MANGO: A MultiAgent ENvironment for Global Optimization*

Lirida Kerçelli
Department of Computer Engineering
Boğaziçi University
TR-34342 Bebek
Istanbul, Turkey
lirida.kercelli@gmail.com

Ayşe Sezer
Department of Computer Engineering
Boğaziçi University
TR-34342 Bebek
Istanbul, Turkey
sezer.ayse@gmail.com

Figen Öztoprak
Department of Industrial Engineering
Sabancı University
Orhanlı/Tuzla 34956
Istanbul, Turkey
figen@su.sabanciuniv.edu

Pınar Yolum
Department of Computer Engineering
Boğaziçi University
TR-34342 Bebek
Istanbul, Turkey
pinar.yolum@boun.edu.tr

Ş. İlker Birbil
Department of Industrial Engineering
Sabancı University
Orhanlı/Tuzla 34956
Istanbul, Turkey
sibirbil@sabanciuniv.edu

ABSTRACT

Hard global optimization problems arise in many areas of engineering. However, solving these problems is a rather difficult task. Most of the existing methods work in isolation and aim at solving specific problems only. The literature encourages parallel hybrid implementation of those existing methods for solving hard global optimization problems efficiently and effectively. The *parallel hybrid* idea leads to a certain type of cooperative agent-based optimization approach that we follow in the design of MANGO (MultiAgent eNvironment for Global Optimization). MANGO is a project in progress that involves the development of an extensible and flexible multiagent platform, in which autonomous agents can solve global optimization problems in cooperation. Each agent in the proposed system executes an optimization algorithm in itself and has some specific cooperative behavior. The system provides cooperation protocols as well as allows flexible organization models and registry of new agents and their services. This paper presents the details of the MANGO project and provides an illustrative example.

Keywords

multi-agent systems, global optimization problems, cooperative agent-based optimization

1. INTRODUCTION

*This research has been partially supported by the Scientific and Technological Research Council of Turkey by a CAREER Award under grant MAG-107M455.

Solving hard global optimization problems that arise in various research and application areas, such as engineering design, telecommunications, molecular biology, and so on, has been a challenging issue for many years. A number of *deterministic* and *stochastic* solution methodologies have been proposed for *discrete* and *continuous* global optimization problems with various characteristics. However, we still do not have the ultimate methodology that can solve all the problems effectively. The deterministic methods guarantee to find the exact but only a local solution of a global optimization problem under a set of usually restrictive assumptions. Stochastic and heuristic methods are based on a *strategic random search* and they usually provide a satisfactory solution to those problems, for which deterministic methods do not perform well, but generally they cannot guarantee convergence [6]. To solve hard global optimization problems effectively and efficiently, hybrid methods, parallel algorithms, and agent-based approaches have also been proposed with close or complementary ideas behind.

1.1 Problem Definition

Given a nonempty set $S \subset R^n$ and a function $f : S \rightarrow R$, a global optimization problem can be written as

$$\begin{aligned} &min f(x) \\ &x \in S. \end{aligned} \tag{1}$$

Here, S is the *feasible region* and f is the *objective function* of the global optimization problem. The aim is finding at least one *global minimizer*, $x^* \in S$, such that $f(x^*) \leq f(x)$ for all $x \in S$. A global optimization problem may have many local optimum solutions. A *local minimizer* $x^* \in S$ satisfies $f(x^*) \leq f(x)$, for all $x \in B(x^*, \epsilon)$ for a given $\epsilon > 0$, $B(x^*, \epsilon)$ indicating the ϵ neighborhood of x^*. The global minimizer is also a local minimizer, but the reverse is not correct.

In this work, we concentrate on a specific class of (1), where the feasible region is defined as $S = \{x \in R^n : l \leq x \leq u\}$ so that solution points may have continuous values in R^n between a lower bound $l \in R^n$ and an upper bound $u \in R^n$. To the best of our knowledge, there does not exist a generic way of dividing

this problem into smaller subproblems. In many real-life instances, even an analytical form of the objective function is unknown and it is not possible to compute how far any solution is away from the global optimum. It is hard to evaluate how good any obtained solution is; the success in solving a global optimization problem is generally evaluated comparatively and expressed in terms of the *number of function evaluations*.

1.2 Related Work

An optimization problem is about finding the points at which a given objective function attains its optimum value. These points are usually required to be from a set of points called the *feasible region*. In many cases, the performance of a solution approach is problem-dependent. That is, the quality of the obtained results vary based on whether the problem has some desirable structure, such as low dimension, continuous differentiability and convexity. The lack of a completely generic solver has motivated the design of several *hybrid* algorithms that combine deterministic methods with stochastic and heuristic approaches. Some of these hybrid algorithms are reported to perform quite well. In addition, the performances of these hybrid algorithms can be improved further, particularly in execution time, by *parallelization* [19, 8, 1]. It has also been argued that parallelization may provide more than the speed up in execution time when parallel processors execute independent heuristics concurrently and cooperate during their search [10, 11]. These results lead to a certain type of *agent-based optimization* approach that we follow in this work.

A *parallel or distributed computing system* consists of a number of processing units that execute a given task together. In the distributed case, processors may be geographically dispersed with an unreliable communication network and an unstable system architecture [5]. Such a system can be viewed as a *distributed problem solving* (DPS) system: a community of cooperative *agents* that are "unable to accomplish their own tasks alone or at least accomplish their tasks better when working with others" [12]. Durfee and Rosenschein [13] distinguish between DPS and multiagent systems (MAS) in the way they view an agent. DPS community focuses at the problem at hand, that is the agents are just programmed units that execute their assigned tasks and cooperate in the way they are told. However, this approach cannot reflect the behavior in social systems. On the other hand, MAS community focuses on agents and assumes that the agents are autonomous software that can perceive, reason, and act.

Several approaches in the literature involve cooperation of agents for solving hard optimization problems. An *asynchronous team* (A-team) is defined as a set of autonomous agents and a set of memories that are connected through a cyclic network. There is no central coordination or planning mechanism. Each agent applies some algorithms or modification operations on the solutions selected from its input-memory. To provide cooperation, input and output memories of agents are connected through communication channels so that an agent may select the output of another agent as its input. The system terminates when a persistent solution is obtained [20]. The approach has been successfully adapted for and applied to numerous problems, where among these are nonlinear large-scale optimization problems [22, 18]. To support the implementation of the approach, a general-purpose JADE-based A-team environment has been developed [2]. However, this environment is naturally restricted in the sense that A-teams are specifically designed for solving combinatorial optimization problems and it provides a certain type of hybridization mechanism. Hence, A-team cannot readily be used to solve any global optimization problem as MANGO aims to do.

Similarly, the main idea of the *distributed constraint satisfaction* approach is that individual agents have limited capabilities so they can only partially contribute to the solution of the global problem. However, cooperation mechanism is different. In this approach, each agent owns a constraint or a variable and the values of the variables are assigned in coordination [23]. The approach has been applied to a variety of combinatorial optimization problems and several variants and extensions have been developed [16, 15].

We should also mention here some population-based heuristