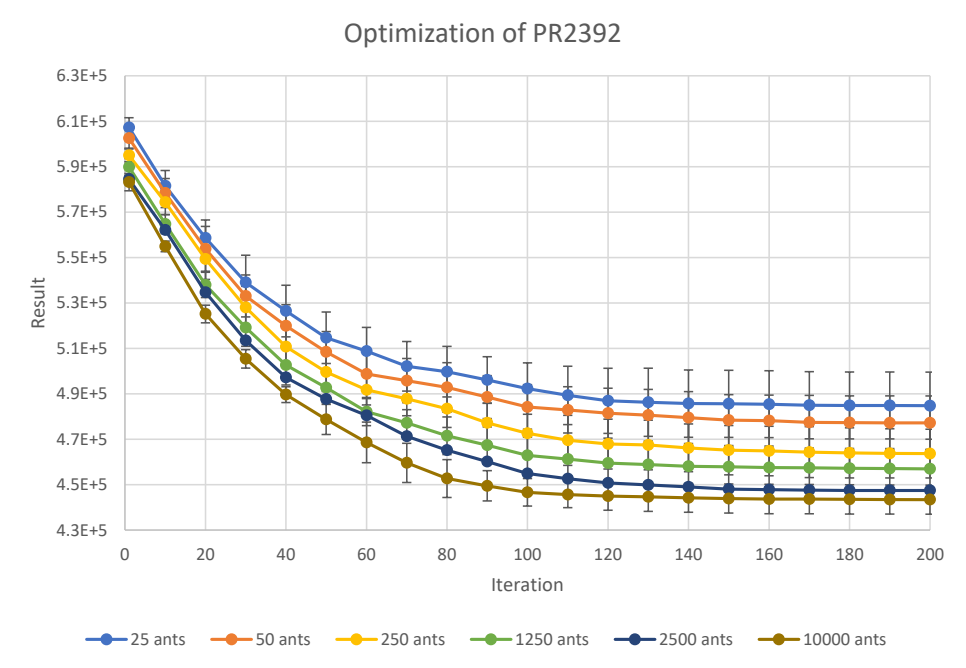


Figure 3.5: Best solution over iterations with varying number of agents – Desynchronized AS.

At the end of the day, what usually matters most is the final result of the optimization. Table 3.3 presents the summary of the results obtained for a reference sequential implementation of a standard MMAS algorithm[2] and both distributed and desynchronized version of the actor-based implementation of the ACO algorithm ([33] and Section 7, [34] and Section 8). The problem solved was a pr2392 instance from TSPLIB, each configuration was repeated 10 times. In order to obtain generalizable results, the algorithms did not use any local optimization mechanisms nor TSP specific assumptions.

	Result
Best known solution	378032
MMAS (25 ants / 2500 iterations)	494735.2 ± 2983.4
MMAS (250 ants / 2500 iterations)	489201.8 ± 1481.6
Distributed Ant System (250 ants / 1000 iterations)	455035.2 ± 5446.7
Distributed Ant System (500 ants / 1000 iterations)	452700 ± 5998.7
Desynchronized Ant System (250 ants / 200 iterations)	463671 ± 10727.0
Desynchronized Ant System (500 ants / 200 iterations)	459025 ± 10002.5
Desynchronized Ant System (2500 ants / 200 iterations)	447427 ± 3153.9
Desynchronized Ant System (10k ants / 200 iterations)	443377 ± 6392.4

Table 3.3: Comparison of results achieved by MMAS, Distributed Ant System and Desynchronized Ant System on *pr2392* from TSPLIB.

Note that the number of iterations varied across the algorithms, in favor of the simpler ones. Furthermore, the other parameters for each algorithm was analogous, which lead to the premature stagnation in case of the desynchronized version (see Fig. 3.5). It could be probably evaded by changing these parameters and lead to even better results.

The results show that:

- For all algorithms increasing the size of the colony improved obtained results.
- Distributed Ant System outperformed MMAS with the same size of the colony and fewer number of iterations which is a direct result of using multiple solutions for pheromone updates.
- Desynchronized Ant System achieved slightly worse results than Distributed Ant System for the same size of the colony (the smaller number of iterations does not influence this observation because, as Fig. 3.5 shows, the desynchronized algorithm already reaches stagnation within 200 iterations).

[2] `http://iridia.ulb.ac.be/~mdorigo/ACO/downloads/ACOTSP-1.03.tgz`

- The possibility to further increase the size of the colony using the desynchronized version enabled achieving better results than the distributed version, finally outperforming MMAS by 9%.

3.3 Solving large TSP instances

As a final experiment, the presented concepts were confronted with the biggest TSPLIB instances. With this end in view, a specialized version of the system was created. Aiming at a specific problem, the implementation could be adequately adjusted; the choice of used data structures was altered and TSP-specific features such as neighborhood limiting and 2-opt local optimization [11] were implemented. In order to reduce RAM requirements the system keeps pheromones only for the nearest neighbors. These changes resulted in further improvement of the system's performance and efficacy, i.e., better and faster solving of TSP instances. The system was tested with *pr2392*, *usa13509*, *pla33810* and *pla85900* problem instances from TSPLIB.

Figures 3.6, 3.7 and 3.8 show the cost of the best solution found so far in consecutive iterations for *pr2392*, *usa13509* and *pla33810* problems respectively. Each configuration was repeated 5 times for varying number of computational nodes. Each node was running 25 ants (total number of ants in the colony was equal to the number of nodes times 25), neighborhood was limited to 50 closest components and 2-opt algorithm was applied to each solution created by ants.

Fig. 3.6 shows that with 2-opt algorithm the system was able to find solutions close to optimal (378032) with average result 378836.8 on 200 nodes. Compared to the previous results, the problem specific optimizations allowed the system to start with much better solutions and achieve better final results. However, increasing ants colony size does not improve the results as significantly as before. In case of *usa13509* the influence is less visible and for *pla33810* it is not visible at all.

One may assume that the results for the bigger TSP instances depend mostly on 2-opt local optimization, so to verify it, more experiments were conducted for *pla85900* with a different configuration. This time each node was running 200 ants, neighborhood was limited to 200 and 2-opt was disabled.

Fig. 3.9 shows positive influence of the increasing number of ants on the final results. It proves that high number of ants might be useless for simple problems like TSP, where heuristic local optimizations are crucial for obtaining good results and the quality of the meta-heuristic itself is not so important. However, a huge colony improves the results if problem specific optimizations like 2-opt are not available.

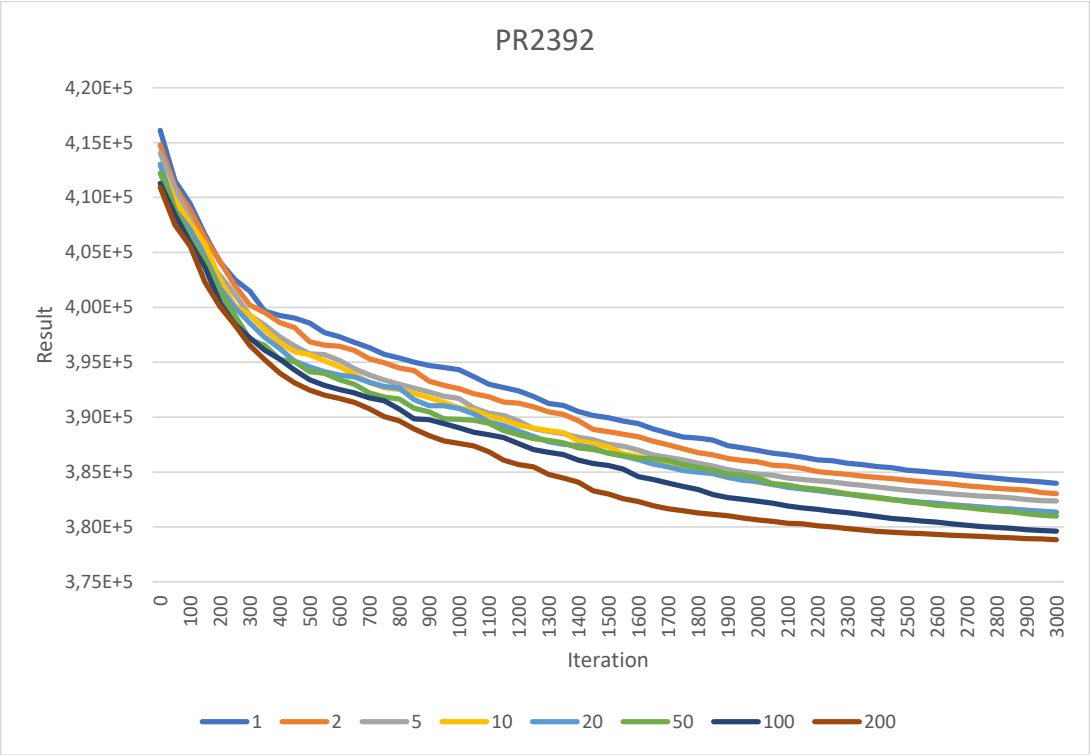


Figure 3.6: Result over iterations with varying number of computational nodes on *pr2392*.

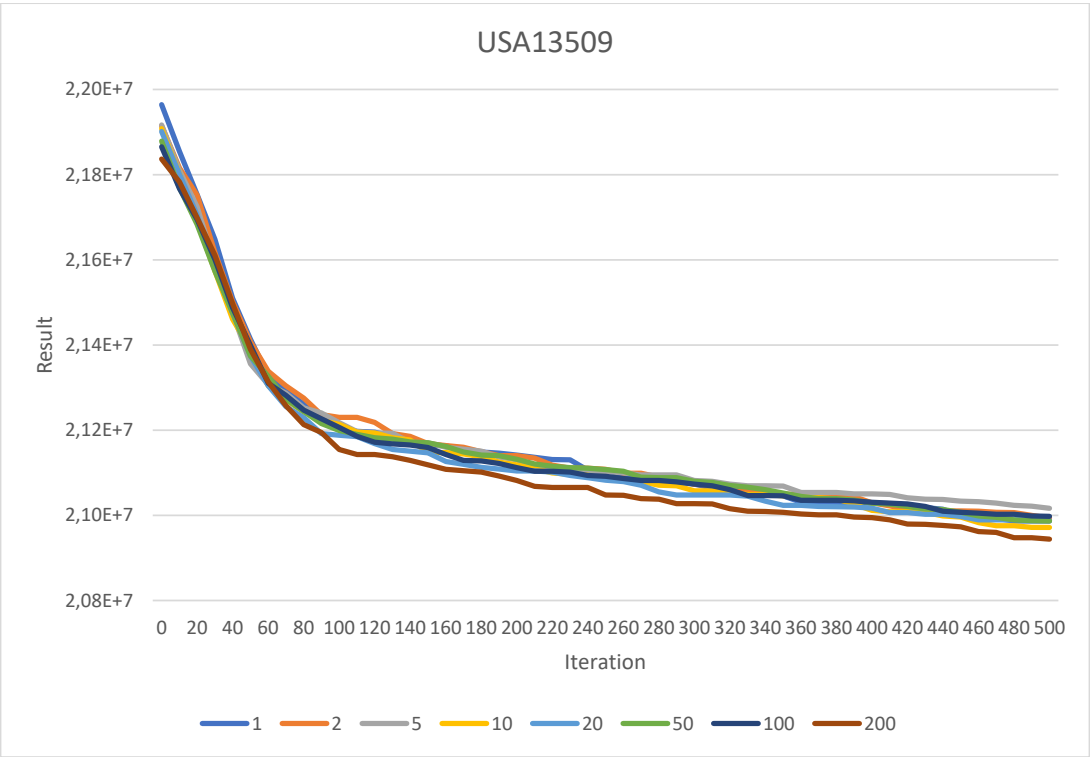


Figure 3.7: Result over iterations with varying number of computational nodes on *usa13509*.

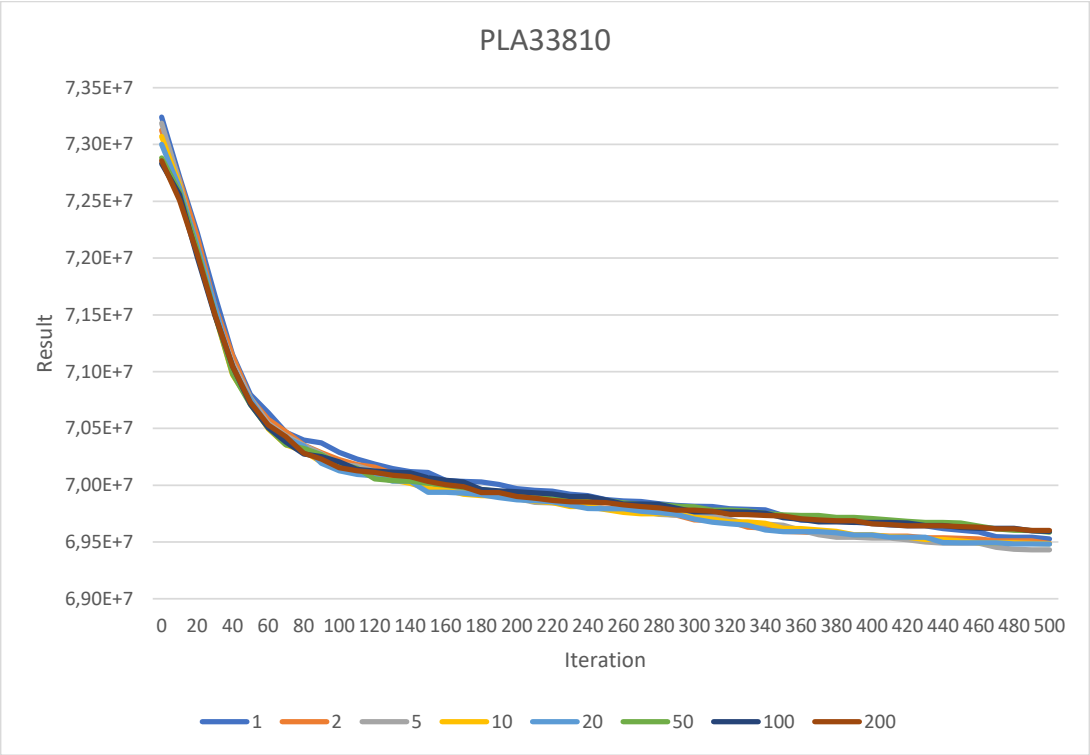


Figure 3.8: Result over iterations with varying number of computational nodes on *pla33810*.

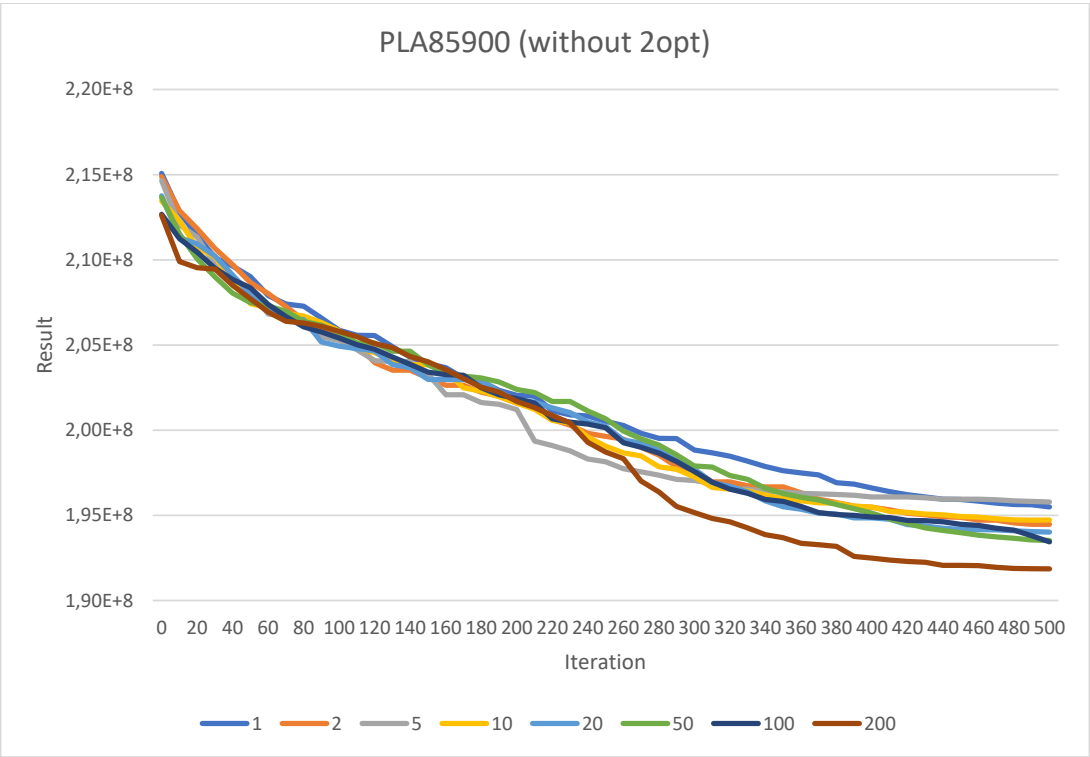


Figure 3.9: Result over iterations with varying number of computational nodes on *pla85900*.

Nodes	1	2	5	10	20	50	100	200
Speedup	-	2.09	5.24	10.44	20.49	49.09	89.85	153.11

Table 3.4: Speedup on pla85900 from TSPLIB.

Tab. 3.4 presents speedup of the new implementation on *pla85900* from TSPLIB. Up to 20 nodes it shows super-linear scalability, which results from distribution of the pheromone matrix between nodes. With further increasing number of nodes the efficiency slightly decreases and achieves 76.5% on 200 nodes.

Chapter 4

Desynchronization use-cases

The desynchronization concept might be applied to the other kinds of algorithms. These algorithms can benefit from better scalability, however, each application should be carefully validated, since it may affect the algorithms results. The altering of the results is unacceptable in some algorithms, where the results determinism is crucial. This chapter contains a short survey of possible desynchronization applications ([35] and Section 9).

4.1 Population-based optimization algorithms

In the previous chapters, the desynchronization of Ant Colony Optimization was presented. This concept might also be applied to other population-based optimization algorithms. This category of algorithms is randomized by design in order to explore the solution space, hence, the desynchronization does not affect the results determinism and might be considered as another random factor in the system.

The Particle Swarm Optimization [24] algorithm uses a swarm of agents, which independently explore the solution space driven by the global best solution. Groups of the agents can be easily distributed across the cluster nodes, however, synchronization of the global best solution can limit the system scalability. In order to mitigate this issue, updates of this value may be applied independently from the agents iterations. This modification will enable agents to efficiently utilize the cluster resources without any negative impact on the final results.

Another algorithm based on the global knowledge and independent agents is Social Cognitive Optimization [41]. The population knowledge is organized as a social sharing library, which is updated by each solution found by the agents. Similarly to the Particle Swarm Optimization and Ant Colony Optimization algorithms, it is straightforward

to parallelize the agents processes, however, updating the social sharing library may reduce the scalability. The updates desynchronization could mitigate this problem without significantly affecting solutions found.

The population-based optimization algorithms have a huge potential of proper utilization of big computation clusters and HPC environments. The independent agents are able to run on separate processors. When the global synchronization point of all processes is removed, the processors are used more efficiently and the global knowledge is updated as soon as possible.

4.2 Agent-based simulations

In the case of simulations, possibility of applying the desynchronization concept is not as straightforward as in the case of optimization algorithms. A lot of simulation algorithms are expected to be deterministic and the determinism is crucial to consider them as valid ones. In such case the desynchronization concept cannot be applied as it will affect the final simulation result and make it useless. Conversely, the notion of autonomy, e.g. in agent-based simulations, can be leveraged in order to successfully employ the idea of desynchronization.

This problem can be visualised by a simple simulation of Conway's Game of Life [10]. The board of the game can be split on two parts and each part can be simulated on an independent node. The points with neighbors on the other node have to communicate with them remotely, which reduces the system scalability. When the desynchronization of this communication is applied, by caching the last known state of remote points and updating them as soon as possible, the scalability increases, but the final result determinism is lost by introducing non-deterministic delays on data updates. To illustrate the problem, Fig. 4.1 presents the results of the Game of Life simulation with desynchronization and some forced communication delays and lost messages ([35] and Section 9). Even after a few iterations the results are significantly different and the differences grow with each iteration.

However, there are other simulations that focus on some statistical analysis of the simulated population rather than the result as such. A simulation of the Iterated Prisoner Dilemma with a huge population of agents may serve as an example here [30]. The simulation used various strategies for selecting pairs of confronting agents and a cooperation level in the population was measured. In order to handle a big population of independent agents, the agents are distributed across the nodes which forces some pairs to communicate remotely, and that has a negative impact on the system efficiency.

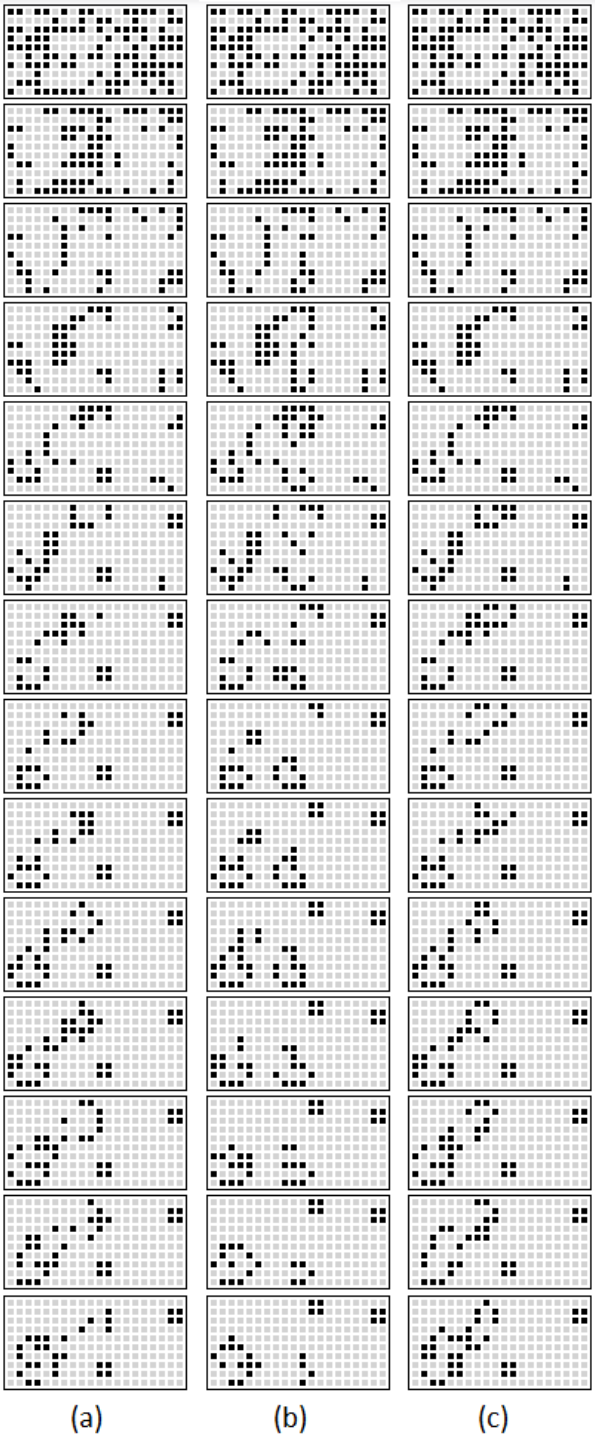
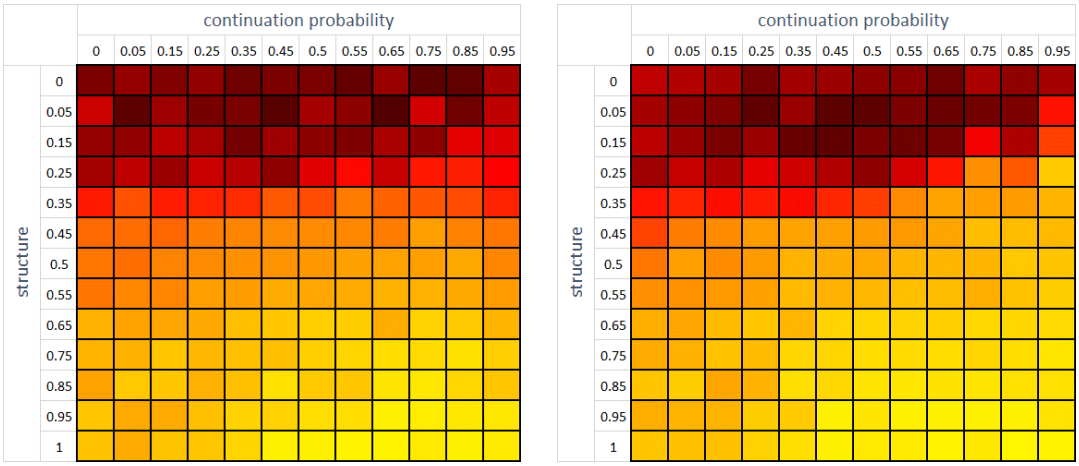


Figure 4.1: Conway's Game of Life simulation on two nodes with desynchronization: (a) simulation without delays; (b) simulation with cache update delayed by one iteration; (c) simulation with every fourth cache update lost.

The desynchronization was applied to this algorithm by introducing a local representation (a stub) of each remote neighboring agent. The stub behaves as the original agent and periodically synchronizes the strategy and the results with it. This way the scalability increased without any noticeable differences in the simulation results. Fig. 4.2 presents comparison of the statistical results from the big population simulation, where dark red color means low and bright yellow high cooperativity in the population. The results are not identical, however, some variations were visible between simulation runs. The simulation included randomness due to the probability of playing with the same neighbor again (continuation probability) and the probability of playing with a neighbor with a similar strategy (structure). Hence, the results can be considered as not affected by the desynchronization of the algorithm.



(a) Without desynchronization **(b)** With desynchronization

Figure 4.2: Desynchronization influence on IPD simulation from [30].

Each application of the desynchronization concept to simulations has to be considered carefully, since it can make the results useless if they were expected to be deterministic. However, when the simulation is focused on statistical results and is randomized by design, the desynchronization may not affect the outcome in any noticeable way. This is the case for simulations of huge populations where the interactions between agents are commonly randomized and the outcome is some statistical data of the whole population and not the state of each individual agent.

Chapter 5

Conclusions

In this extended summary, the design of distributed and desynchronized Ant Colony Optimization algorithm was presented together with the summary of experimental results. Chapter 2 starts with a summary of the latest accomplishments regarding usage of HPC environments to run optimization algorithms based on Ant Colony Optimization. Since the algorithm does not scale well with data synchronization after each iteration, the majority of research focused on multi-colony approaches with various levels of cooperation among the independent colonies.

Section 2.2 depicts how the actor model was employed to prepare a clear description of a distributed system, in order to simplify the system design and focus on the communication between agents. This enabled distribution of the single pheromone matrix across the cluster, which was described in Section 2.3. As presented in Sections 3.1 and 3.2 the distributed system can be scaled up to even 50 nodes without altering the colony's behavior.

In Section 2.4 the desynchronization concept was applied to the distributed Ant System. Since it removed the global synchronization point of the algorithm, it significantly increased the system scalability up to 400 nodes in the HPC environment. The scalability of the system made it possible to run a huge ants colony with a single pheromone matrix. As presented in Section 3.2, this results in improved final solutions quality thanks to more extensive exploration of the solution space. Based on efficient utilization of the knowledge collected by numerous agents, described in Section 2.5, both the distributed and the desynchronized algorithm outperformed a well-known sequential implementation of *Max-Min Ant System*.

In Chapter 4, the possibilities of desynchronization in other algorithms were presented. The considered algorithms and experiments lead to the conclusion that the requirement of deterministic results is crucial for the decision about applying this idea. If the algorithm

results are expected to be deterministic, like in a lot of simulations, the desynchronization will probably lead to disruption of this assumption and make the algorithm useless. On the other hand, if the results are not deterministic by design, because of the algorithm randomization, like in optimization algorithms and also some simulations, the desynchronization disruptions can be considered as just another random factor.

The thesis of this dissertation was as follows:

Introducing a distributed pheromone matrix to the Ant Colony Optimization algorithm will enable a scalable distribution of the computations within a cluster or supercomputer environment, improving the quality of the obtained results in comparison to the referential sequential algorithms.

To the best of my knowledge, the aforementioned thesis was proven, taking into account the following achievements presented in the work:

- Modeling the ACO algorithm using actor model, which lead to a fairly good scalability of the classic Ant System (see Section 2.2 and [32]).
- Introducing a distributed pheromone matrix, which lead to even distribution of work related to the pheromone trails updates among computational nodes (see Section 2.3 and [33]).
- Elimination of the global synchronization point at the pheromones update, which lead to linear scalability up to 400 nodes (see Section 2.4, Section 3.1 and [34]).
- Evaluating various strategies of selecting solutions for pheromone updates, which lead to proper leveraging of the knowledge collected by the extended ants colony, significantly outperforming the well-known Ant Colony Optimization variants by 9% (see Section 2.5, Section 3.2 and [33]).
- Implementing an efficient TSP specific version of the system, aiming at the biggest problem instances from TSPLIB (see Section 3.3, it is an original contribution of this extended summary).
- Providing a more general guideline for applying desynchronization to other algorithms and simulations with indication of potential pitfalls (see Section 4 and [35]).

In the future, the possibilities of applying a similar concept to other algorithms and simulations, as described in Chapter 4, should be explored in depth. One of the research fields that should be easily able to benefit from such desynchronization are other population-based optimization algorithms. They are often randomized by design and any

read or write access to global knowledge might be desynchronized resulting in better scalability and thereby ability to employ a larger population resulting in extended exploration. Desynchronization could also be of advantage to various multi-agent simulations. However, in this case introduced modification have to be carefully analyzed before drawing any final conclusions from desynchronized simulations. It has to incorporate randomization by design or the result that is of interest has to be some statistical data from the simulation. The opportunities of desynchronized simulations are already investigated by one of the co-authors of the survey.

Part II

Contributions

Chapter 6

Distributed ant colony optimization based on actor model

In this paper, a novel approach to the implementation of a parallel and distributed Ant System algorithm was proposed. It leverages the actor model of concurrency in order to simplify the system analysis and implementation. However, it was supposed not to violate any of the standard Ant System properties. The proposed system scaled up well to 30 nodes in HPC environment. It was experimentally compared to the original Ant System algorithm with no significant differences detected in terms of results quality.

Contributions of Mateusz Starzec: In this publication, I designed and co-developed the distributed Ant System based on the actor model of concurrency. I have also prepared scalability tests in HPC environment. I was the primary author of the section *Distributed ant colony platform based on actor model* and the scalability related parts of *Experimental results*. I was co-author of sections *Introduction*, *Conclusion* and *Parallel and distributed ant colony computing*.

Parallel Computing 90 (2019) 102573

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Distributed ant colony optimization based on actor model

Mateusz Starzec, Grazyna Starzec, Aleksander Byrski*, Wojciech Turek

Faculty of Computer Science, Electronics and Telecommunications, Department of Computer Science, AGH University of Science and Technology, Al. Mickiewicza 30, Krakow 30-059, Poland

ARTICLE INFO

Article history:
Received 8 June 2018
Revised 9 April 2019
Accepted 9 July 2019
Available online 18 October 2019

Keywords:
Ant colony optimization
Distributed computing
Actor model of parallelism
Scala/Akka

ABSTRACT

The parallelization of metaheuristics and care for the efficient use of the available infrastructure is very popular in the case of population-based algorithms (e.g., evolutionary ones), as many of them have structures intrinsically easy for parallelization. However, swarm computing algorithms (ACO in particular) must use certain global knowledge in order to be properly implemented (e.g., a pheromone matrix in the case of ACO algorithms). Thus, the parallelization of ACO is known to be difficult to realize. In this paper, we propose an actor-based approach for constructing an efficient and robust ACO implementation that leverages the HPC infrastructure. The presented results show the ability to be scaled for up to 30 nodes, and the relevant results support the claim that the implemented algorithm is equal to the original Ant System algorithm. Improving it further and increasing its scalability with the planned asynchrony in the pheromone matrix updates is envisioned as a direct future work.

1. Introduction

According to Wolpert and MacReady's well-known no free lunch theorem [34], there is always need for tailoring an optimization algorithm for particular needs, and if that is not enough, creation of new metaheuristic algorithm might be inevitable. However, developing a truly novel algorithm that can be proven to be useful is a very difficult task. Therefore, many researchers have proposed variants of existing algorithms; these have often turned out to be nearly equivalent to previously published solutions [28]. This process can slowly lead to more and more efficient algorithms, which seems similar in its essence to optimization itself, slowly converging to optimum.

These methods are very often quite complex, requiring time, experience, and attention when choosing or developing suitable tools, implementing the algorithms, and testing them. It seems that a significant gain in efficiency can be achieved in this area. Improving the performance of existing well-recognized metaheuristics can result in far better results than modifications of the algorithms themselves can offer (especially when massively parallel execution is considered).

Nowadays, metaheuristics (especially population-based ones) are often implemented in consideration of parallelization. In particular, the world of evolution has been quite well-explored in this matter, yielding the famous master-slave, cellular, and island-based parallelization models of evolutionary algorithms [6]. These can more or less be efficiently implemented using any available homogeneous or heterogeneous infrastructure. Thus, one can implement a dedicated computing system or even a more general framework that is able to scale well by leveraging the existing hardware. Efficiently utilizing the infrastructure can help solve any problem posed in a limited amount of time; this has become an important factor in many applications (e.g., in the case of traffic optimization [31]).

When considering Ant Colony Optimization [9], a number of approaches have been made to implement such systems using distributed [15], parallel [23], or even heterogeneous environments [7,27]. To the best of our knowledge, however, there are no computing systems that are capable of efficiently running ACO-computing in large-scale distributed systems (i.e., HPC-grade).

An easy-to-use, highly scalable, and robust way of implementing ACO-computing in a parallel or distributed environment can be based on actor-based parallelism that can be supported by Scala/Akka, for example [3]. Moreover, applying a high-level language instead of leveraging previously proven and efficient techniques such as MPI will make the development process easy and not so prone to errors; this is the case when using low-level programming languages.

In this paper, we would like to propose a novel approach to the implementation of a parallel and distributed ACO by leveraging the actor-model of concurrency. The proposed computing

* Corresponding author.
E-mail addresses: mateusz.starzec@iisg.agh.edu.pl (M. Starzec), grazyna.starzec@iisg.agh.edu.pl (G. Starzec), olekb@agh.edu.pl (A. Byrski), wojciech.turek@agh.edu.pl (W. Turek).

https://doi.org/10.1016/j.parco.2019.102573
0167-8191/

system is designed to be capable of efficiently leveraging the available HPC infrastructure. The presented version can be efficiently run on a cluster of 30 computing nodes; our relevant results will be presented. It is noteworthy that the designed system can run thousands of ants, and the implementation is negligibly different than the sequential version (such results will also be presented). After surpassing the number of 30 nodes, the apparent room for improvement calls for further enhancement of the proposed technique; i.e., the exploration of asynchrony is planned to be realized in the future.

In the next section, the background of the ACO-research will be sketched out in the context of parallelization. The proposed technique will also be discussed after positioning the presented paper in the state-of-the-art literature, showing the relationships of the agents and their interactions. Next, our experimental results will be shown, proving the applicability of the proposed approach. This will be followed by our conclusions.

2. Parallel and distributed ant colony computing

Ant colony optimization algorithms are inspired by the collective behavior of natural ant colonies. The first version of this algorithm was introduced in [9] to solve the traveling salesman problem (TSP). As a meta-heuristic algorithm, ACO was described in [10].

When using ACO algorithms, the optimization problem can be considered as a graph consisting of a finite set of *components* connected by edges with assigned costs. A feasible solution is a path that respects the posed restrictions and fulfills the requirements defined by the problem. The cost of a solution is defined as the function of all of the costs of all connections belonging to the solution. The optimal solution is the path with the minimum cost.

ACO algorithms are based on a population of ants (agents) that iteratively traverse a graph searching for an optimal solution. During each iteration, each ant starts its journey in some initial vertex and gradually builds a solution by moving to subsequent vertices guided by a probabilistic decision rule. The rule is influenced by the values of the pheromone trails left on the edges by the previous generations of ants. In the basic ACO algorithm – Ant System (AS, [12]) – the probability of moving from component i to component j for ant k in iteration t is defined as follows:

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}(t)]^{\beta}}{\sum_{k \in allowed_k} [\tau_{ik}(t)]^{\alpha} \cdot [\eta_{ik}(t)]^{\beta}} & \text{if } j \in allowed_k \\ 0 & \text{otherwise} \end{cases}, \quad (1)$$

where $\tau_{ij}(t)$ is the intensity of a pheromone trail on an edge and $\eta_{ij}(t)$ is the visibility of the edge, which can be defined as the inverse distance between cities in the case of the TSP. The parameters that control the relative importance of trail versus visibility are α and β. Finally, $allowed_k$ is a set of possible transitions for ant k. The tour ends when a feasible solution is found. Based on its knowledge, the ants update the pheromone trails on their paths. In the classic AS version, the update is performed at the end of a single iteration according to the following formula:

$$\tau_{ij}(t+1) = \rho \tau_{ij}(t) + \sum_{k=1}^{m} \Delta \tau_{ij}^{k}, \quad (2)$$

where $\rho \in [0, 1)$ is a pheromone persistence coefficient and m is the number of ants. The pheromone update value for each ant is defined as follows:

$$\Delta \tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k} & \text{if } k\text{th ant uses edge } (i, j) \\ 0 & \text{otherwise} \end{cases}, \quad (3)$$

where L_k is the tour length of the kth ant and Q is a constant (often with a value of 1).

2.1. Parallel ACO

Since the first version of the ACO algorithm, multiple variations have been proposed to further improve its effectiveness and performance. One type of such a variation is based on introducing modifications to the original sequential algorithm but without introducing parallelism or distribution; e.g., Elitist Ant System (EAS), Rank-based Ant System (ASRank, [4]), Max-Min Ant System (MMAS, [29]), or Ant Colony System (ACS, [11]). Another direction is to boost performance by leveraging the parallelization of the algorithm. Numerous articles have been published that propose a parallel or distributed version of the ACO algorithm, and others tried to put forward a taxonomy for categorizing them in a unified way. The taxonomy proposed in [25] classifies the algorithms into four main categories based on two primary criteria: the number of colonies (one or many) and the cooperation or lack thereof among the parallel parts:

- *Master–slave model* – a single colony without cooperation, where the master process manages the global information,
- *Cellular model* – a single colony with cooperation, where the colony is divided into small overlapping neighborhoods with their own pheromone matrices,
- *Parallel independent runs model* – several sequential ACOs are concurrently and independently executed on a set of processors using identical or different parameters,
- *Multi-colony model* – several colonies explore the solution space with their own pheromone matrices but periodically exchange information.

Despite the multitude of parallel algorithms, there are only a few articles regarding the implementation of a single ant colony in a cluster environment. The most common use-case for clusters is to run multiple colonies. Ilie and Bădică [15] presented an agent-based approach to ACO modeling. This article reports a very good scalability of up to 7 nodes on the *gr666* problem from TSPLIB[1]. The algorithm reaches solutions that are comparable to the standard Ant System. Unfortunately, there were no results for large-scale TSP problems nor larger clusters.

Over the last few years, many researchers have focused on a GPU-based ACOs [7,27,36,37] as well as the parallelization of other swarm intelligence algorithms [22]. In [7], Cecilia et al. proposed two algorithms, each with around a five-fold speedup and solutions comparable to a standard sequential ACO. The third proposed algorithm was up to 22 times faster than the sequential ACO running on CPU, but the solutions that were found in this algorithm were worse. Skinderowicz [27] reported a speedup of up to 20-fold on a GPU with very good mean errors from the best-known solutions. Unfortunately, these algorithms are not prepared for further scalability for large-scale problems.

The majority of research has been focused on ACO modifications in order to solve big problems faster, usually at the cost of worse solutions. To the best of our knowledge, there have been no successful experiments that have implemented the standard ACO algorithms in a parallel or scalable way.

2.2. Agent-based computations

The Ant Colony Optimization algorithm and other swarm intelligence algorithms are very often connected with and described by agent-related notions [15,19,21]. Thus, the agent model can become a very handy perspective that can be used in the case of meta-heuristics that are potentially capable of being run in parallel (or in HPC at the end) for both

[1] https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.

- the interpretation, design, and extension of the algorithms themselves, treating the parts of the algorithms as agents and describing their interactions using the agent-oriented approach (see, e.g., [5]),
- the development of parallel swarm-based metaheuristics, treating the parts of the implementation as agents [3] and using the agent-related terminology to describe their interactions and relationships (see, e.g., [18,33]).

There are numerous frameworks that have originally supported agent-based modeling [1]. Based on the message passing interface (MPI), the REPAST HPC framework is an example of such a framework [8]. Another example is FLAME [16,17], which is based on a distributed memory model. Development based on these frameworks is usually difficult, although the technology they use has been proven in numerous ways for implementing parallel, distributed, and HPC applications. Therefore, turning to more-current technologies might be useful in order to ease the development of such frameworks.

The concept of software agents is strongly inspired by the actor model proposed by Hewitt and coworkers [2,14]. It introduces the message passing concurrency model and defines the concurrent processes as the basic units of parallel computations. These processes use asynchronous messages as the communication mechanism (as opposed to the shared-memory method). As a result of receiving a message, an actor can change its state, send messages, or create new actors.

The actor model has experienced a renaissance in recent years thanks to its efficiency in handling massive concurrency on multicore CPUs. It has also been proven to be efficient in building distributed and scalable applications as well as more productive than lower-level APIs and frameworks [13,26]. The Akka framework supports actor-based system modeling and fulfills all of the important features of the multi-agent environment listed by Leitao et al. [20]. It focuses on the fast passing of messages between actors. Each actor processes the messages one-by-one, so there is no need to consider parallel execution problems (e.g., race conditions) inside a single-actor implementation. The abstraction layer over the problems of communication in a cluster enables a faster implementation by focusing on a domain of the system.

In this paper, we would like to focus on leveraging the agent model in order to implement a scalable and reliable system by using novel and easy-to-apply technology like the actor model delivered by Scala and Akka.

3. Distributed ant colony platform based on actor model

This section presents the system architecture that allows us to implement the ACO algorithms in terms of an actor model with high performance and scalability and reach solutions equal to the standard sequential implementation. This project does not introduce any relaxation of the standard Ant System rules.

The system introduces the following actor types (see Fig. 1):

- *Computation Master* – a singleton deployed on the master node, sets up the *Computation Local* actors on every node and collects computation statistics (e.g., the best solution) from all nodes.
- *Computation Local* – one per node, sets up the system-managing actors and collects statistics from them, aggregates the statistics, and passes them on to the *Computation Master*.
- *Problem Description Manager* – one per node, provides possible moves for an ant based on a previously constructed partial solution; internally, it can forward a request to a sub-actor sharing the same problem description if the problem is immutable.
- *Pheromone Manager* – one per node, keeps a part of the pheromone matrix assigned to its node, collects complete routes, and applies pheromone updates after each iteration; collects pheromone trails from remote managers when needed and keeps them in the cache.
- *Ant Manager* – one per node, starts the ants and collects routes from them.
- *Ant* – many per node, collects data from the local pheromone and problem description managers and iteratively constructs a solution.

The communication between the actors is presented in Fig. 2.

3.1. Pheromone matrix

The pheromone matrix is uniformly distributed across all system nodes. The ownership is organized by rows to keep the data regarding the edges outgoing from a single component in a single node, as the requests from the ants always contain edges with the same origins. The *Pheromone Manager* responsible for collecting the pheromone values for the ants provides the owned values to the remote *Pheromone Managers* and applies updates on the owned pheromone trails at the end of each iteration.

When an ant sends a request for the pheromone values on a set of edges, the local *Pheromone Manager* checks whether it has all of these edges in its part of the pheromone matrix. If yes, it sends a response fulfilled by a trivial filtering of the values from the matrix; otherwise, it sends the request to the owner of these edges to get the whole row of the matrix (in order to optimize inter-node communication). The ownership information of the parts of the matrix is distributed during the initialization of the system. The response is collected and saved in a cache, and the ant's request is finally answered. The *Pheromone Manager* also keeps track of the remote requests in progress in order to not duplicate the same requests while it is waiting for a response.

In the case of a request from the remote *Pheromone Manager*, the actor collects data from its part of the matrix and sends the response. The sender keeps it in the cache, which uses an LRU[2] update strategy. The size of the cache should be configured to fit within the available memory of the system and also avoid the constant replacing of the cached data. In the case of the TSP, the optimal size of the cache should fit all of the remote pheromone matrix parts. If this is not possible, then the beginning of the calculations might be slower; however, this should become faster when the exploration decreases after the few first iterations. The optimal size of the cache should be estimated based on the type of problem to be solved, its representation, and the possible ant movements over the graph. In the case of the TSP, all solutions require a visit in all vertices (effectively synchronizing the whole pheromone matrix), but this requirement is generally not applicable to other optimization problems. Hence, the distribution of the pheromone matrix combined with the remote communication and caching has two main advantages: the whole pheromone matrix does not need to fit in the system's memory all at once, and we avoid the synchronization of the unnecessary parts of the matrix. The synchronization process is also split into multiple parts, so the computations do not block until the whole matrix is synchronized. Furthermore, the process of updating the matrix at the end of each iteration is distributed, which is more noticeable the bigger the problem is.

When the *Pheromone Manager* collects the routes from all of the local ants, it sends them batched to all other *Pheromone Managers* and waits for the batches from them. When it has all of the solutions from iteration t, it updates its part of the pheromone matrix and only then starts sending responses to the requests regarding iteration $t+1$.

[2] Least recently used.

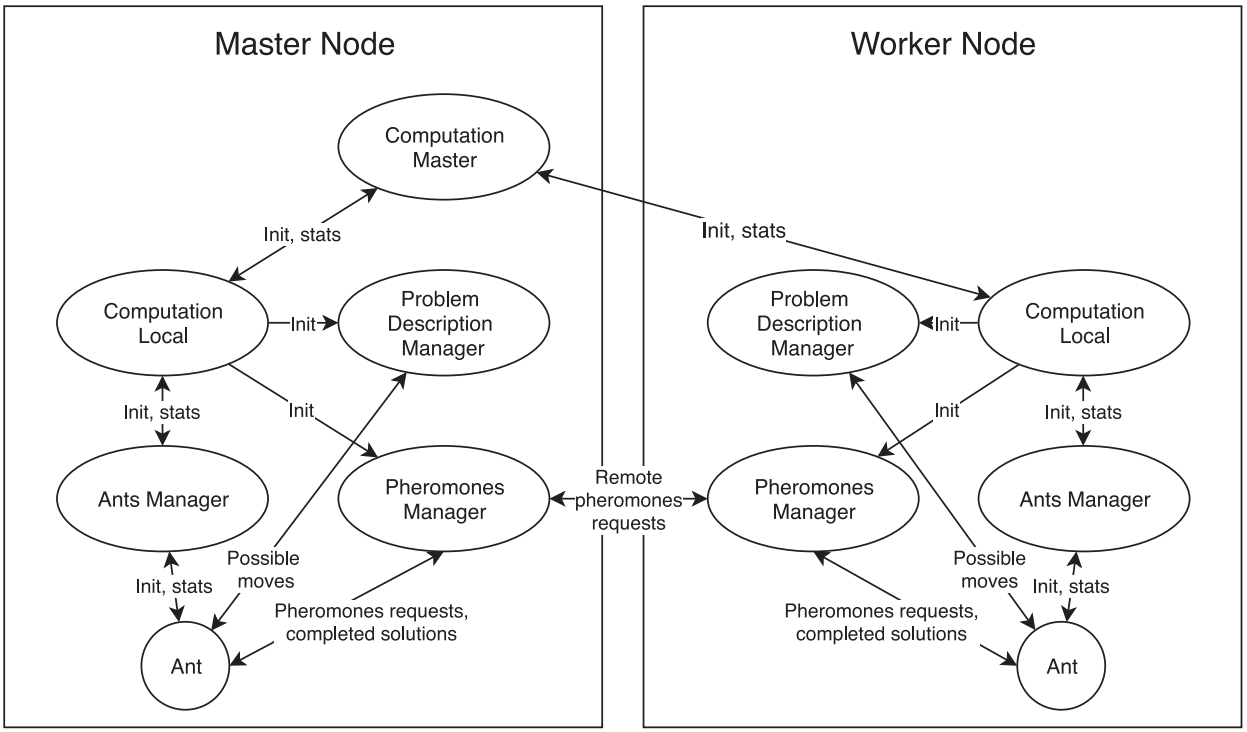


Fig. 1. Relationships and data passed between actors.

This matrix organization does not fit any of the system categories proposed in [25]. The master node is used only for the collection of statistics (including the current best solution), but the global algorithm knowledge (i.e., the pheromone matrix) is evenly distributed across the system nodes.

3.2. Behavior of ant

When an ant is created, it sends a request to the local *Problem Description Manager* to get the starting node and possible moves from this node. Then, it asks the local *Pheromone Manager* for pheromone trails on these edges and selects the move by the probabilistic decision rule (see Eq. (1)). The ant repeats these two steps until the *Problem Description Manager* answers that the solution is complete. Then, the ant sends the complete solution to the local *Ant Manager* to be compared with all other solutions and to the local *Pheromone Manager* to update the pheromone matrix as described above. The ants never communicate with the remote actors; all remote data from the pheromone matrix is provided by the local *Pheromone Manager*.

When an ant finds a solution, it reports the solution to the local *Ant Manager*. The manager collects the solutions from all of the local ants, selects the best one, and sends it to the *Computation Master* actor. This actor then selects the best solution from the reported local bests.

3.3. Distributed versus sequential ant system

The proposed distributed system does not violate any Ant System constraints. The asynchrony of the system does not make consecutive iterations overlap. The synchronization point at the end of an iteration is fuzzy, but the pheromone matrix update is based on all of the solutions found in the iteration, and each ant creates exactly one solution in each iteration based on the pheromone values updated after the previous iteration (as in the sequential implementation).

4. Experimental results

The experiments were conducted on the Prometheus supercomputer—a peta-scale (2.4 PFlop) cluster located in the Academic Computer Center Cyfronet AGH in Krakow, Poland. As of November 2018, Prometheus was ranked 131th in the TOP500 fastest supercomputer list. Prometheus is a cluster of HP Apollo 8000 nodes, each with 24 Xeon E5-2680v3 CPUs working at a 2.5GHz frequency and connected via an InfiniBand FDR network.

On this infrastructure, we ran various experiments solving some of the TSP problems from TSPLIB. As the goal of this paper is to present a general-use meta-heuristic algorithm, we have decided not to implement any TSP specific local optimizations (e.g., 3-opt) or performance improvements (e.g., narrowed neighborhood in the planning of the next step). This decision makes the results more likely to be applicable in case of other problems. First, we presented the set of experiments comparing the effectiveness of the distributed algorithm using a varying number of ants. Then, we moved on to the experiments that proved the scalability of our solution. Finally, we presented a comparison of the behavior of our distributed and sequential versions of the ACO algorithm to show the correspondence between them; i.e., that there is no modification introduced in the concept of the algorithm itself.

Some algorithm parameters remained unchanged for all of the experiments (see Eq. (1)):

- pheromone initial value $\tau_{ij}(0) = 0.01$ [12],
- pheromone update constant $Q = 1$,
- pheromone persistence coefficient $\rho = 0.99$,
- pheromone trial importance $\alpha = 2$,

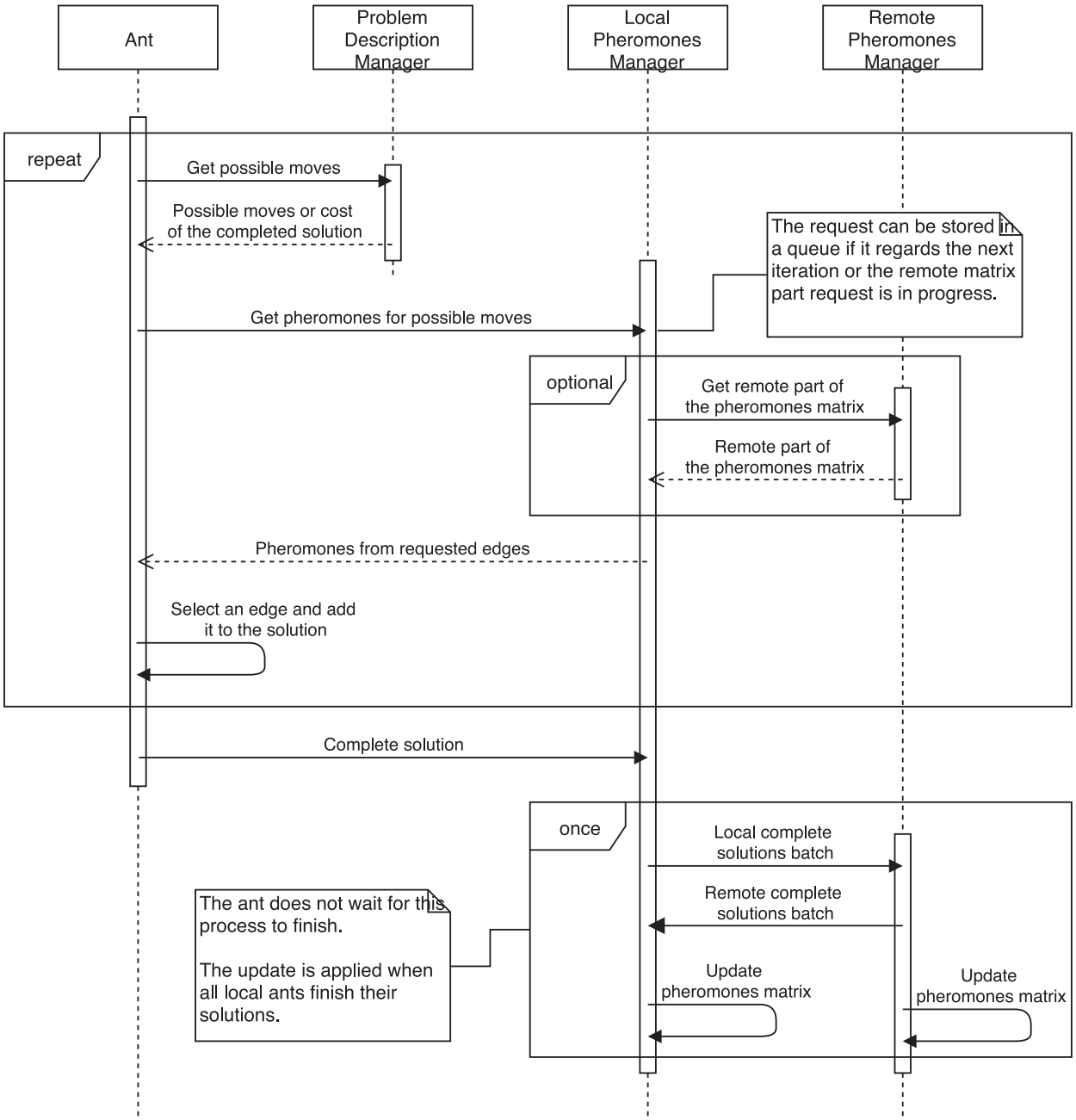


Fig. 2. Communication between actors in one iteration.

- heuristic visibility importance $\beta = 3$.

The first set of experiments compares the behavior of the algorithm with varying numbers of ants in the colony. For this, we used the u1432 and pr2392 problems from TSPLIB and applied our distributed version of the ACO algorithm with colony sizes of 500, 1000, 2500, 5000, and 10,000 distributed as groups of 500 ants per node. The comparison is based on the plot of the cost of the best solution found thus far versus time, presented in Figs. 3 and 4. Each configuration was repeated five times, and the standard deviation was negligible. Notice that we do not aim to improve Ant System in terms of the best solution found. We are going to focus on this matter in the future; for now, we will focus on the scalability of the system.

The results show that the number of ants in the colony does not have a strong influence on the final result of the algorithm, but it strongly improves the solutions found in the initial part of the algorithm. This appears to be a direct result of the forceful exploration of the search space. It is also important to note that even these two examples show that this feature has its limits; at some point, we reach a saturation. For the smaller problem (u1432), we can see that the difference between 5000 and 10,000 ants is

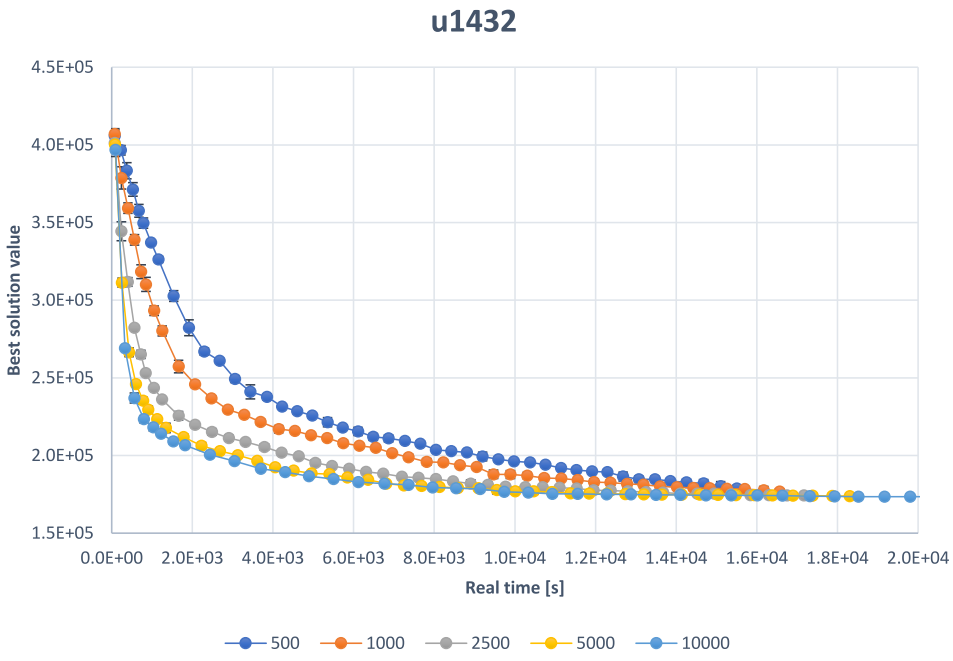


Fig. 3. Best solution over computation time with different ant numbers; Problem TSPLIB: u1432.

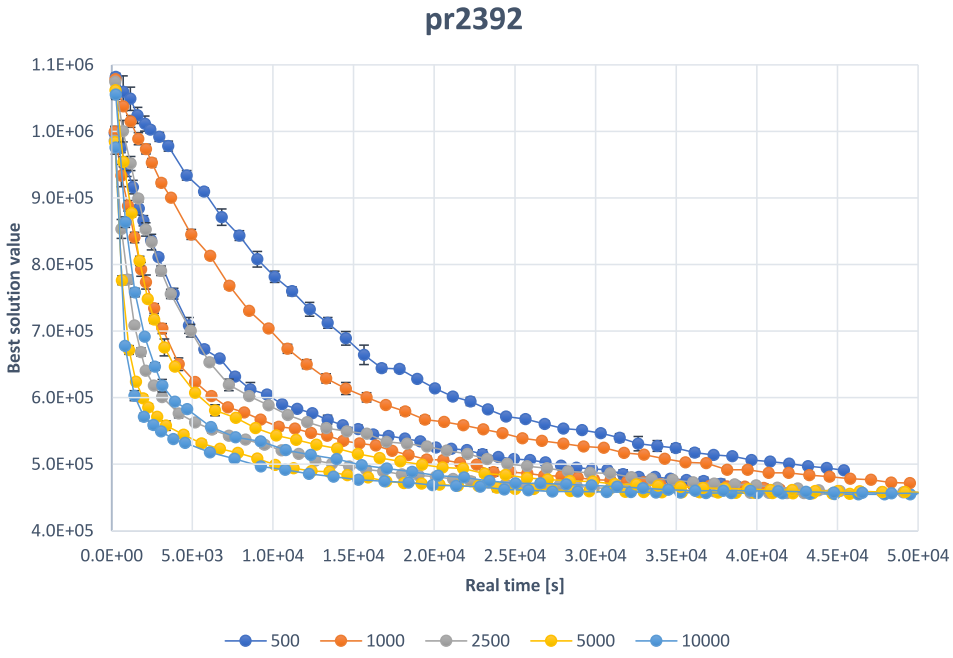


Fig. 4. Best solution over computation time with different ant numbers; Problem TSPLIB: pr2392.

minimal; however, once the size of the problem increases (pr2392), the margin between those two colonies again becomes significant. As visible in the chart, the repetitions showed a very small variation. These results allow us to argue that having a larger population of ants can lead us to obtain earlier results closer to the optimum sought - this may be especially desirable in those cases where the prompt response of the optimization system is needed; e.g., in the cases of large-scale simulations and the optimization of traffic (cf., e.g., [30,32]). Also, having a large population of ants that are able to be run in a very limited time (near real-time) requires the application of reliable and robust methods for implementing dedicated HPC-grade software systems.

The next set of experiments was performed to show the scalability potential of our solution. For this, we applied our algorithm

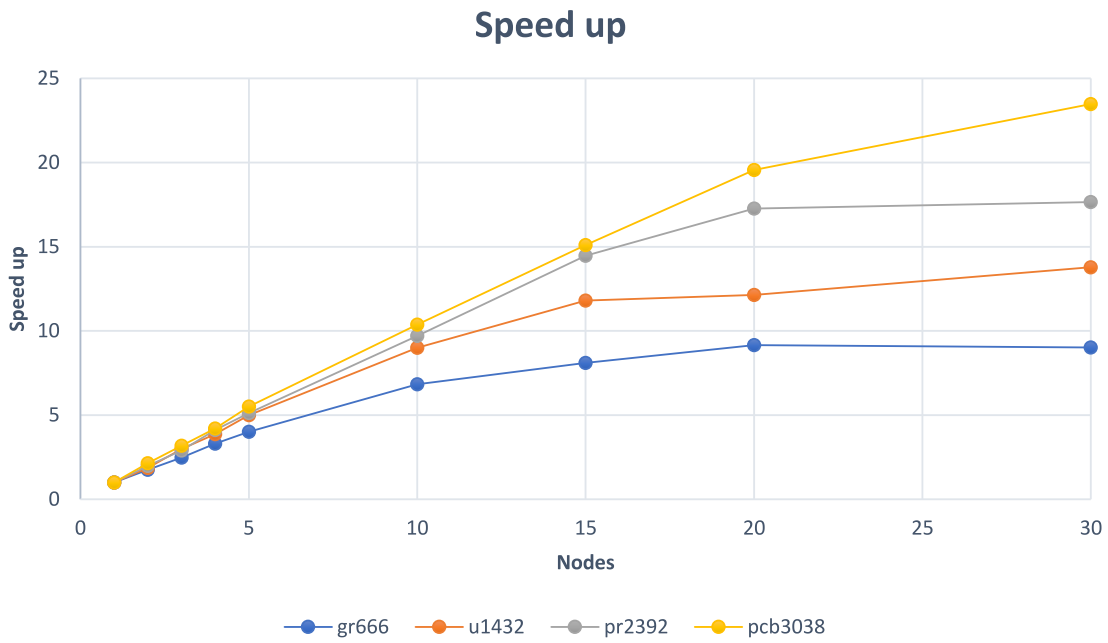


Fig. 5. System scalability with 10k ants.

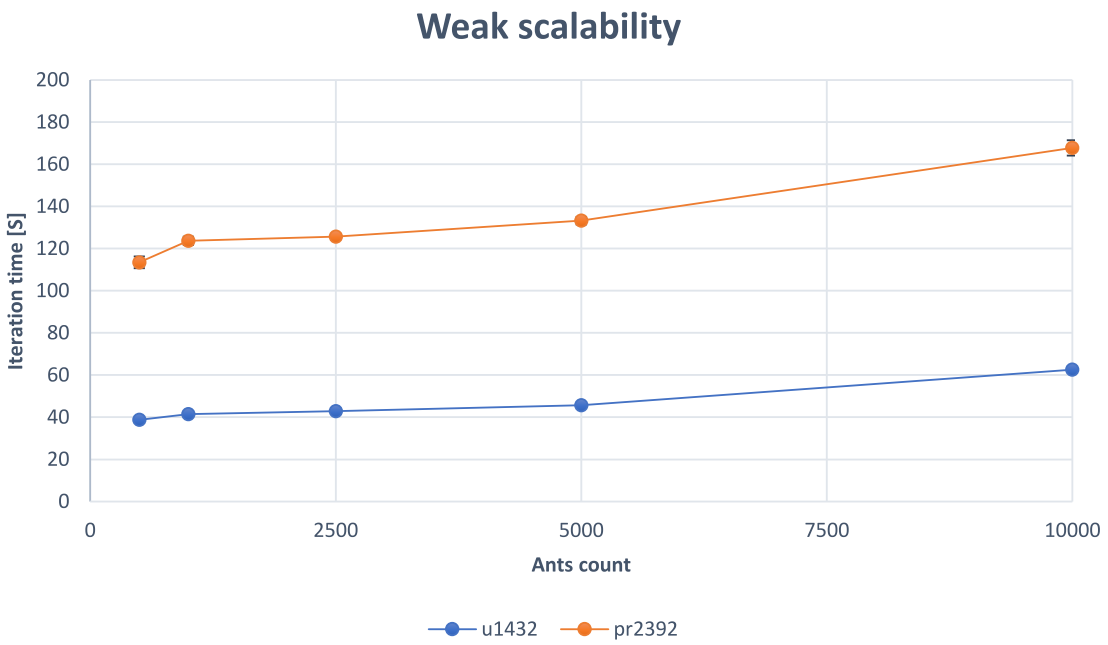


Fig. 6. Single iteration duration with different ant numbers distributed as groups of 500 ants per node.

to four TSPLIB problems with increasing sizes (666, 1432, 2392, and 3038) using a colony of 10,000 ants. Such configurations were all run with the colony being evenly distributed across a varying number of 24-core nodes (from 1 up to 30). Fig. 5 presents the speedup of the algorithm versus the number of nodes, where the speedup was calculated as the proportion of the time length of a single iteration on a single node to the time length of a single iteration on multiple nodes. Every configuration was repeated five times, and the standard deviation of a single iteration duration never exceeded 5% of the mean value.

The results show that the scalability of the solution strictly depends on the size of the problem. For up to five nodes, the algorithm scales well for all problems; however, the bigger the problem, the better is the speedup is for a larger number of nodes. This can be explained by the fact that, for larger problems, the amount of work that needs to be done in a single iteration is greater, so the

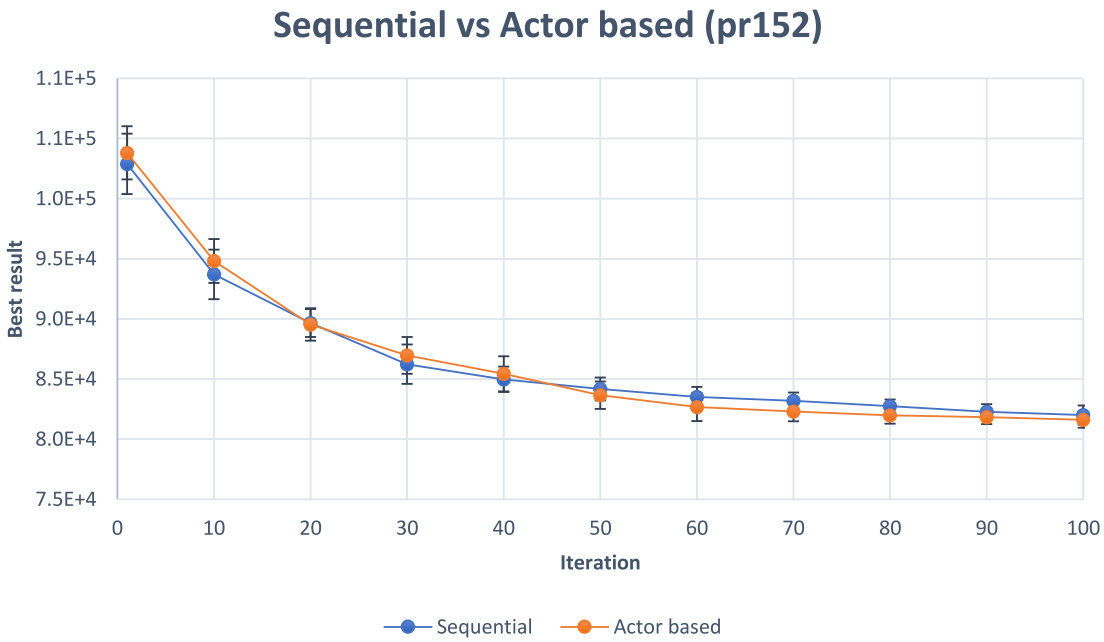


Fig. 7. Single iteration duration with different ant numbers distributed as groups of 500 ants per node.

synchronization of data (found solutions, pheromone updates) that happens after each iteration has less of an impact on the overall performance. All of the observed curves tend to become visibly flat after exceeding a number of nodes equal to 20. In order to overcome this issue, we will focus on desynchronizing the pheromone update between the ant sub-populations in future work, striving towards implementing a system that will leverage the available HPC infrastructure even better.

Another experiment regarding scalability was a test of weak scalability. This was based on two TSPLIB problems that were solved by the algorithm using between 1 and 20 nodes with 500 ants on each node (so, the each colony's size varied from between 500 and 10,000 ants). Fig. 6 is a plot of the time length of a single iteration versus each colony's size. The results demonstrate that our implementation shows good weak scalability – extending the numbers of nodes from 1 to 20 for both problems (thus, multiplying the number of ants in each colony by a factor of 20) increased the duration of a single iteration of the system solving the pr2392 problem by only 17% (from 500 to 5000 ants) and about 48% (from 500 to 10,000 ants). In the case of the u1432 problem, the increase observed was about 18% (from 500 to 5000 ants) and about 61% (from 500 to 10,000 ants). This seems to be another encouragement for working on the desynchronization of the pheromone matrix in order to leverage the available HPC infrastructure to an even greater extent. Each configuration was repeated five times, and the standard deviation was negligible.

The last experiment is a comparison of our two versions of the ACO algorithm implementation - an innovative distributed version and a standard sequential version. Both versions ran with 500 ants (in the case of the actor-based implementation, they were distributed across two nodes) for 100 iterations. The objective here is to convince the reader that these versions do not differ in the concept of the algorithm itself; i.e., that the distributed version is the standard ACO algorithm only implemented in a different way. For this experiment, we used one of the smaller problems from TSPLIB to be able to easily run it multiple times. Fig. 7 presents the best solution found thus far by both algorithms versus the iteration number with a standard deviation across 20 repetitions. It is noteworthy that subsequent runs of even the sequential version are not the same due to the intrinsic randomness of the ACO algorithm.

Both versions of the algorithm present a higher variation in the exploration phase of the algorithm since the early solutions found may strongly differ in their costs. However, the variation falls with consecutive iterations. Most importantly, both versions behave in a similar fashion, which serves as empirical evidence for the identity of the implementations in terms of the heuristic algorithm concept.

5. Conclusion

In this paper, a novel approach to the implementation of the standard Ant System has been presented. The proposed system is based on a popular and easy-to-use high-level technology that allows for the very efficient verification of different system designs. Besides, it can be deployed in a distributed HPC environment, helping to provide a good level of scalability. This article fills the gap of single-colony ACO systems that can efficiently leverage HPC equipment. It also enables rapid initial optimization due to its extensive exploration based on numerous ants. The presented framework could also be easily applied to other ACO variants and swarm intelligence algorithms with global knowledge.

An actor model has been applied to Ant System in a very intuitive way, aiming to create a scalable and efficient system. The design is focused on a distribution of the pheromone matrix, which is the main point of data synchronization. The responsibility of handling the operations on the matrix parts is distributed across all nodes in the system, and the values from remote parts are synchronized on demand.

The first experiments have proven that a large number of distributed ants improves the results of the optimization at the beginning of the computation; additionally, it does not influence the quality of the final solution. The rapid optimization can be very useful in a production environment where a fast answer is as

important as the quality of the solution (e.g., real-time traffic optimization). The next set of experiments have shown that the proposed architecture scales very well even up to 30 nodes – however, the scalability of certain runs began to diminish at the end, clearly pointing out the room for improvement in future work. The bigger problem is, the better the scalability. The last figure presents the fact that the behavior of the actor-based system is an equivalent of the standard sequential implementation in terms of the achieved results.

To sum up, the presented architecture has shown that it is possible to run the Ant Colony Optimization algorithm in a distributed environment without relaxing any constraints. The scalability can be used to run a big colony, which improves the exploration and enables rapid optimization to a reasonable fitness level.

In the future, we plan to apply our architecture to other variations of the ACO metaheuristic. We also want to verify the system against other optimization problems (e.g., VRP, PDPTW, or HCP). Moreover, we believe that it is possible to introduce some asynchrony in the operations regarding the pheromone matrix in order to enhance further scalability without a noticeable impact on the quality of the solutions. The application of this system design to other swarm intelligence algorithms (or more generally to algorithms with global knowledge) is another path for future research on this topic. Our work might also become a basis for conducting research related to image processing using the swarm intelligence approach; see, e.g., [24,35].

Declaration of Competing Interest

The authors whose names are listed immediately below certify that they have NO affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational grants; participation in speakers' bureaus; membership, employment, consultancies, stock ownership, or other equity interest; and expert testimony or patent-licensing arrangements), or non-financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.

Acknowledgment

The research presented in this paper was supported by the Polish Ministry of Science and Technology funds assigned to AGH University of Science and Technology. The authors utilized the PLGrid infrastructure when conducting the experiments described in this work.

References

[1] S. Abar, G.K. Theodoropoulos, P. Lemarinier, G.M. O'Hare, Agent based modelling and simulation tools: a review of the state-of-art software, Comput. Sci. Rev. 24 (2017) 13–33.
[2] G.A. Agha, Actors: A model of concurrent computation in distributed systems., Technical Report, Massachusetts Inst of Tech Cambridge Artificial Intelligence Lab, 1985.
[3] J. Allen, Effective Akka, O'Reilly Media, 2013.
[4] B. Bullnheimer, R.F. Hartl, A New Rank Based Version of the Ant System - A Computational Study, Cent. Eur. J. Oper. Res. 7 (1) (1999) 25–38.
[5] A. Byrski, R. Drezewski, L. Siwik, M. Kisiel-Dorohinicki, Evolutionary multi-agent systems, Knowl. Eng. Rev. 30 (2) (2015) 171–186, doi:10.1017/S0269888914000289.
[6] E. Cantú-Paz, A survey of parallel genetic algorithms, Calcul. Parall. Reseaux Syst. Repart. 10 (2) (1998) 141–171.
[7] J.M. Cecilia, J.M. García, A. Nisbet, M. Amos, M. Ujaldón, Enhancing data parallelism for ant colony optimization on GPUS, J. Parallel Distrib. Comput. 73 (1) (2013) 42–51.
[8] N. Collier, M. North, Repast HPC: a platform for large-scale agentbased modeling, Large-Scale Computing Techniques for Complex System Simulations (2011) 81–110.
[9] M. Dorigo, Optimization, learning and natural algorithms, PhD Thesis, Politecnico di Milano (1992).
[10] M. Dorigo, G. Di Caro, Ant colony optimization: a new meta-heuristic, in: Proceedings of the 1999 Congress on Evolutionary Computation, 1999. CEC 99, 2, IEEE, 1999, pp. 1470–1477.
[11] M. Dorigo, L.M. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem, IEEE Trans. Evol. Comput. 1 (1) (1997) 53–66.
[12] M. Dorigo, V. Maniezzo, A. Colorni, Ant system: optimization by a colony of cooperating agents, IEEE Trans. Syst. Man Cybern. Part B Cybern. 26 (1) (1996) 29–41.
[13] P. Haller, M. Odersky, Scala actors: unifying thread-based and event-based programming, Theor. Comput. Sci. 410 (2–3) (2009) 202–220.
[14] C. Hewitt, P. Bishop, R. Steiger, Session 8 formalisms for artificial intelligence a universal modular actor formalism for artificial intelligence, in: Advance Papers of the Conference, 3, Stanford Research Institute, 1973, p. 235.
[15] S. Ilie, C. Bădică, Multi-agent approach to distributed ant colony optimization, Sci. Comput. Program. 78 (6) (2013) 762–774.
[16] M. Kiran, M. Bicak, S. Maleki-Dizaji, M. Holcombe, Flame: a platform for high performance computing of complex systems, applied for three case studies, Acta Phys. Polon. Ser. B Proc. Suppl. 4 (2) (2011).
[17] M. Kiran, P. Richmond, M. Holcombe, L.S. Chin, D. Worth, C. Greenough, Flame: simulating large populations of agents on parallel hardware architectures, in: Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1, International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 1633–1636.
[18] D. Krzywicki, W. Turek, A. Byrski, M. Kisiel-Dorohinicki, Massively concurrent agent-based evolutionary computing, J, Comput. Sci. 11 (Supplement C) (2015) 153–162, doi:10.1016/j.jocs.2015.07.003.
[19] R. Kumar, D. Sharma, A. Sadu, A hybrid multi-agent based particle swarm optimization algorithm for economic power dispatch, Int. J. Electr. Power Energy Syst. 33 (1) (2011) 115–123.
[20] P. Leitao, U. Inden, C. Rückemann, Parallelising multi agent systems for high performance computing, in: Proceedings of the Third International Conference on Advances Communications and Computation, INFOCOMP 2013, 2013.
[21] C. Leung, T. Wong, K.-L. Mak, R.Y. Fung, Integrated process planning and scheduling by an agent-based ant colony optimization, Comput. Indust. Eng. 59 (1) (2010) 166–180.
[22] G.-H. Luo, S.-K. Huang, Y.-S. Chang, S.-M. Yuan, A parallel bees algorithm implementation on GPU, J. Syst. Architect. 60 (3) (2014) 271–279.
[23] Q. Lv, X. Xia, P. Qian, A parallel ACO approach based on one pheromone matrix, in: Proceedings of the International Workshop on Ant Colony Optimization and Swarm Intelligence, Springer, 2006, pp. 332–339.
[24] N. Nayar, S. Ahuja, S. Jain, Swarm intelligence for feature selection: a review of literature and reflection on future challenges, in: M.L. Kolhe, M.C. Trivedi, S. Tiwari, V.K. Singh (Eds.), Proceedings of the Advances in Data and Information Sciences, Springer Singapore, Singapore, 2019, pp. 211–221.
[25] M. Pedemonte, S. Nesmachnow, H. Cancela, A survey on parallel ant colony optimization, Appl. Soft Comput. 11 (8) (2011) 5181–5197.
[26] G. Skiba, M. Starzec, A. Byrski, K. Rycerz, M. Kisiel-Dorohinicki, W. Turek, D. Krzywicki, T. Lenaerts, J.C. Burguillo, Flexible asynchronous simulation of iterated prisoner's dilemma based on actor model, Simul. Model. Pract. Theory 83 (2018) 75–92.
[27] R. Skinderowicz, The GPU-based parallel ant colony system, J. Parall. Distrib. Comput. 98 (2016) 48–60.
[28] K. Sörensen, Metaheuristicsthe metaphor exposed, Int. Trans. Oper. Res. 22 (1) (2015) 3–18, doi:10.1111/itor.12001.
[29] T. Stützle, H.H. Hoos, Max–min ant system, Future Gen. Comput. Syst. 16 (8) (2000) 889–914.
[30] W. Turek, Erlang-based desynchronized urban traffic simulation for high-performance computing systems, Future Gen. Comput. Syst. 79 (2018) 645–652, doi:10.1016/j.future.2017.06.003.
[31] W. Turek, L. Siwik, A. Byrski, Leveraging rapid simulation and analysis of large urban road systems on HPC, Transp. Res. Part C Emerg. Technol. 87 (2018) 46–57, doi:10.1016/j.trc.2017.12.014.
[32] W. Turek, L. Siwik, M. Kisiel-Dorohinicki, S. Lakomy, P. Kala, A. Byrski, Real-time metaheuristic-based urban crossroad management with multi-variant planning, J. Comput. Sci. 23 (2017) 240–248, doi:10.1016/j.jocs.2017.04.017.
[33] W. Turek, J. Stypka, D. Krzywicki, P. Anielski, K. Pietak, A. Byrski, M. Kisiel-Dorohinicki, Highly scalable Erlang framework for agent-based metaheuristic computing, J. Comput. Sci. 17 (Part 1) (2016) 234–248, doi:10.1016/j.jocs.2016.03.003.
[34] D. Wolpert, W. Macready, No free lunch theorems for optimization, IEEE Trans. Evol. Comput. 67 (1) (1997).
[35] Y. Feng, Z. Wang, Ant Colony Optimization for Image Segmentation, in: Ant Colony Optimization, IntechOpen, Rijeka (2011) https://doi.org/10.5772/14269.
[36] Y. Zhou, F. He, N. Hou, Y. Qiu, Parallel ant colony optimization on multi-core SIMD cpus, Future Gen. Comput. Syst. 79 (2018) 473–487.
[37] Y. Zhou, F. He, Y. Qiu, Dynamic strategy based parallel ant colony optimization on GPUS for TSPS, Sci. China Inf. Sci. 60 (6) (2017) 68102, doi:10.1007/s11432-015-0594-2.

Chapter 7

Distributed ant system for difficult transport problems

The following publication extends the actor model based ACO by proposing an efficient way of running large-scale ant colony. It focuses on the distribution of the pheromone matrix across the computational nodes, resulting in linear scalability to 50 computational nodes. The second part of the article describes experiments aiming at proper utilization of numerous feasible solutions in order to update the pheromone trails at the end of each iteration. The proposed modification was tested on a few TSP instances from TSPLIB and it achieved better solutions quality than the well-known Max-Min Ant System algorithm. The system was also applied to solving VRPTW instances – these experiments showed that increasing ants count improves solutions quality.

Contributions of Mateusz Starzec: In this paper, I was responsible for introducing the distributed pheromone matrix into the system model. I have prepared tests of the system scalability. I have co-created mechanisms of proper utilization of numerous feasible solutions in the pheromone trails updates. I was the primary author of the following sections *High scalability of ACO*, *System scalability*, *Fast optimization with numerous ants* and co-author of *Introduction*, *Conclusions* and *Comparison with MMAS on TSP problem*.

Journal of Intelligent & Fuzzy Systems 37 (2019) 7347–7356
DOI:10.3233/JIFS-179344
IOS Press

Distributed ant system for difficult transport problems

Mateusz Starzec, Grażyna Starzec, Aleksander Byrski*, Wojciech Turek and Marek Kisiel-Dorohinicki
AGH University of Science and Technology, Krakow, Poland

Abstract. Solving difficult, usually NP-hard problems, requires metaheuristic-based approach. Such algorithms are very often demanding from the point of view of computational power. Therefore various approaches to parallelize or distribute such systems were made. Many of such algorithms are structurally very easy to parallelize, e.g. evolutionary ones. However, swarm computing algorithms, in particular ACO (Ant Colony Optimization), in order to be implemented properly must use a significant amount of global knowledge (pheromones matrix). Therefore strict parallelization/distribution strategies for ACO are difficult to work-out. In the presented paper we propose a novel approach for parallelization and distribution of the most important element of ACO, namely the pheromone table. Our prototype implementation is tested on a real-world HPC (High Performance Computing) infrastructure, with good observed scalability. At the end of this paper we present actual experimental results focusing on two class of problems, namely TSP (Travelling Salesman Problem) and VRPTW (Vehicle Routing Problem with Time Windows), using popular benchmarks.

Keywords: parallel and distributed computing, ant colony optimization, swarm intelligence, high performance computing

1. Introduction

Transport-related optimization problems are often characterized by a high complexity which makes them practically impossible or inviable to solve accurately. Metaheuristic algorithms are designed to handle such problems in an efficient way – providing a reasonable solution while only requiring a specific representation of the problem data, e.g. graph for Ant Colony Optimization. In case of well-known problems there are plenty of problem specific optimizations improving the performance of the metaheuristics. Nonetheless the efficiency of the algorithm itself is also important, especially when there are no problem specific optimizations.

Modern computation hardware, especially big homogeneous or heterogeneous clusters, encourage the researchers to consider parallelization of algorithms. The time-limited solving of an optimization problem requires efficient utilization of these resources (e.g., in the case of traffic optimization [12]). The parallelization of evolutionary algorithms has been quite well-explored. These can be implemented in multiple models, like master-slave, cellular, and island based [2], which scale well on the existing hardware.

The Ant Colony Optimization [3] involves creation of multiple solutions in each iteration. That makes it a great candidate to be parallelized, by distributing the process of solutions creation across multiple processors [8] or even computation nodes [7]. The high-scalability enables creation of higher number of solutions in the same time, which may increase exploration of the solutions space and finally lead to better solutions.

The scalability of the algorithm is limited by sharing the global knowledge represented as the pheromone matrix. In case of big problems the matrix

*Corresponding author. Aleksander Byrski, AGH University of Science and Technology, Al. Mickiewicza 30, 30-059 Krakow, Poland. E-mail: olekb@agh.edu.pl.

ISSN 1064-1246/19/$35.00

can be distributed over the computation nodes in order to parallelize the operations of the matrix update at the end of each iteration. To the best of our knowledge, however, there are no computing systems that are capable of running ACO-computing efficiently in large-scale distributed systems (i.e., HPC-grade).

In this paper we would like to apply the ACO with the distributed pheromone matrix, involving more ants than the standard approaches, to hard optimization problem. We also present basic scalability tests. The bottom line is that presented algorithm outperforms classic MMAS (Max-Min Ant System) algorithm.

In the next section we present state of the art related to the investigated research area. Section 3 describes the distributed ACO algorithm, involving the distribution of ants and the pheromone matrix. Section 4 lists conducted experiments and thoroughly discusses obtained results. Section 5 summarizes the article.

2. From classic to parallel ACO

The first version of Ant Colony Optimization was introduced in [3] to solve TSP. The algorithm was inspired by the behavior of the natural ants colonies. ACO as a meta-heuristic algorithm was described in [4].

The ACO meta-heuristic algorithm expects the optimization problem to be considered as a graph consisting of finite set of *components* connected by edges with assigned cost. A valid solution is a path that respects posed restrictions and fulfills requirements defined by the problem. A solution cost is defined as a function of all the costs of all the connections belonging to the solution. The optimal solution is the path with minimum cost. E.g. in case of TSP solution's cost is a total distance or in case of VRP (Vehicle Routing Problem) it is a number of vehicles and a sum of the routes distance, where the number of vehicles is more important.

The optimization process is based on a population of ants. They iteratively traverse the graph creating the candidate solutions. A probabilistic decision rule (see Eq. 1) is a basis of the ant's decision regarding selection of the next selected vertex. In each iteration each ant starts from an initial vertex, which can be selected randomly as in the TSP or can be specified by the problem liek a depot node in the VRP. The probabilistic decision rule takes into account the values of pheromone trails left on the edges by the previous generations of ants.

In the basic ACO algorithm - Ant System (AS, [6]) the probability of moving from component i to component j for ant k in iteration t is defined as follows:

$$p_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}(t)]^{\beta}}{\sum_{k \in allowed_k} [\tau_{ik}(t)]^{\alpha} \cdot [\eta_{ik}(t)]^{\beta}} & \text{if } j \in allowed_k \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $\tau_{ij}(t)$ is the intensity of a pheromones trail on the edge and $\eta_{ij}(t)$ is the visibility (also referenced as desirability) of the edge, which can be defined as an inversed distance from i to j in case of the TSP. The parameters that control the relative importance of trail versus visibility are α and β. Finally, $allowed_k$ is a set of possible transitions for the ant k which has regard to the already constructed path and the problem restrictions. The tour ends when a feasible solution is found. Based on the constructed paths, the ants update the pheromone trails. In the classic AS version the update is performed at the end of a single iteration, according to the formula:

$$\tau_{ij}(t+1) = \rho \tau_{ij}(t) + \sum_{k=1}^{m} \Delta \tau_{ij}^{k} \tag{2}$$

where $\rho \in [0, 1)$ is a pheromone persistence coefficient and m is the number of the ants. The pheromone update value for each ant is defined as follows:

$$\Delta \tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & \text{if } k\text{-th ant uses edge } (i, j) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where L_k is the tour length of the k-th ant and Q is a constant, often with value 1.

Since the first publication describing Ant System [3] researchers have put a lot of effort into improving the optimization performance. One of the most efficient modifications is Max-Min Ant System (MMAS, [10]). The algorithm introduced three major adjustments into the AS algorithm:

- only one ant leaves the pheromone trails after each iteration - either the best from the iteration or the best found so far by the algorithm,
- the pheromone trials values are limited by τ_{min} and τ_{max},
- the pheromone matrix is initialized with the τ_{max} value.

The effectiveness of the MMAS algorithm was further improved by the *pheromone trail smoothing* mechanism. When the algorithm is very close to con-

vergence, the pheromone matrix gets smoothed by increasing the values proportionally to their difference to τ_{max}. This mechanism can also be effectively applied to other elitist ant systems [1, 6].

Besides the standard modifications of the Ant System, e.g., Elitist Ant System (EAS), Rank-based Ant System (ASRank, [1]), Max-Min Ant System (MMAS, [10]) or Ant Colony System (ACS, [5]), which focused on introducing modifications to the original sequential algorithm, the researchers tried to additionally boost the performance by leveraging parallelization of the algorithm. A taxonomy proposed in [9] classifies the numerous parallel algorithms into four main categories based on two primary criteria: the number of colonies (one or many) and the cooperation or lack thereof among the parallel parts:

- *Master-slave model* – a single colony, where the master process manages the global information,
- *Cellular model* – a single colony with cooperation, where the colony is divided into small overlapping neighborhoods with their own pheromone matrices,
- *Parallel independent runs model* – several sequential ACOs are independently executed on a set of processors using identical or different meta-parameters,
- *Multi-colony model* – several colonies explore the solution space with their own pheromone matrices but periodically exchange information.

There are algorithms, which do not fit any of the proposed group. The architecture proposed in this paper is close to the *Master-slave model*, but the global knowledge is also distributed.

There are only a few articles regarding implementation of a single ants colony in a big cluster environment, which could allow not only for improving the computations speed but also leveraging extended exploration by a higher number of ants in a single colony. The most common use-case for clusters is to run multiple colonies. Ilie and Bădică et al. [7] presented an agent-based approach to ACO modeling. This article reports a very good scalability up to 7 nodes on the *gr666* problem from TSPLIB[1]. The algorithm's efficiacy is comparable to the standard Ant System. Unfortunately, the authors did not report any experiments with large-scale TSP problems or larger clusters.

[1] https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

3. Distributed Ant System

The majority of research is focused on ACO modifications in order to solve complex problems faster, usually with a cost of obtaining suboptimal solution. In this paper we propose a scalable, distributed implementation of ACO. The highly scalable architecture is the basis of the efficient and distributed algorithm involving numerous ants. Nonetheless, the problem representation and setting the meta-parameters of this algorithm is also important. This section describes the architecture of the system and the proposed way to setup the parameters, with focus on leveraging the knowledge collected by the numerous ants to improve the results obtained from the algorithm.

3.1. High scalability of ACO

Achieving highly scalable ACO architecture requires distribution of many processes and components and assuring low inter-process (also inter-node) communication overhead. The first step is a distribution of the ants (representing the processes of a single solution creation), which is a standard master-slave approach to this problem (see Fig. 1). The architecture enables parallel creation of the new solutions, however, it is limited by the performance of the master node, which has to provide the pheromones data to all other nodes. Moreover, the process of the pheromones update at the end of each iteration is performed only on the single node, while other resources are waiting for the results.

The distributed pheromone matrix (see Fig. 2) reduces influence of these limits. Each computation node owns a part of the matrix and has the following responsibilities:

- providing the pheromone values to all other nodes (see Fig. 3),
- caching values from the remote parts of the matrix and providing them to the local ants (see Fig. 3),
- collecting the solutions from the local ants and broadcasting them to all other nodes,
- applying the pheromone updates on its part of the matrix.

The pheromone matrix data is uniformly distributed across all the system nodes, which enables distribution of the matrix related operations and improves utilization of the computing resources. Each node runs numerous ants, so the values loaded from the remote parts are cached, which greatly reduces the network communication (a single value

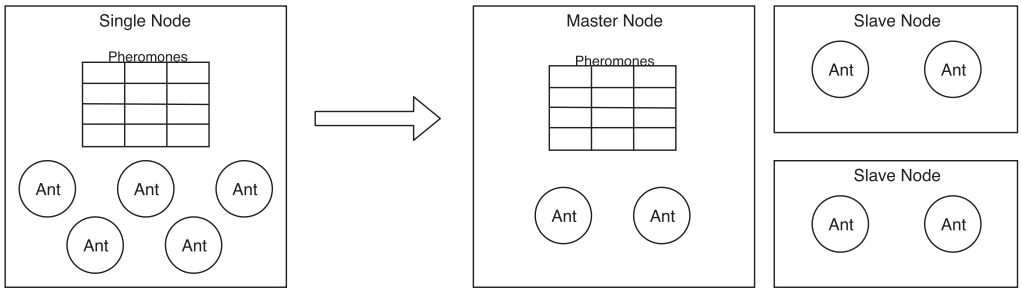


Fig. 1. Master-slave model of the distributed ACO.

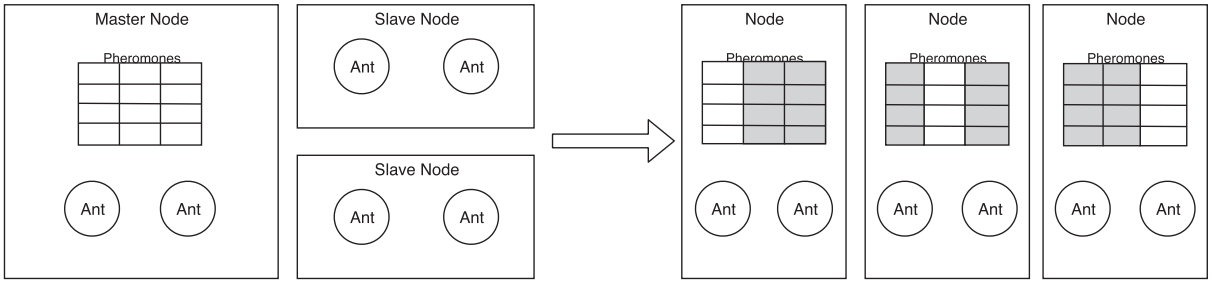


Fig. 2. ACO with distributed pheromone matrix.

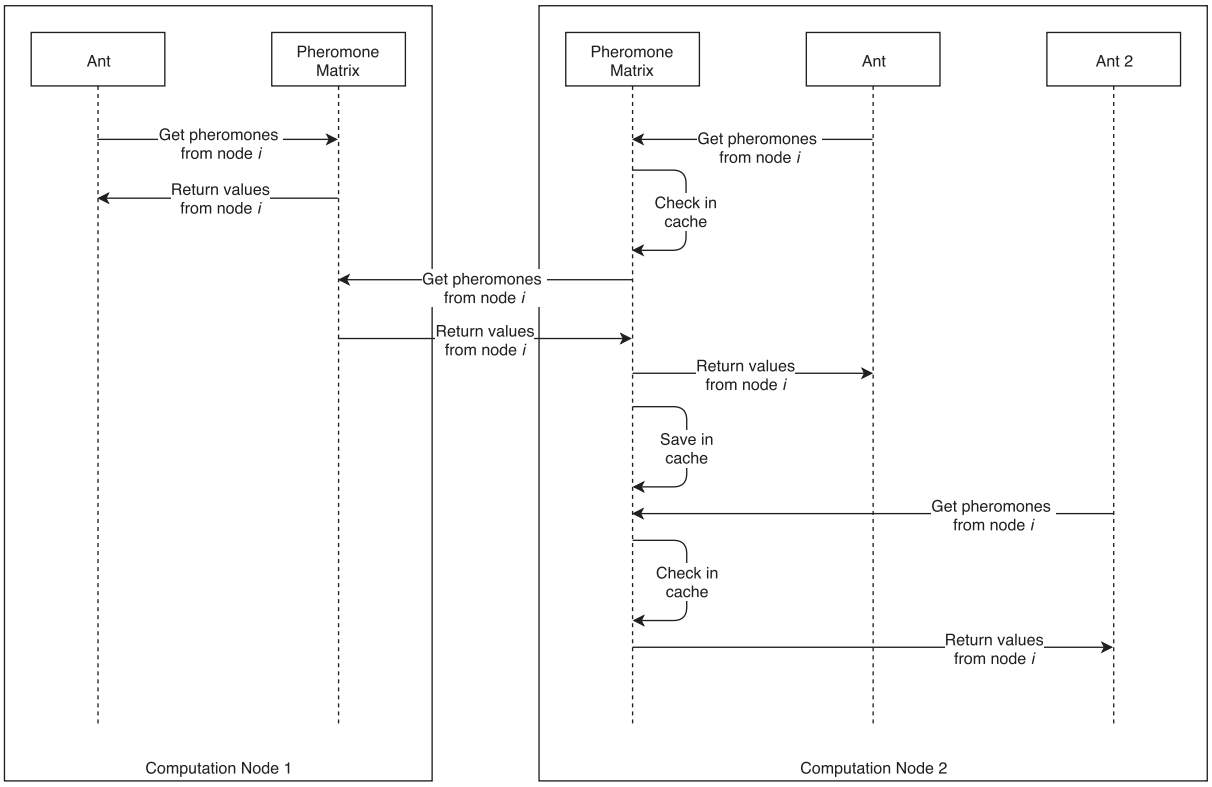


Fig. 3. Providing pheromone values to the ants on different nodes. Computation node 1 contains pheromone data regarding node i.

from the matrix can be loaded only once per iteration). The remote values are loaded on-demand, which enables parallel data synchronization and solutions constructing. Also, in case of problems which do not require visiting all of the graph nodes (such as the discrete knapsack problem), the on-demand loading does not synchronize unnecessary parts of the data.

In order to further reduce the network communication, the loading of the remote data does not ask for a single cell from the pheromone matrix, instead it loads pheromones of all outgoing edges from the given node in the problem graph, since these values are required by the ant's probabilistic decision rule (see Eq. 1).

The architecture do not change any constraints of the sequential Ant System implementation. Each ant creates exactly one solution in each iteration, then it starts creation of the new solution, but blocks on waiting for the pheromone values for the next iteration, until the other ants finish their job in the current iteration and all nodes update their parts of the matrix.

3.2. Parameters of the algorithm and their relations

This subsection describes parameters of the proposed algorithm and relations between their values. The parameters are similar to the ideas from the standard ACO variations, the proposed configuration modifies their relations and default values in order to efficiently use the knowledge collected by the numerous ants.

The configuration of the algorithm includes:

- N - a number of ants in the system.
- ρ - a pheromone persistence coefficient.
- Q - a pheromone update unit.
- α - importance of the pheromones.
- β - importance of desirability.
- IBU - iteration best updates; a number of the best ants from the iteration updating pheromones after each iteration, their update is not weighted by the ranking position.
- GBU - a multiplier for the pheromones left on the global best solution after each iteration.
- $\tau(0)$ - global best updates; an initial amount of the pheromones on every edge, usually it is equal to the τ_{Max} value.
- τ_{Max} - a maximum value of the pheromone on an edge.
- τ_{Min} - a minimum value of the pheromone on an edge.

In a standard implementation of ACO selection of values for the parameters α and β depends on the desirability values from a problem representation and the $\tau_{Max/Min}$ parameters (if they are used). If desirability values and τ values are normalized and fit $[0, 1)$ bounds, the choice of α and β values is less dependent on the problem instance.

A similar situation occurs with the Q parameter. The range of usable values of this parameter depends on the fitness values of the problem solutions and the ρ and N parameters. The order of magnitude of values $(IBU + GBU)\frac{Q}{Fitness}$ and $\tau_{Max}(1 - \rho)$ should be similar, as this allows to keep the current best solutions close to τ_{Max}. To make Q independent from the fitness values of the problem solutions, the fitness value is normalized by the value of the current global best solution. The global best solution has fitness equal to 1 and hence the fitness of any solution can change during the computations. The standard cost value is still used to report the solutions and track the global best solution. With this assumption, the value of parameter Q can be calculated as follows:

$$Q = X\frac{\tau_{Max}(1 - \rho)}{IBU + GBU} \tag{4}$$

where X is a constant with a value around 1 - the equation can optionally ignore IBU or GBU values.

3.3. Problem representation

The algorithm uses the standard representation of TSP for use in ACO algorithms: the problem nodes are the cities to visit, all nodes are connected and the cost of each edge is defined as a distance between cities. The solution needs to include all nodes exactly once and the cost is a sum of costs of all included edges. The desirability of an edge is an inverse of its cost [3].

The representation of VRPTW includes one special node which is the depot node. The rest of the nodes represent clients to be served. Each client node has an assigned order size (related to the cars capacity) and a time window when the client is to be visited. All nodes are connected. A solution starts from the depot node and can come back to this node multiple times – every return is considered as a new vehicle selection. The route of each vehicle in the solution starts at time 0, the time of travel between nodes is equal to the distance between them. The sum of clients' orders assigned to a single vehicle cannot exceed the globally defined vehicle capacity. The cost of the solution is a number of vehicles used by

the solution, in case of equal number of vehicles the traveled distance is used to select the better solution. The desirability of node is an average of inverse of distance to travel and inverse of the end of target's time window. These values are normalized respectively by the maximum distance between nodes and the maximum time window end time [11].

4. Experiments

This section describes experiments aiming to prove that the presented algorithm can outperform the standard ACO implementations. The conducted experiments show the scalability of the system and compare different configurations performance in terms of the quality of the solution.

In order to verify the proposed algorithm of distribution of pheromone table, a dedicated computing system was implemented based on Scala and Akka technologies.

The experiments were conducted on the Prometheus supercomputer—a peta-scale (2.4 PFlops) cluster located in the Academic Computer Center Cyfronet AGH in Krakow, Poland. As of November 2018, Prometheus is ranked on the 131st position on the TOP500 fastest supercomputer list. Prometheus is a cluster of the HP Apollo 8000 nodes, each with 24 Xeon E5-2680v3 CPUs working at 2.5GHz frequency, connected via InfiniBand FDR network.

4.1. System scalability

The scalability of the system is not our main point in this paper, however, it is the basis of reasonableness of running numerous ants in a single iteration. In this section, we present basic tests of the scalability, which we will extend in the future.

Table 1 and Table 2 show a single iteration duration on various number of computing nodes with 25 ants simulated on each of them. The former presents results for a problem from TSPLIB: *pr2392*, the latter uses *RC1_10_1* from Gehring & Homberger benchmark. Every test case was repeated 10 times.

When the problem exceeds particular size, additional computing nodes reduce duration of a single iteration, which is a result of the distribution of updating the pheromones matrix. In case of a big matrix for the *pr2392* problem, the system scale very well up to 50 nodes. The VRPTW instance requires a much smaller matrix, so the system achieved good scalability up to 30 nodes. When the number of nodes is too high, the performance breaks down due to high impact of the inter-node communication.

4.2. Comparison with MMAS on TSP problem

In this section we will compare the effectiveness of the sequential Max-Min Ant System, Distributed Ant System (distributed Ant System without improvements regarding problem representation and parameters setup) and Enhanced Distributed Ant System (including all improvements described in the previous section).

The MMAS algorithm [10] is one of the most efficient sequential extensions of the Ant System. The Table 3 presents comparison of the results obtained by the reference MMAS implementation[2] with 25/250 ants, Distributed Ant System with 1k/10k ants and Enhanced Distributed Ant System with 250/500 ants. None of the algorithms used any local optimization mechanisms nor TSP specific assumptions and each tests case was repeated 10 times.

Distributed Ant System was running 400 iterations for both problems, Enhanced Distributed Ant System had 1000 iterations and MMAS had 2500 iterations.

The algorithms configuration:

- MMAS: $\alpha = 3$, $\beta = 5$, $\rho = 0.95$.
- DAS: $\alpha = 2$, $\beta = 3$, $\tau(0) = 0.01$, $\rho = 0.99$, $Q = 1$.
- EDAS with 250 ants: $\alpha = 3$, $\beta = 5$, $\tau_{Max} = 0.9999$, $\tau_{Min} = 0.0001$, $\tau(0) = 0.9999$, $\rho = 0.95$, $Q = 0.0005$, $IBU = 100$, $GBU = 100$.
- EDAS with 500 ants: $\alpha = 3$, $\beta = 5$, $\tau_{Max} = 0.9999$, $\tau_{Min} = 0.0001$, $\tau(0) = 0.9999$, $\rho = 0.95$, $Q = 0.0002$, $IBU = 250$, $GBU = 250$.

The results show that increasing the number of ants in case of the standard MMAS does not improve the solutions - in case of *u1432* the result of 25 ants was better. MMAS allows only one ant to put the pheromones after each iteration, so the knowledge from the extended exploration is wasted.

In both cases the DAS and EDAS algorithms significantly outperform the MMAS implementation. The DAS achieved better results than EDAS in case of *u1432* and similar in case of *pr2392*, but

[2] http://iridia.ulb.ac.be/~mdorigo/ACO/downloads/ACOTSP-1.03.tgz - the code was modified to remove optimizations specific to TSP: symmetric pheromones matrix and reduction of neighbourhood to the nearest neighbours. The goal is to measure performance of the meta-heuristic algorithm itself without any problem-specific improvements.

Table 1
Scalability on *pr2392* from TSPLIB

Nodes	1	2	5	10	15	20	30	50	75	100
Iteration time (s)	16.77	15.74	12.65	12.24	11.63	11.63	12.04	13.9	24.77	39.93

Table 2
Scalability on RC1_10_1 from Gehring & Homberger benchmark

Nodes	1	2	5	10	15	20	30	50	75	100
Iteration time (s)	2.7	2.65	2.56	2.94	2.97	3.09	3.5	5.55	11.09	18.95

Table 3
Comparison of results achieved by MMAS, Distributed Ant System and Enhanced Distributed Ant System on TSP instances from TSPLIB

Problem	*u1432*	*pr2392*
Best known solution	152970	378032
MMAS (25 ants)	184575 ± 531.37	494735.2 ± 2983.4
MMAS (250 ants)	188164 ± 548.13	489201.8 ± 1481.6
Distibuted Ant System (1k ants)	176835.3 ± 1760.7	471657.8 ± 3201.2
Distibuted Ant System (10k ants)	173164.5 ± 1152.1	454735 ± 1058.9
Enhanced Distributed Ant System (250 ants)	179350.1 ± 2522.2	455035.2 ± 5446.7
Enhanced Distributed Ant System (500 ants)	177731 ± 2102.2	452700 ± 5998.7

Table 4
Comparison of results of Enhanced Distributed Ant System on VRPTW instances from Gehring & Homberger benchmarks. Best/average/worst result

Problem	C1_10_1	R1_10_1	RC1_10_1	C2_10_1	R2_10_1	RC2_10_1
Best known solution	100	100	90	30	19	20
25 ants - (GBU: 0 / IBU: 1)	131/133.8/136	100/101.25/102	97/98.4/101	63/64.8/66	36/39.4/40	42/42.4/43
25 ants - (GBU: 10 / IBU: 10)	141/143/145	101/101.4/102	97/98.8/101	68/71/73	41/41.4/42	43/45.5/48
125 ants - (GBU: 100 / IBU: 100)	136/136.8/138	101/101/101	97/99.2/102	66/68/70	38/40.2/42	41/43/45
250 ants - (GBU: 100 / IBU: 100)	135/137/138	100/101/102	93/**95**/96	65/66.8/69	36/39.6/42	40/42.4/44
500 ants - (GBU: 100 / IBU: 100)	132/134.8/138	100/100.4/101	95/96/97	63/64.8/66	36/**38.2**/41	40/42.4/44
500 ants - (GBU: 250 / IBU: 250)	133/135.6/138	100/100.8/101	93/95.2/97	65/66/67	37/39.6/41	41/42.2/43
500 ants - (GBU: 500 / IBU: 500)	134/136.4/138	100/100.7/101	96/96.6/98	64/66.6/68	38/39.6/41	42/42.8/43
1000 ants - (GBU: 100 / IBU: 100)	130/133/136	100/**100.2**/101	94/**95**/96	63/**64.6**/66	36/**38.2**/40	42/**42**/42
1000 ants - (GBU: 250 / IBU: 250)	129/**131.7**/133	100/100.6/102	94/95.4/96	64/65.8/69	38/38.6/40	41/42.2/43

EDAS requires much smaller number of ants, so the computation time was around 10 times smaller in case of EDAS with the same number of computing nodes.

The experimental results show that the additional exploration conducted by the numerous ants makes significant difference when the collected knowledge is properly utilized.

4.3. Ants count influence

EDAS outperforms MMAS, because it extends exploration by numerous ants and efficiently uses the knowledge collected by these ants. This sections covers the topic of selecting number of ants and the values of IBU and GBU parameters. Table 4 contains results of various VRPTW calculations with different parameters configurations. The result contains only the number of vehicles, for every test case we present best/average/worst result of 5 repeats. The configuration parameters were: $\alpha = 3$, $\beta = 5$, $\tau_{Max} = 0.9999$, $\tau_{Min} = 0.0001$, $\tau(0) = 0.9999$, $\rho = 0.95$, $Q = \frac{\tau_{Max}(1-\rho)}{IBU}$.

The first part of a problem name describes the problem type:

- C / R / RC - clustered / random / mixed problem.
- 1 - short planning time horizon - shorter routes, more vehicles.
- 2 - long planning time horizon - longer routes, fewer vehicles.

The results show that the number of ants itself without any changes of GBU and IBU has an influence on the solutions - the more ants the better solution is. The GBU and IBU parameters changes seem to be also important - the best results were yielded when

less than a half of the best solutions were strengthen in the pheromone matrix.

4.4. Fast optimization with numerous ants

The extended exploration arisen from the high number of ants leads to better solutions at the beginning of computations. It can be useful in dynamic optimization tasks, where rapid optimization is required.

Figure 4 and Figure 5 show that there is an upper limit of profitability of higher number of ants for each problem. The bigger the problem is, the higher is the limit. In case of VRPTW with 1000 nodes there is

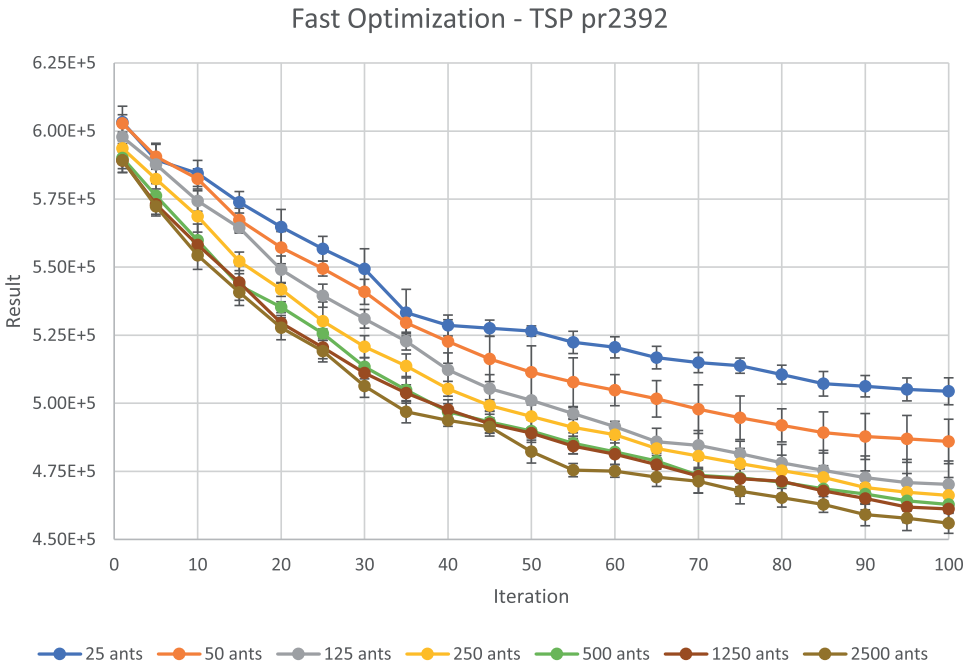


Fig. 4. Best solution over iterations with different ants number; Problem TSPLIB: pr2392.

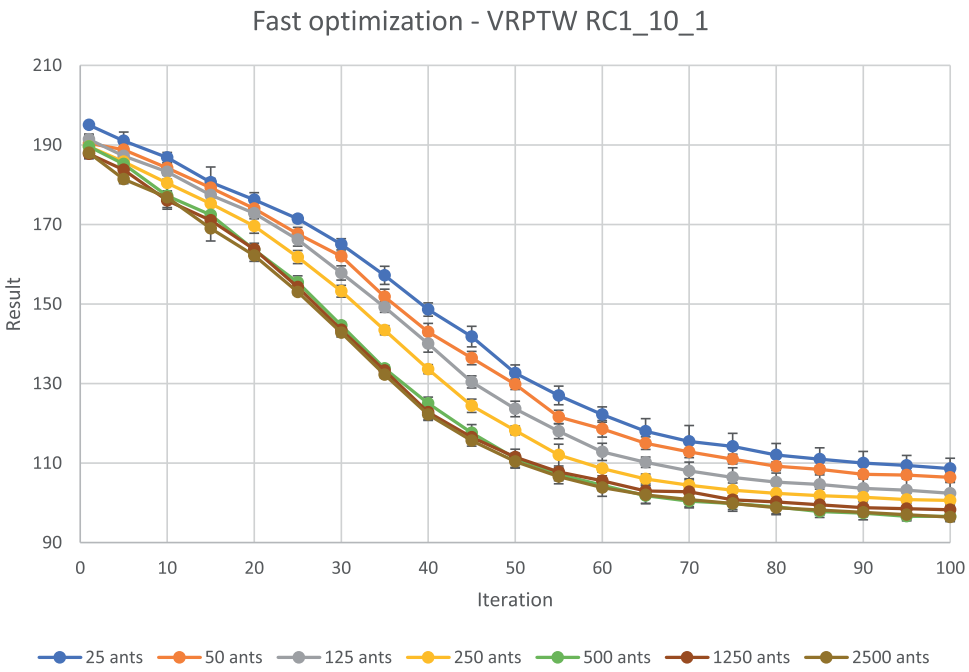


Fig. 5. Best solution over iterations with different ants number; Problem VRPTW: RC1_10_1.

no gain from increasing ants number further than 500 ants, but for TSP with 2392 nodes even 2500 ants configuration provides better results than the less numerous configurations.

4.5. Summary

The experiments show that the higher number of ants leads to better optimization results when the additional knowledge is used properly. The scalability of the algorithm enables usage of big clusters in order to obtain better results without noticeable computation time elongation. The distributed system architecture is able to handle big optimization problems.

5. Conclusions

In this paper a design of ACO with distributed pheromone matrix has been presented. The idea is based on the well-known *master-slave* model. The distribution of the matrix enables high scalability and effective running of numerous ants. Besides scalability, we have focused on leveraging the knowledge collected in the system. The proposals were inspired by the well known sequential Ant System extensions and adapted to the numerous ants in the simulated Ant System. The problem representation normalization has been proposed, which enabled defining sensible values of the algorithm parameters, independently from the problem instance specific values like route length in TSP.

The algorithm compares well with Max-Min Ant System when basing on the two TSP problems from TSPLIB. The bigger a problem instance is the more prominent is the difference. The experiments based on VRPTW showed that the higher number of ants in the system improves the results. This property combined with the high scalability enables high potential of improving the efficiency of the raw metaheuristic (without local and problem specific optimizations) without rising the computation time. The experiments proved that the high number of ants also improves rapid optimization, due to extensive exploration at the beginning of the computations.

To sum up, the presented metaheuristic algorithm gives a significant improvement when compared to the standard Ant System variations like the Min-Max Ant System. It is a result of efficient usage of the knowledge collected by the numerous ants without extending the computation time thanks to the high scalability.

In order to conduct the experiments in this paper, a prototype computing system was implemented using Scala/Akka technology. The obtained efficiency and scalability results are very promising (and we will devote a dedicated publication to these phenomena) and confirm that this technology should be further leveraged when developing our research.

In the future we plan to improve scalability of the algorithm by desynchronizing the operations related to the pheromones matrix. It should enable efficient deployment of the ants on the higher number of computing nodes and enhance the algorithm's performance on huge problem instances. The application of this system design to other swarm intelligence algorithms (or more generally to the algorithms with global knowledge) is another path of future research in this topic.

Acknowledgments

The research presented in this paper was partially supported by the Polish Ministry of Science and Higher Education funds assigned to AGH University of Science and Technology. The authors acknowledge using of PLGrid infrastructure.

References

[1] B. Bullnheimer, R.F. Hartl and C. Strauss, A new rank based version of the ant system, *A Computational Study* (1997).

[2] E. Cantú-Paz, A survey of parallel genetic algorithms, *Calculateurs Paralleles, Reseaux et Systems Repartis* **10**(2) (1998), 141–171.

[3] D. Marco, *Optimization, learning and natural algorithms. PhD Thesis*, Politecnico di Milano, 1992.

[4] M. Dorigo and G.D. Caro, Ant colony optimization: A new meta-heuristic, In *Evolutionary Computation, 1999. CEC 99 Proceedings of the 1999 Congress on*, volume 2, 1999, pp. 1470–1477. IEEE.

[5] M. Dorigo and L.M. Gambardella, Ant colony system: A cooperative learning approach to the traveling salesman problem, *IEEE Transactions on Evolutionary Computation* **1**(1) (1997), 53–66.

[6] M. Dorigo, V. Maniezzo and A. Colorni, Ant system: Optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **26**(1) (1996), 29–41.

[7] S. Ilie and C. Bădică, Multi-agent approach to distributed ant colony optimization, *Science of Computer Programming* **78** (19966), 762–774.

[8] Q. Lv, X. Xia and P. Qian, A parallel aco approach based on one pheromone matrix, In *InternationalWorkshop on*

Ant Colony Optimization and Swarm Intelligence, Springer, 2006, pp. 332–339.

[9] M. Pedemonte, S. Nesmachnow and H. Cancela, A survey on parallel ant colony optimization, *Applied Soft Computing* **11**(8) (2011), 5181–5197.

[10] T. Stützle and H.H. Hoos, Max–min ant system, *Future Generation Computer Systems* **16**(8) (2000), 889–914.

[11] P. Toth and D. Vigo, *The Vehicle Routing Problem*, Society for Industrial and Applied Mathematics, Philadelphia, 2002.

[12] W. Turek, L. Siwik and A. Byrski, Leveraging rapid simulation and analysis of large urban road systems on hpc, *Transportation Research Part C: Emerging Technologies* **87** (2018), 46–57.

Chapter 8

Desynchronization in distributed Ant Colony Optimization in HPC environment

The third article proposes an application of the desynchronization concept into Ant Colony Optimization algorithm. Based on the previous experimental results, further scalability improvements and bigger colonies may improve obtained results without substantial increase of the computation time. Together with the distributed pheromone matrix, desynchronization enabled scalability up to 400 computational nodes in HPC environment with 76% efficiency. The new algorithm significantly outperformed the previous versions of the algorithm when tested on *pr2392* from TSPLIB. The experiments proved again the positive influence of a numerous ants colony on the optimization results quality. Furthermore, the good scalability allows to use it without noticeable increase of computation times.

Contributions of Mateusz Starzec: I was responsible for redesign and implementation of the system with desynchronization applied to the pheromone matrix. I have prepared the scalability tests of the redesigned system. I was the primary author of *Desynchronization in distributed ant colony optimization* and co-author of *Experimental results* parts related to the system scalability, *Parallel and distributed ant colony computing* and *Conclusion* sections.

Future Generation Computer Systems 109 (2020) 125–133

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Desynchronization in distributed Ant Colony Optimization in HPC environment

Mateusz Starzec, Grażyna Starzec, Aleksander Byrski *, Wojciech Turek, Kamil Piętak

AGH University of Science and Technology, Department of Computer Science, Faculty of Computer Science, Electronics and Telecommunications, Al. Mickiewicza 30, 30-059, Krakow, Poland

ARTICLE INFO

Article history:
Received 14 October 2019
Received in revised form 17 February 2020
Accepted 23 March 2020
Available online 31 March 2020

ABSTRACT

Metaheuristics have significant computing requirements, in particular Ant Colony Optimization (ACO) processes a population of individuals (agents/ants) roaming in a graph, leaving the pheromone trails and getting inspired by its amount perceived on the edges. If the considered problem instance is large or the time is crucial, one can try to leverage parallel, hybrid or distributed infrastructure, but the algorithm itself must be properly prepared to deal with new possibilities. We have already presented a method for efficient implementation of distributed ACO, in this paper we follow up with introducing planned desynchronization in the pheromone matrix updates in order to further increase the scalability of the proposed system. The proposed modifications allowed the algorithm to scale up to 400 computations nodes without a significant impact on results quality. Efficacy of the algorithm outperforms the standard Max–Min Ant System by 10%.

1. Introduction

Search for new metaheuristics for solving difficult problems can be supported by some conclusion of the so-called no free lunch theorem [1] by Wolpert and MacReady. Of course, metaheuristics such as evolutionary algorithms, swarm intelligence methods and similar approaches pose demanding requirements for the infrastructure, as they very often govern a high number of individuals (forming some kind of population), and the search for an optimal solution is connected with processing this population. Therefore parallel metaheuristics were proposed [2], in order to leverage parallel, distributed and hybrid infrastructures, such as multi- and many-core systems, GPGPU, clusters, clouds and supercomputers.

During the design of parallel and distributed computing methods, in particular parallel metaheuristics, one has to deal with synchronization problems. For example, in the case of evolutionary algorithms, the migration should be realized in particular generation (i.e. in particular moment of time). Those are classic problems of communication, thus the processes must sometimes wait for each other, utilizing classic synchronization methods like semaphors or monitors, or novel ones (e.g. actor-based parallelism model).

However, while dealing with metaheuristic computing, one has to realize that those algorithms actually deal with stochasticity. If we want to obtain a feasible result from the stochastic algorithm, we are forced to repeat the experiments and statistically process the outcomes. Thus approximation and uncertainty are present in those algorithms. Therefore one may wonder, what if we took a little more liberal approach to the synchronization itself? What would happen if, e.g. in the parallel evolutionary algorithms, some migration arrived too late? To what extent would that hamper the outcomes of the experiment?

One has to remember a very good aim of such endeavor – certain lack of synchronization could lead us to obtaining very scalable algorithms, capable of running on large infrastructures like supercomputers, able to process and solve very large instances of the problems, and this is the main aim of this paper – to present a desynchronized metaheuristic (namely Ant Colony Optimization) and to test its properties, advocating the feasibility of the produced results. This work follows several previously published articles, both in the domain of computing and simulation [3–6].

When considering Ant Colony Optimization [7], a number of approaches have been made to implement such systems using distributed [8], parallel [9], or even heterogeneous environments [10,11]. To the best of our knowledge, however, there are no computing systems that are capable of efficiently running ACO-computing in large-scale distributed systems (i.e., HPC-grade). Thus we have proposed an easy-to-use, highly scalable and robust way of implementing ACO-computing in a parallel or distributed environment, based on actor-based parallelism that can be supported by Scala/Akka, for example [12]. Moreover,

* Corresponding author.
E-mail addresses: mateusz.starzec@iisg.agh.edu.pl (M. Starzec), grazyna.starzec@iisg.agh.edu.pl (G. Starzec), olekb@agh.edu.pl (A. Byrski), wojciech.turek@agh.edu.pl (W. Turek), kpietak@agh.edu.pl (K. Piętak).

https://doi.org/10.1016/j.future.2020.03.045
0167-739X/

applying a high-level language instead of leveraging previously proven and efficient techniques such as MPI will make the development process easy and not so prone to errors; this is the case when using low-level programming languages.

In the paper [5], we have proposed a novel approach to the implementation of a parallel and distributed ACO by leveraging the actor-based model of concurrency. The proposed computing system is designed to be capable of efficiently leveraging the available HPC infrastructure. The presented version was efficiently run on a cluster of 30 computing nodes. It is noteworthy that the designed system can run thousands of ants, and the implementation is negligibly different from the sequential version (such results were also presented [6]). Following this paper, we would like to present a next version of the constructed distributed ACO, this time focusing on introducing desynchronization into the computing algorithm.

The standard implementation of ACO has to collect all feasible solutions created by the ants and apply pheromone update to the whole pheromone matrix at the end of each iteration. This synchronization point requires a lot of network communication and stops the ants from efficient utilization of available processors. The idea of the desynchronized algorithm is to apply pheromone updates independently using smaller batches of solutions and to allow the ants to read pheromone values between the batch updates. It means that the pheromone values may change during building a single solution by an ant and there is no strict iteration end point. The modifications allow the algorithm to scale up well to hundreds of computing nodes without any significant impact on solutions quality. Moreover, improved scalability allows to run bigger colonies and thereby obtain better solutions for larger problems.

In the next section, the background of the desynchronization in general and very short reference to ACO-methods will be sketched out in the context of parallelization. The proposed technique will also be discussed after positioning the presented paper in the state-of-the-art literature, showing the relationships of the agents and their interactions. Next, our experimental results will be shown, proving the applicability of the proposed approach. This will be followed by our conclusions.

2. Synchronization and desynchronization

In parallel and distributed computing the synchronization feature is omnipresent and can be treated as condition *sine qua non*, regarding the feasibility of the computing system. Solving classic problems, like dining philosophers, readers and writers, producers and consumers became seminal for proposing solutions such as semaphors, monitors and much later sophisticated mechanisms e.g. actor-based model of parallelism [13,14].

Regarding the desynchronization, it is usually treated as a flaw of a system, that can lead to harmful update of the critical resources and is generally not a welcome feature of the parallel and distributed systems. On the other hand, in applications regarding distributed communication, e.g. in wireless networks, the senders and receivers must work in a desynchronized manner so that their transmission time does not overlap [15], thus many (also natural) inspirations are used to introduce this feature in the systems.

When dealing with stochastic algorithms and certain simulations that require high computational power, we are very often forced to use parallel, hybrid and distributed infrastructure, to process large instances of the problems and to quickly deliver the solution. Therefore feasible scaling of the system is necessary.

Over one decade ago, super-scalability notion was coined. A "super-scalable" method should be able to efficiently use thousands of computing nodes with tens of thousands computing cores, provided by contemporary high-performance computing (HPC) systems or computing cloud services. Efficient utilization of computing resources is understood as linear or almost linear growth in simulation performance with the increase of available computing units. A good definition of "super-scalability" is given in [16], where the authors advocate that such algorithms should be scale invariant and naturally fault-tolerant. The individual tasks in a larger parallel job should have a fixed number of other tasks they communicate with, independent of the total number in the application. Naturally fault-tolerant tasks should have the internal ability to tolerate failures, without requiring notification or remote recovery.

An interesting feature of such super-scalable systems is that they are naturally fault-tolerant. Thus we advocate that encompassing the desynchronization in the algorithm could allow to strive towards super-scalability, instead of fighting for constant synchronization, increasing the non-scalable part of the whole system, still being governed by rules such as Amdahl's law [17]. We have already put some effort to realize desynchronized simulations [4,18] based on the modern technologies like Erlang or Scala with Akka. The concept of controlled desynchronization allowed the algorithms to scale up to hundreds nodes without significant violation of the model and the results. Now we would like to enhance the metaheuristic computing in such a way.

Let us remember, that metaheuristic computing has always dealt with approximation and uncertainty. Why do not we design parallel and distributed algorithms, capable of dealing with (a certain level of) desynchronization, e.g. by accepting somewhat "late" information from the neighboring processes about migrating individuals (in the parallel evolutionary algorithms). Thus we would like to strive towards reaching high scalability of distributed and parallel implementations of ACO — and later try this also for other algorithms which can be parallelized.

3. Parallel and distributed ant colony computing

The behavior of natural ants colonies was an inspiration for the Ant colony optimization algorithms. At first it was used to solve the traveling salesman problem (TSP) in [7]. Then it was described as a meta-heuristic algorithm in [19].

The optimization problem has to be described as a graph consisting of a finite set of *components* connected by edges with assigned costs in order to be solved with the ACO algorithm. For a path to be a feasible solution it has to respect the posed restrictions and fulfill the requirements defined by the problem. The optimal solution is the path with the minimum cost, which is defined as the function of all of the costs of all connections belonging to the solution.

The optimization process in the ACO algorithms is driven by a population of ants (agents) that iteratively traverse a graph searching for a new solution. During each iteration, each ant builds a new feasible solution. It starts in some initial vertex and gradually moves to subsequent vertices. The selection of the path is guided by a probabilistic decision rule. The rule considers both the heuristic attractiveness of the path and the values of the pheromone trails left on the edges by the previous generations of ants. In the basic ACO algorithm – Ant System (AS, [20]) – the probability of moving from component i to component j for ant k in iteration t is defined as follows:

$$p_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}(t)]^{\beta}}{\sum_{k \in allowed_k} [\tau_{ik}(t)]^{\alpha} \cdot [\eta_{ik}(t)]^{\beta}} & \text{if } j \in allowed_k \\ 0 & \text{otherwise} \end{cases}, \tag{1}$$

where $\tau_{ij}(t)$ is the intensity of a pheromone trail on an edge and $\eta_{ij}(t)$ is the visibility of the edge, which can be defined as the inverse distance between cities in the case of the TSP. The indexes

i and j are labels of the vertices. $allowed_k$ is the set of the vertices which can be reached from the kth vertex. The parameters that control the relative importance of trail versus visibility are α and β. Finally, $allowed_k$ is a set of possible transitions for ant k. The tour ends when a feasible solution is found. Based on its knowledge, the ants update the pheromone trails on their paths. In the classic AS version, the update is performed at the end of a single iteration according to the following formula:

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \tag{2}$$

where $\rho \in [0, 1)$ is a pheromone persistence coefficient and m is the number of ants. The pheromone update value for each ant is defined as follows:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & \text{if } k\text{th ant uses edge } (i, j) \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

where L_k is the tour length of the kth ant and Q is a constant (often with a value of 1).

Numerous improvements have been proposed since the first publication describing Ant System, Max–Min Ant System (MMAS, [21]) being one of the most efficient and commonly used. It was based on the standard AS algorithm but introduced the three major changes:

- the pheromone trials values are limited by τ_{min} and τ_{max},
- the pheromone matrix is initialized with the τ_{max} value,
- only one solution strengthens the pheromone trails after each iteration — it is either the best from the current iteration or the globally best found so far by the algorithm.

A further improvement of the algorithm is a so called *pheromone trail smoothing* mechanism. When the algorithm falls into stagnation, the pheromone matrix gets smoothed by increasing the values proportionally to their difference to τ_{max} in order to increase exploration of the solutions space. Other ant systems can also profit from applying this mechanism [20,22].

The Ant Colony Optimization algorithm with modifications has been used in many use-cases. Besides transport related problems like TSP, VRP or PDP, it can be used for community detection in large-scale social networks [23] or load balancing in Peer-to-Peer networks [24]. Each problem type has some problem-specific optimizations which can be applied in order to improve the ACO performance by reducing its time and memory complexity [25] or enabling faster convergence with local optimizations [26].

3.1. Parallel and distributed ACO

Besides improvements aiming at better solutions, the performance boost is another goal of the ACO research. Some researchers focus on a parallelization of the algorithm on a single machine with multi-core processor [27,28], but there are also papers focused on ACO distributed across multiple computation nodes [8]. Schlueter and Munetomo proposed a parallel problem function evaluation in MIDACO framework [29] designed to solve the optimization of problems known as mixed-integer nonlinear programs (MINLP). The variety of approaches encouraged attempts of their classification. The taxonomy proposed in [30] divides the algorithms into four main categories based on two primary criteria: the number of colonies (one or many) and the cooperation or lack thereof among the parallel parts:

- *Master–slave model* – a single colony without cooperation, where the master process manages the global information,
- *Cellular model* – a single colony with cooperation, where the colony is divided into small overlapping neighborhoods with their own pheromone matrices,
- *Parallel independent runs model* – several sequential ACOs are concurrently and independently executed on a set of processors using identical or different parameters,
- *Multi-colony model* – several colonies explore the solution space with their own pheromone matrices but periodically exchange information.

The researchers often focus on distributed algorithms limiting the communication between ants or even separating ants into multiple colonies with no communication between them. There are only a few articles regarding the implementation of a single ant colony in a cluster environment. Ilie and Bădică et al. [8] presented an agent-based approach to ACO modeling. This article reports a very good scalability up to 7 nodes on the *gr666* problem from TSPLIB.[1] The algorithm reaches solutions comparable to the standard Ant System. Unfortunately, no results were presented for large-scale TSP problems nor larger clusters.

As we have shown in [6] the standard Ant System can be implemented in terms of actor model and reach good scalability up to dozens of nodes in HPC environment. The proposed architecture is close to the *Master–slave model*, but keeps the pheromone matrix parts distributed across all nodes. The good scalability allows usage of higher number of ants in a single colony, which improves efficiency of the results [5,6]. In [5] we have proposed many ways to efficiently employ large ant colonies in order to outperform algorithms like MMAS in terms of solutions quality.

3.2. Agent-based computations

The agent-based architecture is a very convenient way of describing parallel algorithms. This way of algorithms modeling is supported by numerous frameworks, like REPAST HPC [31], based on the message passing interface (MPI), or FLAME [32,33], which is based on a distributed memory model. The quality of these frameworks has been proven in numerous ways for implementing parallel, distributed and HPC applications, but programming in low-level languages is rather difficult and error-prone. Therefore, turning to more-current technologies might be useful in order to speed-up the prototyping and development of such systems.

The actor model proposed in 1973 by Carl Hewitt [14,34] introduces the message passing concurrency model and defines the concurrent processes as the basic units of parallel computations. These processes use asynchronous messages as the communication mechanism. As a result of receiving a message, an actor can change its state, send messages, or create new actors.

Describing the algorithm parts as actors and their interactions as messages passing makes design and interpretation of the algorithms easier. The implementation based on the modern technologies like the Scala language and the Akka framework enables programming on the higher abstraction level without significant performance decrease. This approach has been proven in both computation [5] and simulation [35] applications.

In this paper, we would like to focus on improving the scalability of the Distributed ACO [5] by reducing the synchronization pauses and verify influence of these changes on the algorithm efficiency.

4. Desynchronization in distributed ant colony optimization

This section presents the changes in the architecture described in [5]. The proposed modifications introduce some asynchronism into the agents communication in order to improve the system's scalability. The desynchronized algorithm is expected to have

[1] https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

similar efficiency in terms of the solutions quality to the previous implementations.

The standard decentralized algorithm performs the following steps at the end of each iteration:

- collect a solution from each local ant,
- broadcast the solutions to all other nodes,
- wait for the solutions from all other nodes,
- update the pheromone matrix.

During these steps the ants cannot build next solutions, so the idea of desynchronization is to allow building new solutions based on the available pheromone values (even from the previous iteration). Moreover, the update is performed independently for each node's batch in order to deliver updates faster and make a single update shorter.

Fig. 1 presents an example of communication in case of two nodes, each with a single ant. As soon as all local solutions are collected, pheromone trails are updated locally and the data is broadcasted to other nodes. In the meantime the ants are able to use the old pheromone values in order to create next solutions. After the remote update is applied, the ants start using the updated values for the subsequent steps. In the standard implementation the ants have to wait for the pheromone data update across the whole cluster.

Below we present a simplistic pseudocode of handling ant's request and matrix updates by a pheromone manager. The first one presents a standard (decentralized) version. In this version the pheromones manager first checks whether the request concerns current iteration. If so, the response is sent immediately. Otherwise, it is queued and waits until the matrix is updated, when all queued requests are resolved. This way we obtain the synchronization of iteration among ants. The manager updates the matrix once it collects information about iteration solutions found from all nodes.

Algorithm 1 Ant request handling — standard

```
1: procedure StandardHandleAntRequest(iteration)
2:     if matrixUpdated(iteration) then SendResponse()
3:     else AddRequestToQueue(iteration)
4: procedure StandardHandleLocalSolution
5:     AddSolutionToLocalBatch()
6:     if batchSize == antsCount then BroadcastUpdate()
7: procedure StandardHandleUpdateBatch
8:     AddBatchToQueue()
9:     if batchesCount == nodesCount then StandardMatrixUp-
   date()
10: procedure StandardMatrixUpdate(iteration)
11:     ApplyUpdateAndEvaporation()
12:     HandleAllQueuedRequests(iteration)
```

The second pseudocode presents corresponding procedures in case of pheromones manager in desynchronized version of the algorithm. It is simplified in comparison to the previous one as there is no verification of current iteration and all requests are handled immediately. It also applies pheromone updates from each node independently, right after receiving an update batch.

The updates based on smaller batches require modification of the pheromones evaporation algorithm. In order to keep the system parameters meaning analogous, we have decided to apply evaporation after each batch with evaporation coefficient divided by the number of nodes.

Introduced changes blur the boundaries between iterations, whereas on the other hand we use iterations as a stop condition and for measuring the system's scalability. Therefore we have to estimate the current iteration by counting total solutions built in the whole cluster and dividing it by the number of all ants.

Algorithm 2 Ant request handling — desynchronized

```
1: procedure DesynchronizedHandleAntRequest
2:     SendResponse()
3: procedure DesynchronizedHandleLocalSolution
4:     AddSolutionToLocalBatch()
5:     if batchSize == antsCount then BroadcastUpdate()
6: procedure DesynchronizedHandleUpdateBatch
7:     ApplyUpdateAndEvaporation()
```

The new way of updating pheromone values also requires changes in the caching mechanism. The value for each remote edge can be cached locally after the first request. Then the edge value is removed from the cache when a node receives a batch of solutions or an update acknowledgment from the node which owns this edge.

The desynchronized version of the algorithm is mostly equivalent to the decentralized one in terms of the number of messages passed in the system. The number of ant's requests for pheromone values in a single iteration remains unchanged and is equal to solution size. These messages are sent locally on a single machine. The same applies to reporting a solution found at the end of iteration — it is a single local message sent to local pheromones manager. Inter-node communication happens between pheromones managers. Statistically once per iteration each of them broadcasts information about locally found solutions to other managers. The only main difference between both versions concerns loading pheromone values from remote managers. In the decentralized version it happens once per iteration. In the desynchronized one, according to the previous paragraph, it may happen twice per iteration.

As there is no synchronization at the end of each iteration the size of the broadcasted solutions batch can be configurable. It might be equal to the number of ants on a single node (so that it resembles broadcast after each iteration) or higher in order to reduce inter-node communication. At the same time, each node reports to the sender that the batch was applied, which allows the sender to wait for all nodes to apply the previous batch before sending the next one. This way the system is able to adjust the batch size dynamically if some nodes generate batches too fast.

In order to reduce the impact of network communication, the nodes do not broadcast the whole solution paths. Instead, each node creates a map from edges to a sequence of the solutions cost. Then it sends to each node only the part of the map containing the edges owned by the target node. This prevents the increase of total amount of data sent over the network in case of high number of nodes.

5. Experimental results

The experiments were conducted on the Prometheus supercomputer — a peta-scale (2.4 PFlop) cluster located in the Academic Computer Center Cyfronet AGH in Krakow, Poland. As of June 2019, Prometheus was ranked 174th in the TOP500 fastest supercomputer list. Prometheus is a cluster of HP Apollo 8000 nodes, each with 24 Xeon E5-2680v3 CPUs working at a 2.5 GHz frequency and connected via an InfiniBand FDR network.

The HPC infrastructure allowed us to run various experiments involving hundreds of nodes. We used the TSP problems from TSPLIB as a benchmark. As the goal of this paper is to present a general-use meta-heuristic algorithm, we decided not to implement any TSP specific local optimizations (e.g., 3-opt) or performance improvements (e.g., narrowing neighborhood in the

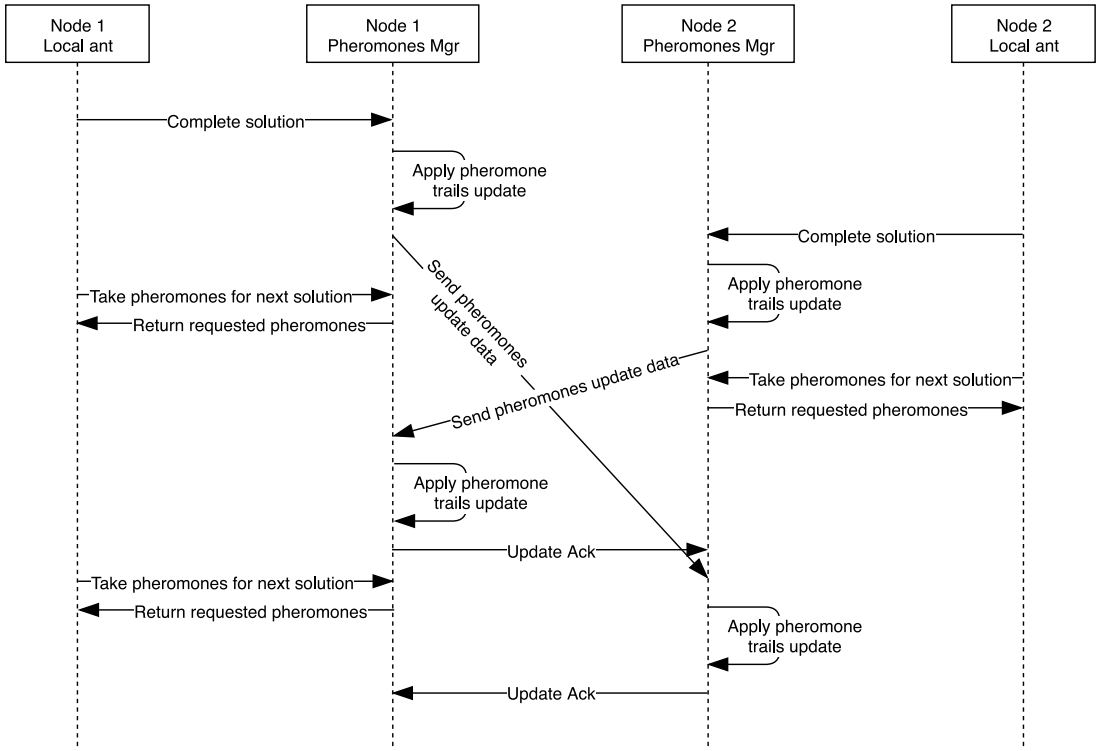


Fig. 1. An example of the communication in desynchronized ACO.

planning of the next step). This decision makes the results more likely to be applicable in case of other problems.

First, we present the set of experiments comparing the scalability of the desynchronized algorithm with the previous distributed version. Then, we show the influence of the ants count on the optimization results. Finally, we move on to the experiments that explore the impact of the synchronization frequency on the solutions found.

Some algorithm parameters remained unchanged for most of the experiments (See Eq. (1)). The default values are listed below, where *Nodes* is the number of computation nodes:

- ants per node $N = 25$,
- broadcast local solutions minimum count $SyncN = 25$,
- pheromone initial value $\tau(0) = 0.9999$,
- pheromone update constant $Q = 0.05/(10 * Nodes)$,
- pheromone persistence coefficient $\rho = 0.95$,
- pheromone trial value limits $\tau_{Max} = 0.9999$, $\tau_{Min} = 0.0001$,
- pheromone trial importance $\alpha = 3$,
- heuristic visibility importance $\beta = 5$,
- iteration best updates $IBU = 10 * Nodes$,
- global best updates $GBU = 10 * Nodes$.

The first set of experiments proves that the desynchronized algorithm scales better than the previous decentralized version, see Tables 1 and 2. On smaller clusters the new approach is slower, but on more than 100 nodes it outperforms the synchronized version which was not able to reach sensible iteration time on 200 nodes or more. Up to 50 nodes the speedup of the distributed version is better, but the desynchronized keeps efficiency higher than 0.75 even up to 400 nodes. Each configuration was repeated 10 times with 200 iterations, which are estimated as feasible solutions count divided by the ant count in case of Desynchronized ACO.

The distributed algorithm is more efficient on the cluster with less than 50 nodes, because time lost on the nodes synchronization is smaller than the time spent on multiple pheromone matrix transitions after each solutions batch received from every node in the cluster. The efficiencies higher than 1.0 on the smaller clusters result from the distribution of the pheromone matrix update process, since its size does not change while adding new nodes and ants.

Due to high scalability we are able to run numerous ants without significant increase of computation time. In Table 3 we compare the results of four algorithms: the standard MMAS, the distributed Ant System implementation, the enhanced version of the distributed Ant system [5] and desynchronized version of it. Distributed Ant System ran 400 iterations for both problems, Enhanced Distributed Ant System had 1000 iterations, MMAS had 2500 iterations and Desynchronized Ant System had 200 iterations. Each tests case was repeated 10 times.

All proposed versions of distributed and desynchronized Ant System significantly outperforms the standard implementation of Max–Min Ant System. First, the distributed algorithm uses higher number of ants in order to increase exploration level, which results in better final results. The enhanced distributed algorithm improved the usage of data collected by numerous ants, leading to similar solutions with a smaller colony running on a smaller computation cluster and shorter computation times. The reduced computation time allowed to run more iterations and slightly improve the results. Desynchronization of the algorithm slightly reduced its efficacy, but its scalability allowed us to run more ants in a similar time and provide the best results using bigger ant colonies. The desynchronized algorithm combines the advantages of big ant colonies and efficient usage of the knowledge collected by numerous ants.

Table 1
Iteration time (s) on pr2392 from TSPLIB.

Nodes	1	2	5	10	20	50	100	200	300	400
Distributed ACO	16.77	15.74	12.65	12.24	11.63	13.9	39.93	–	–	–
Desynchronized ACO	16.75	17.72	15.55	15.58	15.94	17.93	19.34	20.72	21.35	22.00

Table 2
Speedup on pr2392 from TSPLIB.

Nodes	1	2	5	10	20	50	100	200	300	400
Distributed ACO	–	2.13	6.22	10.33	28.98	43.31	62.04	–	–	–
Desynchronized ACO	–	1.89	5.38	10.75	21.01	46.71	86.64	161.67	235.41	304.58

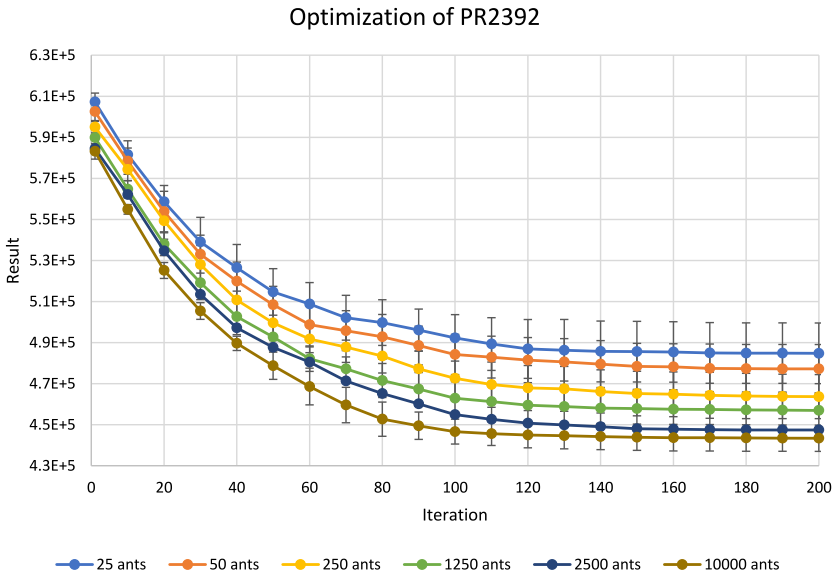


Fig. 2. Best solution over iterations with different ants number; Problem TSPLIB: pr2392.

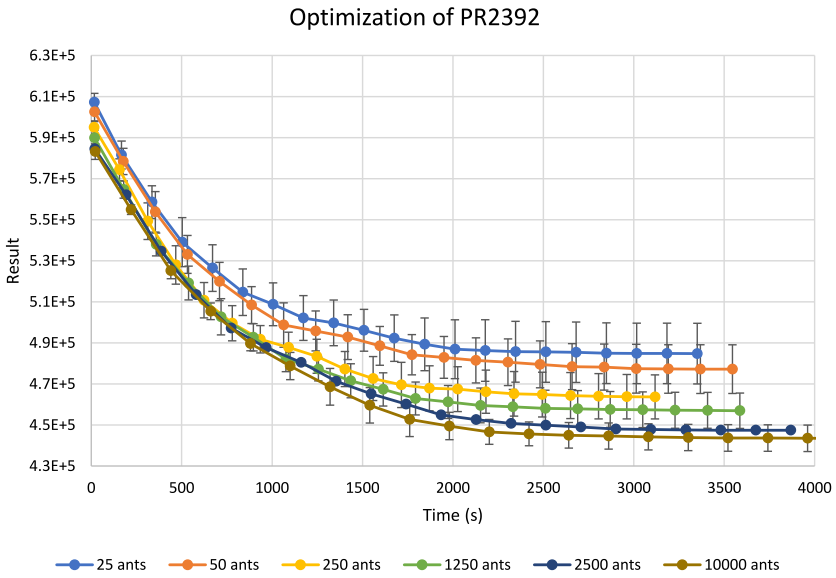


Fig. 3. Best solution over computation time with different ants number; Problem TSPLIB: pr2392.

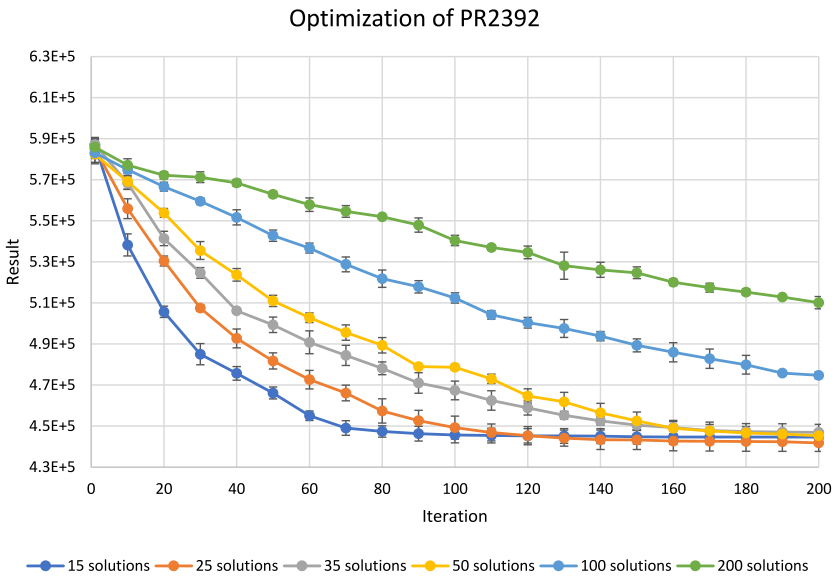


Fig. 4. Best solution over iterations with different synchronization periods and 5000 ants on 200 nodes; Problem TSPLIB: pr2392.

Table 3
Comparison of results achieved by MMAS, Distributed Ant System and Desynchronized Ant System on pr2392 from TSPLIB.

	Result
Best known solution	378032
MMAS (25 ants)	494735.2 ± 2983.4
MMAS (250 ants)	489201.8 ± 1481.6
Distributed Ant System (1k ants)	471657.8 ± 3201.2
Distributed Ant System (10k ants)	454735 ± 1058.9
Enhanced Distributed Ant System (250 ants)	455035.2 ± 5446.7
Enhanced Distributed Ant System (500 ants)	452700 ± 5998.7
Desynchronized Ant System (250 ants)	463671 ± 10727.0
Desynchronized Ant System (500 ants)	459025 ± 10002.5
Desynchronized Ant System (2500 ants)	447427 ± 3153.9
Desynchronized Ant System (10k ants)	443377 ± 6392.4

Table 4
Synchronization period influence on scalability on 200 nodes.

Synchronization period (solutions)	15	25	35	50	100	200
Iteration time (s)	39.91	20.72	14.16	10.54	8.09	6.65

Fig. 2 shows that the more ants the better the results are in each iteration. Fig. 3 presents the same results with the X-axis (number of iterations) replaced by the computation time based on the average iteration time from Table 1. The results of colonies bigger than 250 ants are similar in the first 1000 s, but the smaller the colony is the sooner it reaches stagnation. It proves that the scalability is good enough to take advantage of expanding exploration with higher number of ants, which leads to better final results.

The desynchronization of the algorithm allows us to select how often each node broadcasts the data. As presented in Table 4 the value significantly affects the iteration time.

Fig. 4 shows that the more often the nodes exchange the data the faster (in terms of the number of iterations) optimization we get. We also noticed that the final result is similar for synchronization period lower that 50 solutions. However, Fig. 5 points out that the increase of the iteration time (resulting from frequent data exchange) levels faster optimization and the best results are obtained for synchronization period set between 25 and 50 solutions. Such configuration leads to reasonable balance between the iteration time and optimization rate.

The presented experiments show that desynchronization of the algorithm improves the scalability without significant reduction of the efficiency. It allows to run much bigger ants colonies in the same time which notably improves final results. The proper balance between ant colony size, efficient usage of collected data and desynchronization level is crucial to obtain good results. It is easy to improve scalability with reduced data exchange periods, but it also inhibits efficient data usage which might hinder optimizing the solutions.

The proposed algorithm does not include any TSP specific optimizations, so it is applicable to the same class of optimization problems as the standard ACO algorithm. The usage of the algorithm with a big computation cluster is not justified in case of small optimization problems, where standard or parallel algorithms are good enough. In case of TSP the algorithm could be used for much bigger problems, but it should employ problem specific optimizations (e.g., narrowing neighborhood of each node) in order to reduce the problem's complexity and computation time of a single iteration.

6. Conclusion

In this paper, we have proposed desynchronization of the distributed ACO algorithm presented in [5]. The introduced modifications allowed the algorithm to scale up even to 400 computation nodes while running 10000 ants in a single colony, yet their impact on the results quality is negligible. The proposed system is based on the actor model and implemented with a popular and easy-to-use high-level technology that allows a very efficient verification of different system designs. The desynchronized algorithm enables very effective usage of HPC equipment. We have also explored the influence of synchronization period on the optimization efficacy. The presented algorithm does not use any problem specific optimizations, so it can be easily applied to any problem solvable with the standard ACO meta-heuristic.

The actor model allowed a very intuitive description of the desynchronized processes. The desynchronization was focused on a high scalability of the algorithm running numerous ants in a single colony. The most important change is removal of the

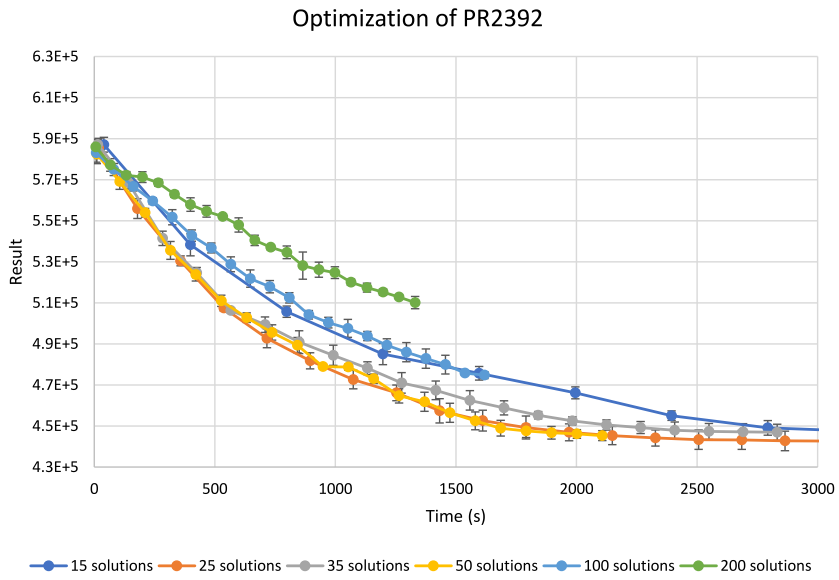


Fig. 5. Best solution over computation time with different synchronization periods and 5000 ants on 200 nodes; Problem TSPLIB: pr2392.

synchronization point of all ants at the end of each iteration. The pheromone matrix is updated with small batches and allows the ants to use the most current values ignoring an iteration of the ant.

The experiments have proven that the desynchronization does not affect the algorithm's efficacy in a noticeable way. At the same time, higher number of ants allows the algorithm to reach better solutions and improves the optimization speed. The desynchronized algorithm outperforms the standard Max–Min Ant System roughly by 10%. Finally, the experiment showed that the best solutions broadcasting period is between 1–2 times the number of ants on a single node. Higher values improve the scalability, but also reduce the information exchange which leads to much worse solutions. On the other hand, lower values force a lot of network communication and result in much longer iteration's time.

To sum up, the proposed desynchronized algorithm has shown that it is possible to efficiently run huge ant colonies on HPC equipment. High number of ants leads to better optimization results, so usage of the HPC equipment is reasonable.

In the future, we plan to explore the possibilities unveiled by the numerous ants in a single colony. It could be very useful for socio-cognitively inspired ant colony optimization [36] to efficiently handle multiple types of ants. We also want to verify the system against the multi-objective problems. The application of this system design to other swarm intelligence algorithms (or more generally to algorithms with global knowledge) is another path for future research in this topic.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Mateusz Starzec: Conceptualization, Investigation, Methodology, Software, Visualization, Writing - original draft. **Grażyna Starzec:** Conceptualization, Investigation, Methodology, Software, Visualization, Writing - original draft. **Aleksander Byrski:** Conceptualization, Investigation, Methodology, Supervision, Writing - review & editing. **Wojciech Turek:** Conceptualization, Methodology, Writing - review & editing. **Kamil Piętak:** Conceptualization, Methodology, Supervision, Validation.

Acknowledgment

The research presented in this paper was supported by the Polish Ministry of Science and Higher Education funds assigned to AGH University of Science and Technology. The authors utilized the PLGrid infrastructure when conducting the experiments described in this work.

References

[1] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Trans. Evol. Comput. 67 (1) (1997).
[2] E. Cantú-Paz, A survey of parallel genetic algorithms, Calc. Paralleles Reseaux Syst. Repartis 10 (2) (1998) 141–171.
[3] Wojciech Turek, Leszek Siwik, Aleksander Byrski, Leveraging rapid simulation and analysis of large urban road systems on HPC, Transp. Res. C 87 (2018) 46–57.
[4] Jakub Bujas, Dawid Dworak, Wojciech Turek, Aleksander Byrski, High-performance computing framework with desynchronized information propagation for large-scale simulations, J. Comput. Sci. 32 (2019) 70–86.
[5] Mateusz Starzec, Grażyna Starzec, Aleksander Byrski, Wojciech Turek, Marek Kisiel-Dorohinicki, Distributed ant system for difficult transport problems, J. Intell. Fuzzy Systems 37 (6) (2019) 7347–7356.
[6] M. Starzec, G. Starzec, A. Byrski, W. Turek, Distributed ant colony optimization based on actor model, Parallel Comput. 90 (2019) 102573.
[7] Marco Dorigo, Optimization, Learning and Natural Algorithms (Ph.D. thesis), Politecnico di Milano, 1992.
[8] Sorin Ilie, Costin Bădică, Multi-agent approach to distributed ant colony optimization, Sci. Comput. Program. 78 (6) (2013) 762–774.
[9] Qiang Lv, Xiaoyan Xia, Peide Qian, A parallel ACO approach based on one pheromone matrix, in: International Workshop on Ant Colony Optimization and Swarm Intelligence, Springer, 2006, pp. 332–339.
[10] José M. Cecilia, José M. García, Andy Nisbet, Martyn Amos, Manuel Ujaldón, Enhancing data parallelism for ant colony optimization on GPUs, J. Parallel Distrib. Comput. 73 (1) (2013) 42–51.
[11] Rafał Skinderowicz, The GPU-based parallel ant colony system, J. Parallel Distrib. Comput. 98 (2016) 48–60.
[12] J. Allen, Effective Akka, O'Reilly Media, 2013.
[13] A. Silberschatz, G. Gagne, P.B. Galvin, Operating System Concepts, John Wiley & Sons, 2012.

[14] Carl Hewitt, Peter Bishop, Richard Steiger, Session 8 formalisms for artificial intelligence a universal modular actor formalism for artificial intelligence, in: Advance Papers of the Conference, Vol. 3, Stanford Research Institute, 1973, p. 235.
[15] A. Patel, J. Degesys, N. Radhika, Desynchronization: The theory of self-organizing algorithms for round-robin scheduling, in: Proc. of First International Conference on Self-Adaptive and Self-Organizing System SASO, 2007.
[16] C. Engelmann, A. Geist, Super-scalable algorithms for computing on 100,000 processors, in: Proc. of International Conference on Computational Science 2005, in: Lecture Notes in Computer Science 3514, Springer, 2005.
[17] M. Hill, M. Marty, Amdahl's law in the multicore era, Computer 41 (7) (2008) 33–38.
[18] Wojciech Turek, Erlang-based desynchronized urban traffic simulation for high-performance computing systems, Future Gener. Comput. Syst. 79 (2018) 645–652.
[19] Marco Dorigo, Gianni Di Caro, Ant colony optimization: a new meta-heuristic, in: Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, Vol. 2, IEEE, 1999, pp. 1470–1477.
[20] Marco Dorigo, Vittorio Maniezzo, Alberto Colorni, Ant system: optimization by a colony of cooperating agents, IEEE Trans. Syst. Man Cybern. B 26 (1) (1996) 29–41.
[21] Thomas Stützle, Holger H. Hoos, MAX–MIN ant system, Future Gener. Comput. Syst. 16 (8) (2000) 889–914.
[22] Bernd Bullnheimer, Richard F. Hartl, Christine Strauss, A New Rank Based Version of the Ant System. a Computational Study, SFB Adaptive Information Systems and Modelling in Economics and Management Science, WU Vienna University of Economics and Business, 1997.
[23] Dongxiao He, Jie Liu, Dayou Liu, Di Jin, Zhengxue Jia, Ant colony optimization for community detection in large-scale complex networks, in: 2011 Seventh International Conference on Natural Computation, Vol. 2, IEEE, 2011, pp. 1151–1155.
[24] Saied Asghari, Nima Jafari Navimipour, Resource discovery in the peer to peer networks using an inverted ant colony optimization algorithm, Peer-to-Peer Netw. Appl. 12 (1) (2019) 129–142.
[25] Hassan Ismkhan, Effective heuristics for ant colony optimization to handle large-scale problems, Swarm Evol. Comput. 32 (2017) 140–149.
[26] Georges A. Croes, A method for solving traveling-salesman problems, Oper. Res. 6 (6) (1958) 791–812.
[27] José M. Cecilia, José M. García, Re-engineering the ant colony optimization for CMP architectures, J. Supercomput. (2019) 1–22.
[28] Ekaterina Grakova, Kateřina Slaninová, Jan Martinovič, Jan Křenek, Jiří Hanzelka, Václav Svatoň, Waste collection vehicle routing problem on HPC infrastructure, in: IFIP International Conference on Computer Information Systems and Industrial Management, Springer, 2018, pp. 266–278.
[29] Martin Schlueter, Masaharu Munetomo, MIDACO parallelization scalability on 200 MINLP benchmarks, J. Artif. Intell. Soft Comput. Res. 7 (3) (2017) 171–181.
[30] Martín Pedemonte, Sergio Nesmachnow, Héctor Cancela, A survey on parallel ant colony optimization, Appl. Soft Comput. 11 (8) (2011) 5181–5197.
[31] Nicholson Collier, Michael North, Repast HPC: A platform for large-scale agentbased modeling, in: Bernard Schott Werner Dubitzky (Ed.), Large-Scale Comput. Tech. Complex Syst. Simul., Wiley, 2012, pp. 81–109.
[32] Mariam Kiran, Paul Richmond, Mike Holcombe, Lee Shawn Chin, David Worth, Chris Greenough, FLAME: simulating large populations of agents on parallel hardware architectures, in: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1-Volume 1, International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 1633–1636.
[33] Mariam Kiran, Mesude Bicak, Saeedeh Maleki-Dizaji, Mike Holcombe, FLAME: A platform for high performance computing of complex systems, applied for three case studies, Acta Phys. Polon. B 4 (2) (2011).
[34] Gul A. Agha, Actors: A Model of Concurrent Computation in Distributed Systems, Technical report, Massachusetts Inst of Tech Cambridge Artificial Intelligence Lab, 1985.
[35] Grażyna Skiba, Mateusz Starzec, Aleksander Byrski, Katarzyna Rycerz, Marek Kisiel-Dorohinicki, Wojciech Turek, Daniel Krzywicki, Tom Lenaerts, Juan C Burguillo, Flexible asynchronous simulation of iterated prisoner's dilemma based on actor model, Simul. Model. Pract. Theory 83 (2018) 75–92.
[36] Aleksander Byrski, Ewelina Świderska, Jakub Łasisz, Marek Kisiel-Dorohinicki, Tom Lenaerts, Dana Samson, Bipin Indurkhya, Ann Nowé, Socio-cognitively inspired ant colony optimization, J. Comput. Sci. 21 (2017) 397–406.

Mateusz Starzec received M.Sc. from the AGH University of Science and Technology in 2017. Currently he is finishing a Ph.D. at the Faculty of Computer Science, Electronics and Telecommunications, AGH University of Science and Technology. His research interests are simulation, distributed computing and metaheuristics.

Grażyna Starzec received M.Sc. from the AGH University of Science and Technology in 2017. Currently she is a Ph.D. student at the Faculty of Computer Science, Electronics and Telecommunications, AGH University of Science and Technology. His research interests are simulation, distributed computing and metaheuristics.

Aleksander Byrski received Ph.D. in 2007 and D.Sc. in 2013. He works as associate professor at AGH University of Science and Technology in Krakow Poland. He is interested in parallel and distributed computing, metaheuristic computing and agent-based systems.

Wojciech Turek obtained Ph.D. in 2010 and D.Sc. in 2018. He works as associate professor at AGH University of Science and Technology in Krakow Poland. He is interested in parallel and distributed computing and simulation systems, multi-robot systems and functional programming.

Kamil Piętak obtained Ph.D. in 2017. He works as assistant professor at AGH University of Science and Technology. His reseaarch interests are distributed computing and software engineering, he leads also a number of significant R&D projects.

Chapter 9

Desynchronization of simulation and optimization algorithms in HPC Environment

The concept of desynchronization may be useful for anyone who wants to utilize HPC infrastructure for a highly scalable algorithm. However, the removal of synchronization points will probably lead to altering the algorithm results. The influence of desynchronization could make simulation results useless when the algorithm is expected to be deterministic. On the other hand, it may be negligible for simulations of social relations or optimization algorithms. In this paper, the guideline with examples of desynchronization applications is presented. It aims to show possible directions of future work regarding desynchronization of algorithms in HPC environments.

Contributions of Mateusz Starzec: I was responsible for Conway's Game of Life and Iterated Prisoner's Dilemma related parts of the article. In cooperation with the other authors, we have prepared the general desynchronization guideline, *Introduction*, *Conclusion* and *Scalability in computations*.

Computer Science • 21(3) 2020 https://doi.org/10.7494/csci.2020.21.3.3772

Mateusz Starzec
Grażyna Starzec
Mateusz Paciorek

DESYNCHRONIZATION OF SIMULATION AND OPTIMIZATION ALGORITHMS IN HPC ENVIRONMENT

Abstract *The need for the scalability of an algorithm is essential when one wants to utilize an HPC infrastructure in an efficient and reasonable way. In such infrastructures, synchronization affects the efficiency of the parallel algorithms. However, one can consider introducing certain means of desynchronization in order to increase the scalability. Allowing certain messages to be omitted or delayed can be easily accepted in the case of metaheuristics. Furthermore, some simulations can also follow this pattern and thereby handle bigger environments. The paper presents a short survey on the desynchronization idea, pointing out already obtained results, or sketching out future work focused on scaling the parallel and distributed computing or simulation algorithms.*

Keywords scalability, desynchronization, simulations, optimization algorithms

Citation Computer Science 21(3) 2020: 319–333

1. Introduction

Since the accessibility of clusters or even HPC environments is becoming more and more common, the scalability of the algorithms gains importance. This scalability enables algorithms to solve huge tasks faster and simulate bigger problems.

A lot of work has been done to scale algorithms on multi-processor computers where the issue was the slow access to the system memory, which constrains the efficient utilization of multiple processors. An alternative way of achieving a speedup on a single computer is to use the GPUs in order to perform fast matrix computations; however, this requires a specific algorithm construction and data transfer between the system and the GPU memory. Working on the scalability across the cluster encounters a different obstacle – costly communication over the network between nodes enters into the equation.

As the communication between nodes may easily reduce the whole system's performance, there is a need to reduce its amount and organize it in such a way that impacts the computational efficiency as little as possible. It is rarely possible to ensure an identical load on all computational nodes, especially in a heterogeneous environment. Inequalities in the execution time of a single algorithm step will cause the fastest nodes to idly await the completion of the step on the slowest nodes, effectively wasting potential computational capabilities. The concept of desynchronizing inter-node data exchange aims to remove the synchronization points from the algorithm and replace them with an asynchronous data exchange. This might lead to the modified behavior of the algorithm when compared to the non-desynchronized approach.

In this paper, we review several methods of applying desynchronization to inter-node communication in various optimization algorithms and simulations. We note some issues that one should be aware of when applying such desynchronization to computations. The modifications enable high scalability but might also lead to unsatisfactory or even invalid results under some circumstances. The majority of the discussed cases are referenced from previous works related to the topic discussed. However, we decided it would be helpful to bridge the gap between optimization algorithms and complex simulations by introducing a new example – a simple implementation of Conway's Game of Life [4].

It is to note that the main aim of this paper is to present a short survey of the findings connected with the application of desynchronization in metaheuristic computing, also pointing out the possibility of applying the same mechanism (though more planning is required) to simulations, thus supporting the capability to efficiently scale the concurrent versions of the algorithms under consideration on parallel and distributed HPC infrastructures.

In Section 2, we sum up the current research status regarding the scalability of algorithms. Section 3 describes the concept of desynchronization in an abstract way and proposes some example applications for simulations and optimization algorithms. The next chapter presents example experiments regarding the desirable and unwanted results of desynchronization. In Section 5, we sum up the whole article.

2. Scalability in computations

As computation clusters and HPC environments become more accessible, the horizontal scalability of algorithms is becoming more and more important. A lot of articles have been published regarding the scalability of both optimization algorithms [17,18] and simulations [2,13,14]. However, the majority of this research focuses on multi-processor single-node scalability [1, 7] or, at most, dozens of computational nodes [8, 11]. The synchronization points related to updates of global knowledge are omnipresent. A lack of strict synchronization is treated as a risk of data loss, which leads to invalid results.

The optimization algorithms based on a population of agents are relatively easy to model and implement in parallel. As an example, particle swarm optimization [9] iteratively improves a set of candidate solutions basing on the best solutions collected in the previous iterations. Each solution can be handled independently, so it is straightforward to run in parallel. The algorithm was proven to be usable in inverse rendering with good scalability for up to ten nodes. Unfortunately, the efficiency drops for more computational nodes, and the tests were conducted for only up to 30 nodes [11]. The parallel implementation of PSO shown even super-linear speedups; however, the tests utilized only a single multi-core computer [1].

Ant colony optimization [5] runs multiple independent agents constructing candidate solutions based on a pheromone matrix that is updated after each iteration. The solution-construction processes are independent, so they could be run in parallel. The parallel version of the algorithm was adapted to a waste collection vehicle routing problem, reaching 6.8 speedup on a 24-thread processor [7]. Ilie and Bădică et al. [8] presented an agent-based approach to ACO modeling. This article reports very good scalability for up to 7 nodes on the *gr666* problem from TSPLIB[1]. The algorithm reaches solutions comparable to the standard ant system. Unfortunately, no results were presented for large-scale TSP problems nor for larger clusters.

As we have shown in [18], ACO can reach good scalability for up to hundreds of nodes in an HPC environment. The proposed architecture keeps the pheromone matrix parts distributed across all nodes and uses desynchronized updates in order to achieve good scalability without a noticeable deterioration of the result quality. The good scalability allows for increasing the size of a colony, which improves the achieved results [16, 17]. In [17], we have proposed many ways to efficiently employ large ant colonies in order to outperform algorithms like MMAS in terms of solution quality. What is more, the higher number of ants enables the faster optimization of big problem instances.

The distributed traffic simulation system [14] was proven to be able to efficiently utilize an HPC environment with 800 computational nodes. In the simulation, the traffic network was distributed across nodes with a limited desynchronization, allow-

[1] https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

ing the processes to compute a few iterations without waiting for updates from the neighbor crossings. The simulation achieved super-linear scalability.

Another approach to distribution and desynchronization was shown in [2]. Several simulation models were adapted to use the signal propagation method, which was designed to minimize the required communication between computational models. The resulting framework has proven to be highly scalable, efficiently utilizing thousands of computational cores.

The idea of desynchronization was also utilized in our publications regarding the iterated prisoner's dilemma with a large population of agents [13,15]. The experiments showed that desynchronization does not affect the behavior of a big population in any noticeable way. Moreover, the system achieved much better scalability than the version with a standard (synchronized) communication model between nodes, especially for environments with dense neighborhoods.

Desynchronization is a way to improve the scalability of the algorithms; however, it requires foresight and awareness of its potential impact on the algorithm's behavior. In the next section, we present a discussion on the desynchronization of metaheuristic and simulation algorithms.

3. Desynchronization in computations

One of the major issues regarding the scalability of algorithms is the access to global information. In a large cluster, the global knowledge may be stored on a master node and updated in a synchronous manner after each algorithm step. It takes some time to transfer the update data (any data collected from the nodes used for updating the global information), and the cluster is blocked from running another part of the computations until the synchronization is done, leading to the performance regression and lower scalability.

The basis of the desynchronization idea is to allow the algorithm to run when the global knowledge update is still in progress. The algorithm should start another iteration based on the old data and switch to the updated one as soon as the update is done, even in the middle of the iteration. The computations should be parallel to the updates of global knowledge in order to use the cluster resources more efficiently. Depending on the algorithm type, the parallelism may lead to a loss of result accuracy and conflicts in the global data updates, so each application should be considered carefully and individually.

One of the possible application fields are optimization algorithms, where a small delay of the global knowledge update will not be critical but the improvement of scalability will be noticeable. The negative influence of non-deterministic data losses or conflicts may be compensated by the possibilities facilitated by the high scalability.

On the other hand, the desynchronization of simulations is a substantially different problem. One of the significant differences between simulations and optimization algorithms is that a simulation represents a scenario in which intermediate steps might

be as important as the final result. This is significantly different from the optimization problems where only the final result is important in most cases. Additionally, the simulation state is often more complex and more susceptible to small changes. Therefore, desynchronization might not only cause simulations to yield imprecise results but also invalidate all observations made about the behavior of the system.

Fortunately, not all simulations employ deterministic mechanisms. In the case of a simulation algorithm that relies heavily on randomness, small discrepancies caused by the desynchronization may be less significant as compared to the variance of the results expected to occur naturally. Moreover, the more fine-grained the simulation is, the lesser the impact is of a single error or conflict to the whole process.

3.1. Desynchronization in optimization algorithms

In the case of optimization algorithms, we may often assume that small inaccuracies in global information do not result in significant changes to the final results. This section describes the application of desynchronization to various optimization algorithms.

The particle swarm optimization algorithm [9] consists of multiple independent agents – particles. It can be easily distributed to run on a cluster where each node can handle computations of some particles. The algorithm needs to know the global best solution in order to find new solutions. In the standard implementation, the global best solution should be updated after each iteration, which can reduce the system's scalability. The desynchronized algorithm could keep the global best solution locally and broadcast an update to all other nodes when it finds a better one. The particles could move independently based on the current local value of the global best solution. This might introduce some delays in knowledge distribution but should not affect the algorithm's efficacy, as the knowledge will be eventually consistent.

Social cognitive optimization [19] is another meta-heuristic optimization algorithm that uses global knowledge based on the data accumulated by independent agents. This creates a solution set (social sharing library) and updates it after each iteration with the solutions created by all agents in the system. The agents could be easily distributed in the cluster environment; however, updating the social sharing library requires the synchronization of all processes in the standard implementation. The desynchronized version may allow the agents to build the next solutions based on the current content of the sharing library and apply updates asynchronously. The sharing library may be maintained independently on each node, and the proposed solutions may be broadcasted to all nodes if they are good enough to replace one of the solutions currently contained in the list. The process of library update is not deterministic, so the content of the libraries may differ between nodes. Figure 1 presents an example of the communication sequence; it shows that the updates from the remote nodes are independent from the agents' iterations.

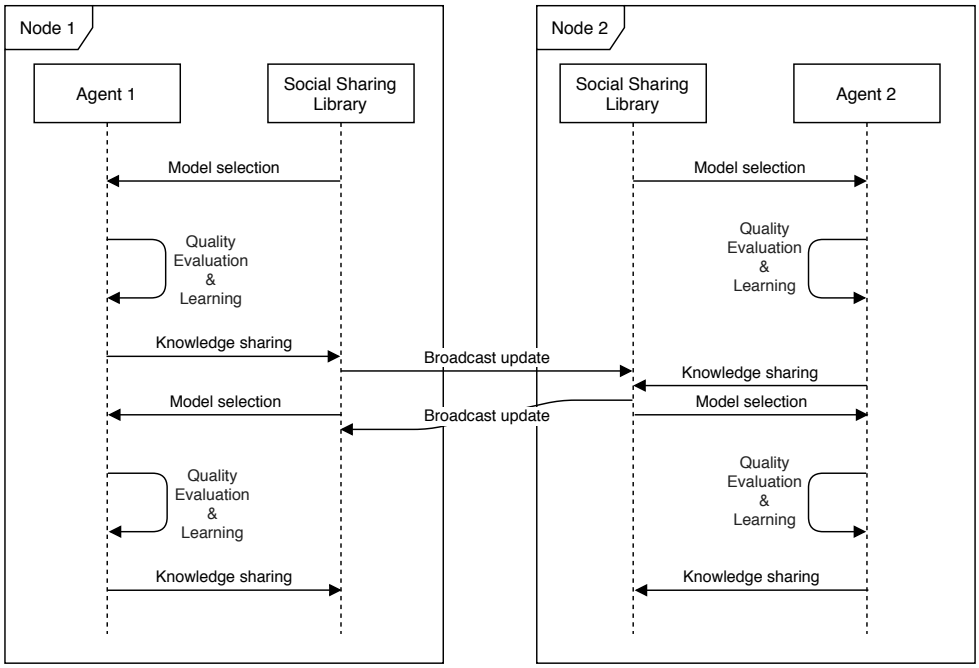


Figure 1. Social cognitive optimization – desynchronized social sharing library update

Evolutionary multi-agent systems [3] use the idea of islands as separated computation environments and utilize migrations as a communication and knowledge-exchange mechanism. The islands are feasible to be distributed across the computation nodes, and the desynchronized implementation should not synchronize the islands after each iteration in order to perform migrations of the solutions. The migrations may be handled asynchronously to increase the scalability of the system. Figure 2 presents an example of overlapping iterations and migrations.

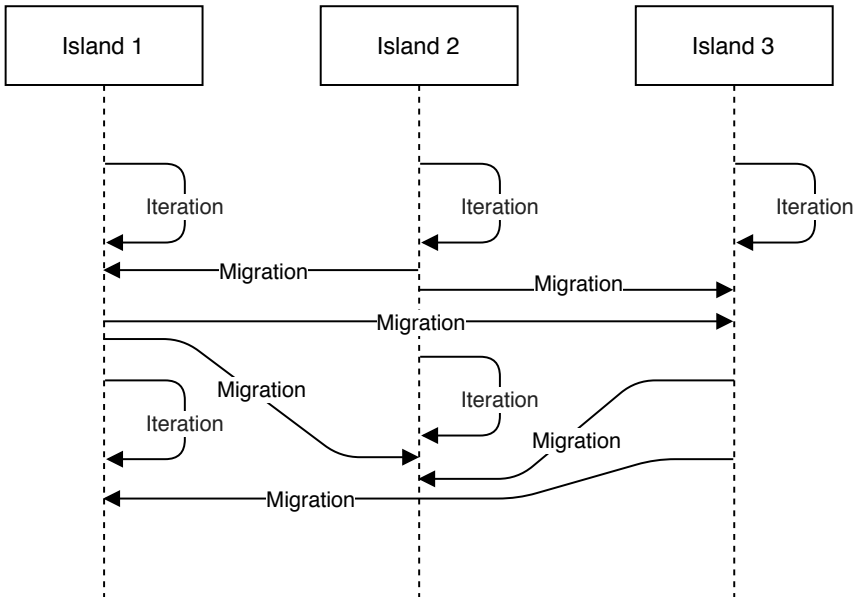


Figure 2. Evolutionary multi-agent systems – desynchronized migration process

After each iteration, each node can send an asynchronous message with the solutions to be migrated and handle such incoming messages. Sending and receiving the messages is not synchronized and may overlap with the iterations – this also means that the size of the population on an island can vary.

3.2. Desynchronization in simulations

In the case of simulations, discrepancies may have a significant impact on the final results. In this section, we describe how the desynchronization ideas may be applied to examples of simulations.

Conway's Game of Life [4] uses a grid as the simulation environment. In the case of a huge simulation, the grid could be distributed across the computation nodes; however, this requires inter-node communication on the edges of the grid's parts during each iteration. The synchronous data exchange may reduce performance, so it could be done asynchronously. On the other hand, the asynchronous approach requires caching the state of the remote neighbors, which could lead to altering the simulation results. Figure 3 presents an example of the grid distribution, with the cached rows represented as gray. The arrows represent the migration of the statuses after each iteration.

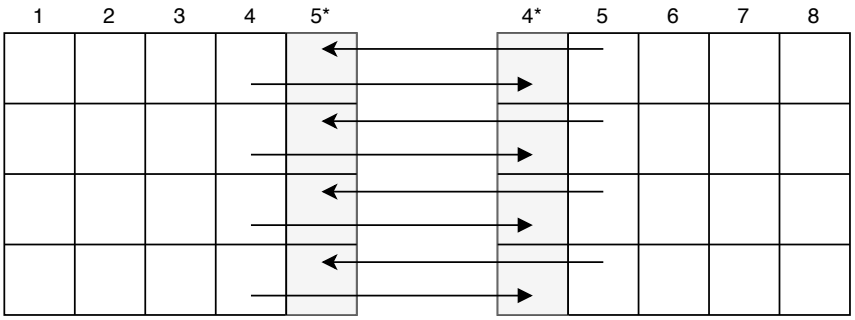


Figure 3. Conway's Game of Life – grid distribution with data caching and updates

A simulation of social processes (like the iterated prisoner's dilemma or a gift-exchange game) in a big society can be organized with some kind of neighborhood; e.g., a two-dimensional grid. To improve the scalability of the simulation, the neighborhood should be distributed in such a way as to minimize inter-node pairs. In order to further reduce the inter-node communication, the remote agents could be represented by local copies, which could asynchronously exchange their status updates with the original agents. The local copy should behave like the original one and provide it with updates regarding the results of the moves it made. Figure 4 presents the idea of a local representation of the remote agent. The communication between the nodes is replaced by the communication with local copies of the remote agents and the periodic synchronization with the original agents. This will reduce the network's communication and improve scalability.

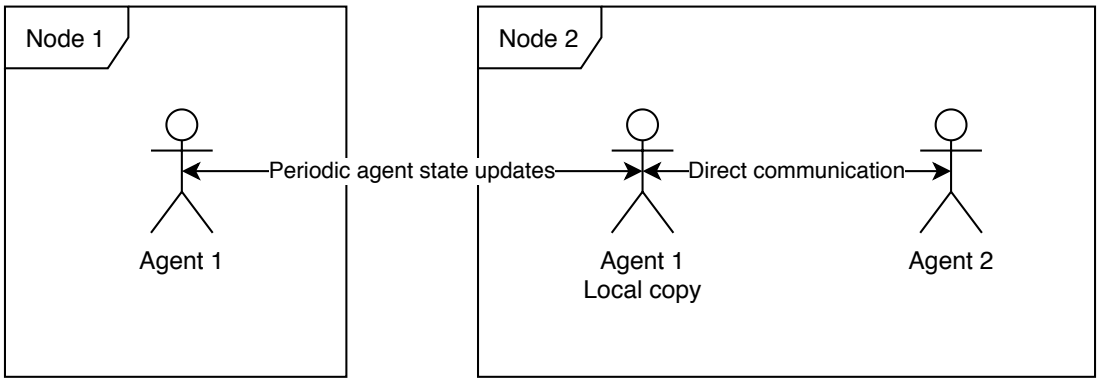


Figure 4. Social simulations – local copy of remote agents

More-complex simulations introduce much more space for the mistakes to either diminish or amplify. The main focus of [12] was to analyze the impact of desynchronization applied to several simulations: the simple predator-prey scenario, fire emergency evacuation, and microscopic organisms development on a seabed. The simulations represented a variety of levels of environment complexity and determinism. Each simulation was desynchronized in a similar manner – the grid was divided into parts and distributed among the computational nodes. The nodes were operating on their parts of the grid and periodically sending updates to the neighboring nodes. As a result, the algorithm had to be capable of resolving conflicts arising from the incoming data potentially contradicting the decisions made locally.

4. Experimental results

The implications of desynchronization may be different for various applications. In this section, we present the experimental results for some optimization algorithms and simulations.

4.1. Ant colony optimization

In the previous publication [18], we described and tested the desynchronized version of ant colony optimization. The idea of the algorithm was to remove the synchronization point at the end of each iteration (when the ants update the pheromone matrix). Instead of synchronizing the whole cluster, each ant submits its solution into a batch and immediately starts to create another solution (without waiting for the update of the pheromone matrix).

The main findings of the experiments conducted for the aforementioned publication are as follows. The algorithm reaches 76% efficiency on 400 computation nodes (see Tab. 1, $Efficiency = Speedup/Nodes$). The scalability enables calculations based on huge colonies, which leads to higher exploration and better final results. Table 2 shows that the bigger colony finds better final results when the exploration is utilized efficiently. The advantages of high scalability outweigh the drawbacks of desynchronized pheromone matrix updates.

Table 1
Desynchronized ACO speedup from [18]

Nodes	2	5	10	50	100	200	300	400
Distributed ACO	2.13	6.22	10.33	43.31	62.04	–	–	–
Desynchronized ACO	1.89	5.38	10.75	46.71	86.64	161.67	235.41	304.58

Table 2
Desynchronized ACO final results compared with MMAS from [18].

	Result
Best-known solution	378,032
MMAS (25 ants)	$494,735.2 \pm 2,983.4$
MMAS (250 ants)	$489,201.8 \pm 1,481.6$
Desynchronized Ant System (250 ants)	$463,671 \pm 10,727.0$
Desynchronized Ant System (500 ants)	$459,025 \pm 10,002.5$
Desynchronized Ant System (2500 ants)	$447,427 \pm 3,153.9$
Desynchronized Ant System (10k ants)	$443,377 \pm 6,392.4$

4.2. Iterated prisoner's dilemma

We have also applied the desynchronization to a simulation of the iterated prisoner's dilemma with a big society [13]. All agents are distributed across the computation nodes. When the game requires communication with a remote agent, it communicates with its local representation, which periodically synchronizes its state with the original agent. Figure 5 presents the results of the experiment from [13].

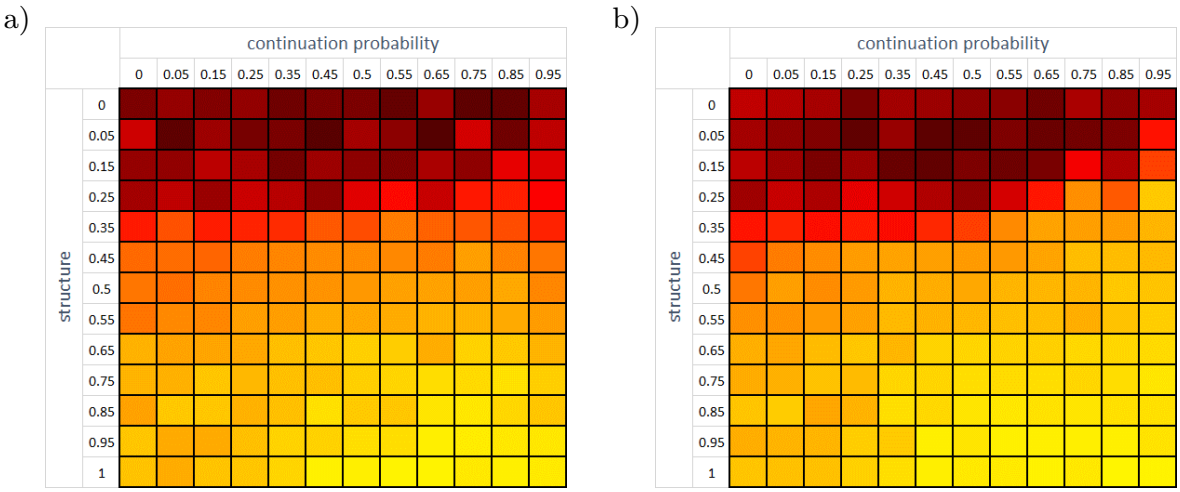


Figure 5. Desynchronization influence on IPD simulation from [13]:
a) without desynchronization; b) with desynchronization

It shows the level of cooperativity in the society depending on two parameters: the probability of playing the next turn with the same opponent (continuation probability), and the probability that an agent will be paired with another with a similar

strategy (structure). The dark color means low cooperativity, and bright yellow means a high level of cooperation among agents.

The desynchronization enabled us to simulate much bigger societies; at the same time, the results imply that this did not significantly influence the simulation results.

4.3. Conway's game of life

One of the most important problems described in the previous chapter was the impact of desynchronization on the simulation result. As an extension of the survey, we decided to further explore this issue using Conway's Game of Life [4] as a simple and well-recognized example. The simulation was implemented in a single process and emulated some conditions that are possible in a desynchronized environment (such as latency or packet losses).

In the simulation, we used a board with 10 rows and 20 columns. In each iteration:

- any black cell with two or three black neighbors remains black,
- any gray cell with three black neighbors becomes black,
- all other black cells become gray, and all other gray cells remain gray.

We assume that there are no neighbors outside the board. The consecutive board states in the standard simulation are presented in Figure 6a.

In order to simulate the environment with two nodes, we split the board into two parts (as presented in Fig. 3). Figure 6b presents the results with the cache updates delayed by one iteration (simulating latency in inter-node communication). The differences in the results appear in the center of the board and spread with each iteration.

Figure 6c shows the results of the simulation that drops every fourth cache update (simulating packet losses in inter-node communication). The result after 13 iterations also differs from the standard simulation. The interruption is not as severe as in the previous example, so the differences are smaller (but still noticeable).

The results prove that desynchronization is not suitable for applications where deterministic and consistent results are critical. On the other hand, when an algorithm is randomized, delays in the cache updates may be considered to be another random factor, which is not significant in terms of interpreting the results.

4.4. Complex simulations

The results obtained in [12] varied significantly among the simulation models. Two of the simulations – fire emergency evacuation and microorganism (foraminifera) habitat – were successfully distributed and desynchronized. The influence of the distribution proved to be marginal as compared to the changes caused by modifying the simulation parameters. An example of the results presented in [12] can be seen in Figure 7: each group of points marked on the x-axis represents one set of input parameters, while each of the series represents the degree of distribution; i.e., the number of parts into which the grid was divided.

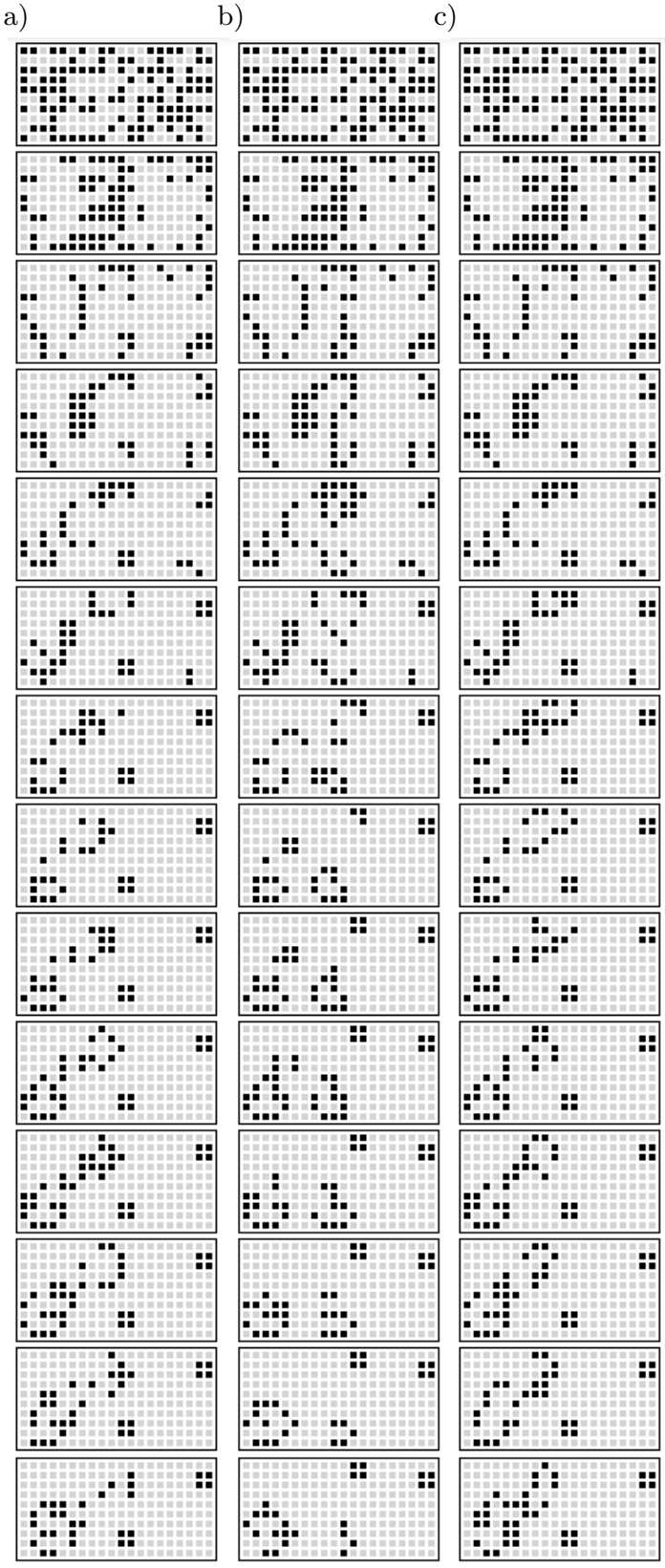


Figure 6. Conway's Game of Life simulation with two nodes:
a) simulation without delays; b) simulation with cache update delayed by one iteration;
c) simulation with every fourth cache update lost

Distribution had some impact on the measured values, which was the total energy among all simulated organisms in this case; however, more important was that the differences among configuration variants were noticeably greater than the differences between the values in each group.

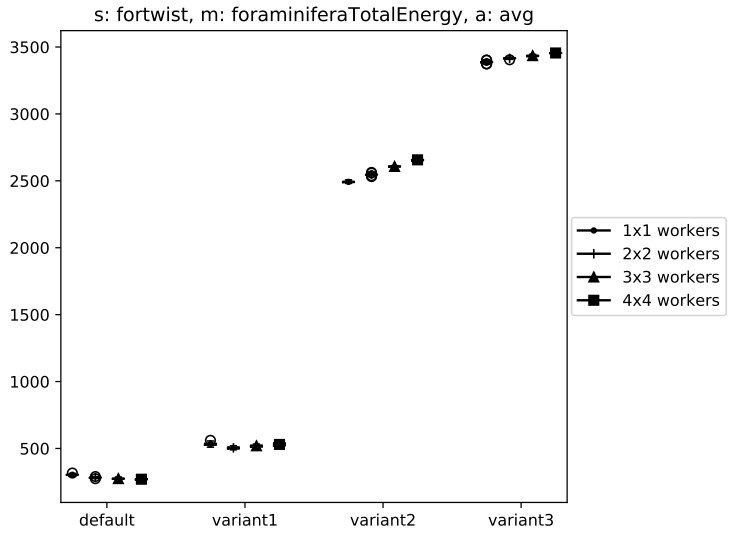


Figure 7. Simulation of foraminifera habitat: average total energy of foraminifera. Chart from [12].

On the other hand, the simulation of the predator-prey scenario yielded results that varied dramatically depending on the degree of distribution. The algorithm did not handle conflicts properly, as there was no possibility to resolve them in a way that would not violate the model. As a result, desynchronization led to the emergence of completely different behavior and inconsistent conclusions from the simulation.

In all three cases, the distribution and desynchronization led to substantial increases in the scalability of the simulations. In the two successful cases, the subtle differences in the measured values and observed results are definitely outweighed by the ability to assign many more computational resources. It is worth keeping in mind, however, that the simulation must be carefully analyzed and validated to ensure that the yielded results are useful.

5. Conclusion

In this paper, we have presented applications of the desynchronization concept to various algorithms. The concept aims to improve the scalability of the algorithms in HPC environments. It reduces the inter-node communication and limits synchronization-related hold-ups; however, it may affect the computational results. We have discussed two types of algorithms: simulations and optimization algorithms.

In the case of optimization, desynchronization might be applied to many algorithms with some kind of global knowledge; e.g., ant colony optimization, particle

swarm optimization, social cognitive optimization, or evolutionary multi-agent systems. The impact of desynchronization on the results is negligible and might be considered as another randomization factor. On the other hand, the speedup of the algorithm allows it to compute many more possible solutions of a problem, which leads to increased exploration and improved final results [18]. The speedup also enables faster optimization thanks to extensive parallelization.

Desynchronization might also be applied to simulations that involve multiple agents communicating with each other (e.g., the iterated prisoner's dilemma or gift-exchange game) or neighborhood-oriented global knowledge (like the board of Conway's Game of Life). The simulations require much more caution when subjected to desynchronization, since it might be the only randomization factor that affects the final results in an unacceptable manner. We have presented a simple experiment based on Conway's Game of Life, which has shown how desynchronization will significantly affect even such a simple simulation. When we introduced some desynchronization delays or lost updates, the results quickly became noticeably different.

On the other hand, the simulations of social processes are commonly randomized. In this case, desynchronization enables the system to simulate much bigger societies; however, one should be aware that the results might be affected by the specific algorithm construction. In the case of more-complex simulations, desynchronization might be successfully applied to models that heavily rely on randomness. There is no single strategy for handling such simulations; each desynchronized approach must be carefully analyzed and tested before being used as a proper model for further experiments.

To sum up, desynchronization is a concept that is applicable to multiple algorithms; however, it requires some caution (especially in the case of simulations where it might noticeably affect the results). Before drawing any further conclusions from the desynchronized version of a simulation, one should first estimate the influence of the introduced inaccuracies on the results. In optimization algorithms, we might assume that the introduced changes are negligible as long as we achieve comparable or better results when compared to the standard version. Hence, it is worth considering when a system requires high scalability in order to efficiently utilize HPC environments. One of the research areas related to optimization algorithms that has a strong potential of benefiting from introducing desynchronization is the domain of machine-learning algorithms. For example, it could be possible to desynchronize updating neuron weight matrices in neural networks, leading to improved efficiency and scalability.

Increasing scalability as a consequence of desynchronization actually does not require any explanation, being a simple consequence of applying Amdahl's or Gustafson's law [10] to predict the efficiency of concurrent and (in particular) distributed systems. However, one must ask the following question: to what extent does introducing desynchronization (thus allowing for some perturbations in the communication) impair the efficacy of the computing and simulation? We would like to try to answer this questions in one of our future works by constructing Markov chain models of possible perturbations and simulating several computing and simulation

cases similarly to well-known Markov-chain models of network packet loss (like the Simple Gilbert model [6]).

Acknowledgements

The research presented in this paper was supported by the Polish Ministry of Science and Technology funds assigned to AGH University of Science and Technology. The authors utilized the PLGrid infrastructure when conducting the experiments described in this work.

References

[1] Atashpendar A., Dorronsoro B., Danoy G., Bouvry P.: A scalable parallel cooperative coevolutionary PSO algorithm for multi-objective optimization, *Journal of Parallel and Distributed Computing*, vol. 112, pp. 111–125, 2018.

[2] Bujas J., Dworak D., Turek W., Byrski A.: High-performance computing framework with desynchronized information propagation for large-scale simulations. *Journal of Computational Science*, vol. 32, pp. 70–86, 2019.

[3] Byrski A., Dreżewski R., Siwik L., Kisiel-Dorohinicki M.: Evolutionary multiagent systems, *The Knowledge Engineering Review*, vol. 30(2), pp. 171–186, 2015.

[4] Conway J.: The game of life, *Scientific American*, vol. 223(4), p. 4, 1970.

[5] Dorigo M.: *Optimization, learning and natural algorithms*, PhD Thesis, Politecnico di Milano, 1992.

[6] Ellis M., Pezaros D.P., Kypraios T., Perkins C.: A two-level Markov model for packet loss in UDP/IP-based real-time video applications targeting residential users, *Computer Networks*, vol. 70, pp. 384–399, 2014, https://doi.org/10.1016/j.comnet.2014.05.013.

[7] Grakova E., Slaninová K., Martinovič J., Křenek J., Hanzelka J., Svatoň V.: Waste Collection Vehicle Routing Problem on HPC Infrastructure. In: *IFIP International Conference on Computer Information Systems and Industrial Management*, pp. 266–278, Springer, 2018.

[8] Ilie S., Bădică C.: Multi-agent approach to distributed ant colony optimization, *Science of Computer Programming*, vol. 78(6), pp. 762–774, 2013.

[9] Kennedy J., Eberhart R.: Particle Swarm Optimization. In: *Proceedings of ICNN'95 – International Conference on Neural Networks*, vol. 4, pp. 1942–1948, IEEE, 1995.

[10] McCool M., Robinson A., Reinders M.: *Structured Parallel Programming: Patterns for Efficient Computation*, Elsevier, 2013.

[11] Nagano K., Collins T., Chen C.A., Nakano A.: Massively parallel inverse rendering using Multi-objective Particle Swarm Optimization, *Journal of Visualization*, vol. 20(2), pp. 195–204, 2017.

[12] Paciorek M., Bujas J., Dworak D., Turek W., Byrski A.: Validation of signal propagation modeling for highly scalable simulations, *Concurrency and Computation: Practice and Experience*, p. e5718.

[13] Skiba G., Starzec M., Byrski A., Rycerz K., Kisiel-Dorohinicki M., Turek W., Krzywicki D., Lenaerts T., Burguillo J.C.: Flexible asynchronous simulation of iterated prisoner's dilemma based on actor model, *Simulation Modelling Practice and Theory*, vol. 83, pp. 75–92, 2018.

[14] Ślaski M., Turek W., Gil A., Szafran B., Paciorek M., Byrski A.: Analysis of Distributed Systems Dynamics With Erlang Performance Lab, *Computer Science*, vol. 19, 2018.

[15] Starzec G., Starzec M., Byrski A., Kisiel-Dorohinicki M., Burguillo J.C., Lenaerts T.: Towards Large-Scale Optimization of Iterated Prisoner Dilemma Strategies. In: *Transactions on Computational Collective Intelligence XXXII*, pp. 167–183, Springer, 2019.

[16] Starzec M., Starzec G., Byrski A., Turek W.: Distributed ant colony optimization based on actor model, *Parallel Computing*, vol. 90, p. 102573.

[17] Starzec M., Starzec G., Byrski A., Turek W., Kisiel-Dorohinicki M.: Distributed ant system for difficult transport problems, *Journal of Intelligent & Fuzzy Systems*, vol. 37(6), pp. 7347–7356.

[18] Starzec M., Starzec G., Byrski A., Turek W., Piętak K.: Desynchronization in distributed Ant Colony Optimization in HPC environment, *Future Generation Computer Systems*, 2020.

[19] Xie X.F., Zhang W.J., Yang Z.L.: Social cognitive optimization for nonlinear programming problems. In: *Proceedings. International Conference on Machine Learning and Cybernetics*, vol. 2, pp. 779–783, IEEE, 2002.

Affiliations

Mateusz Starzec
AGH University of Science and Technology, Department of Computer Science, Faculty of Computer Science, Electronics and Telecommunications, al. Mickiewicza 30, 30-059 Krakow, Poland, mateusz.starzec@iisg.agh.edu.pl

Grażyna Starzec
AGH University of Science and Technology, Department of Computer Science, Faculty of Computer Science, Electronics and Telecommunications, al. Mickiewicza 30, 30-059 Krakow, Poland, grazyna.starzec@iisg.agh.edu.pl

Mateusz Paciorek
AGH University of Science and Technology, Department of Computer Science, Faculty of Computer Science, Electronics and Telecommunications, al. Mickiewicza 30, 30-059 Krakow, Poland, mpaciorek@agh.edu.pl

Received: 26.04.2020
Revised: 28.06.2020
Accepted: 28.06.2020

List of Figures

List of Tables

Bibliography

[1] G. A. Agha. Actors: A model of concurrent computation in distributed systems. Technical report, Massachusets Institute of Technology Cambridge Artificial Intelligence Lab, 1985.

[2] E. Alba, G. Leguizamon, and G. Ordonez. Analyzing the behavior of parallel ant colony systems for large instances of the task scheduling problem. In *19th IEEE International Parallel and Distributed Processing Symposium*, pages 14–pp. IEEE, 2005.

[3] E. Alba, G. Leguizamon, and G. Ordonez. Two models of parallel aco algorithms for the minimum tardy task problem. *International Journal of High Performance Systems Architecture*, 1(1):50–59, 2007.

[4] J. Allen. *Effective Akka*. O'Reilly Media, 2013.

[5] B. Bullnheimer, R. Hartl, and C. Strauss. A new rank based version of the ant system - a computational study. *Central European Journal of Operations Research*, 7:25–38, 1999.

[6] A. Byrski, R. Drezewski, L. Siwik, and M. Kisiel-Dorohinicki. Evolutionary multiagent systems. *Knowledge Eng. Review*, 30(2):171–186, 2015.

[7] E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10(2):141–171, 1998.

[8] J. M. Cecilia, J. M. García, A. Nisbet, M. Amos, and M. Ujaldón. Enhancing data parallelism for ant colony optimization on gpus. *Journal of Parallel and Distributed Computing*, 73(1):42–51, 2013.

[9] J. Chintalapati, M. Arvind, S. Priyanka, N. Mangala, and J. Valadi. Parallel antminer (pam) on high performance clusters. In *International Conference on Swarm, Evolutionary, and Memetic Computing*, pages 270–277. Springer, 2010.

[10] J. Conway. The game of life. *Scientific American*, 223(4):4, 1970.

[11] G. A. Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958.

[12] K. F. Doerner, R. F. Hartl, S. Benkner, and M. Lucka. Parallel cooperative savings based ant colony optimization—multiple search and decomposition approaches. *Parallel processing letters*, 16(03):351–369, 2006.

[13] M. Dorigo. Optimization, learning and natural algorithms. *PhD Thesis, Politecnico di Milano*, 1992.

[14] M. Dorigo and G. Di Caro. Ant colony optimization: a new meta-heuristic. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 2, pages 1470–1477. IEEE, 1999.

[15] M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66, 1997.

[16] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.

[17] I. Ellabib, P. Calamai, and O. Basir. Exchange strategies for multiple ant colony system. *Information Sciences*, 177(5):1248–1264, 2007.

[18] J. Fu, L. Lei, and G. Zhou. A parallel ant colony optimization algorithm with gpu-acceleration based on all-in-roulette selection. In *Third International Workshop on Advanced Computational Intelligence*, pages 260–264. IEEE, 2010.

[19] D. Gaertner and K. L. Clark. On optimal parameters for ant colony optimization algorithms. In *IC-AI*, pages 83–89, 2005.

[20] C. Hewitt, P. Bishop, and R. Steiger. Session 8 formalisms for artificial intelligence a universal modular actor formalism for artificial intelligence. In *Advance Papers of the Conference*, volume 3, page 235. Stanford Research Institute, 1973.

[21] S. Ilie and C. Bădică. Multi-agent approach to distributed ant colony optimization. *Science of Computer Programming*, 78(6):762–774, 2013.

[22] D. Krzywicki, W. Turek, A. Byrski, and M. Kisiel-Dorohinicki. Massively concurrent agent-based evolutionary computing. *Journal of Computational Science*, 11(Supplement C):153 – 162, 2015.

[23] M. Lucka and S. Piecka. Parallel posix threads based ant colony optimization using asynchronous communication. In *Proceedings of the 8th International Conference on Applied Mathematics*, volume 2, pages 229–236, 2009.

[24] F. Marini and B. Walczak. Particle swarm optimization (pso). a tutorial. *Chemometrics and Intelligent Laboratory Systems*, 149:153–165, 2015.

[25] J. A. Mocholí, J. Jaen, and J. H. Canos. A grid ant colony algorithm for the orienteering problem. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 942–949. IEEE, 2005.

[26] M. Pedemonte and H. Cancela. A cellular ant colony optimisation for the generalised steiner problem. *International Journal of Innovative Computing and Applications*, 2(3):188–201, 2010.

[27] M. Pedemonte, S. Nesmachnow, and H. Cancela. A survey on parallel ant colony optimization. *Applied Soft Computing*, 11(8):5181–5197, 2011.

[28] M. Rahoual, R. Hadji, and V. Bachelet. Parallel ant system for the set covering problem. In *International Workshop on Ant Algorithms*, pages 262–267. Springer, 2002.

[29] A. Sameh, A. Ayman, and N. Hasan. Parallel ant colony optimization. *International Journal of research and reviews in Computer Science*, 1(2):77, 2010.

[30] G. Skiba, M. Starzec, A. Byrski, K. Rycerz, M. Kisiel-Dorohinicki, W. Turek, D. Krzywicki, T. Lenaerts, and J. C. Burguillo. Flexible asynchronous simulation of iterated prisoner's dilemma based on actor model. *Simulation Modelling Practice and Theory*, 83:75–92, 2018.

[31] R. Skinderowicz. The gpu-based parallel ant colony system. *Journal of Parallel and Distributed Computing*, 98:48–60, 2016.

[32] M. Starzec, G. Starzec, A. Byrski, and W. Turek. Distributed ant colony optimization based on actor model. *Parallel Computing*, 90:102573, 2019.

[33] M. Starzec, G. Starzec, A. Byrski, W. Turek, and M. Kisiel-Dorohinicki. Distributed ant system for difficult transport problems. *Journal of Intelligent & Fuzzy Systems*, 37(6):7347–7356, 2019.

[34] M. Starzec, G. Starzec, A. Byrski, W. Turek, and K. Pietak. Desynchronization in distributed ant colony optimization in hpc environment. *Future Generation Computer Systems*, 2020.

[35] M. Starzec, G. Starzec, and M. Paciorek. Desynchronization of simulation and optimization algorithms in hpc environment. *Computer Science*, 21(3), 2020.

[36] T. Stützle and H. H. Hoos. Max–min ant system. *Future generation computer systems*, 16(8):889–914, 2000.

[37] W. Turek, L. Siwik, and A. Byrski. Leveraging rapid simulation and analysis of large urban road systems on hpc. *Transportation Research Part C: Emerging Technologies*, 87:46 – 57, 2018.

[38] W. Turek, J. Stypka, D. Krzywicki, P. Anielski, K. Pietak, A. Byrski, and M. Kisiel-Dorohinicki. Highly scalable erlang framework for agent-based metaheuristic computing. *Journal of Computational Science*, 17(Part 1):234 – 248, 2016.

[39] C. Twomey, T. Stützle, M. Dorigo, M. Manfrin, and M. Birattari. An analysis of communication policies for homogeneous multi-colony aco algorithms. *Information Sciences*, 180(12):2390–2404, 2010.

[40] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[41] X.-F. Xie, W.-J. Zhang, and Z.-L. Yang. Social cognitive optimization for nonlinear programming problems. In *Proceedings. International Conference on Machine Learning and Cybernetics*, volume 2, pages 779–783. IEEE, 2002.

[42] X.-S. Yang. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[43] Y. Zhou, F. He, N. Hou, and Y. Qiu. Parallel ant colony optimization on multi-core simd cpus. *Future Generation Computer Systems*, 79:473–487, 2018.

[44] Y. Zhou, F. He, and Y. Qiu. Dynamic strategy based parallel ant colony optimization on gpus for tsps. *Science China Information Sciences*, 60(6):68102, 2017.

Scientific curriculum

Scientific degrees obtained

- B.Sc., 2015, title: *Automated testing framework for devices supporting CWMP protocol*, supervisor: Wojciech Czech, Ph.D.
- M.Sc., 2017, title: *Asynchronous simulation of iterated prisoner's dilemma using actor model*, Co-authored with G. Skiba, supervisor: Aleksander Byrski, Ph.D., D.Sc. The dissertation was awarded with a distinction at the AGH University of Science and Technology competition "AGH Diamonds" for the best M.Sc. dissertations.

Publications

- G. Skiba, **M. Starzec**, A. Byrski, K. Rycerz, M. Kisiel-Dorohinicki, W. Turek, D. Krzywicki, T. Lenaerts, J. C. Burguillo: *Flexible asynchronous simulation of iterated prisoner's dilemma based on actor model*. In: Simulation Modelling Practice and Theory: International Journal of the Federation of European Simulation Societies, 2018 vol. 83, s. 75–92.
  IF=2.426, MNiSW=25
- **M. Starzec**, G. Starzec, A. Byrski, W. Turek: *Distributed ant colony optimization based on actor model*. In: Parallel Computing, 2019 vol. 90 art. no. 102573, s. 1–9.
  IF=1.281, MNiSW=70
- **M. Starzec**, G. Starzec, A. Byrski, W. Turek, M. Kisiel-Dorohinicki: *Distributed ant system for difficult transport problems*. In: Journal of Intelligent & Fuzzy Systems, 2019 vol. 37 iss. 6, s. 7347–7356.
  IF=1.637, MNiSW=70
- G. Starzec, **M. Starzec**, A. Byrski, M. Kisiel-Dorohinicki, J. C. Burguillo, T. Lenaerts: *Towards large-scale optimization of iterated prisoner dilemma strategies*. In: Transactions on computational collective intelligence XXXII / eds. Ngoc Thanh Nguyen, Ryszard Kowalczyk, Marcin Hernes. — Berlin : Springer-Verlag GmbH, cop. 2019.
  MNiSW=16.33

- **M. Starzec**, G. Starzec, A. Byrski, W. Turek, K. Piętak: *Desynchronization in distributed Ant Colony Optimization in HPC environment.* In: Future Generation Computer Systems, 2020 vol. 109, s. 125–133.
  IF=5.768, MNiSW=140

- **M. Starzec**, G. Starzec, M. Paciorek: *Desynchronization of simulation and optimization algorithms in HPC Environment.* In: Computer Science 21(3) 2020: 307-322
  MNiSW=20

Participation in research projects

- cybercrypt@gov – distributed, scalable, high-performance system allowing for breaking cryptographic security (since 2020).

Reviewer activity

- Reviewer of Computer Science Journal, published by AGH University Press, indexed by, i.a., SCOPUS and ESCI Web of Science.

- Reviewer of Journal of Ambient Intelligence and Humanized Computing, published by Springer.

Teaching activity

- 2020 (spring semester): Object- and component-oriented systems, LLP Erasmus students, AGH University of Science and Technology International Courses, 2020.

- Support in realization of M.Sc. thesis by Wojciech Radwański and Krzysztof Węgrzyński.

Bibliometry

- H-index: 3 (Google scholar).

- 6 papers published (4 in JCR journals).

- 20 citations registered (Google scholar).

- Total IF: 11.113.

- Total MNiSW points: 341.33.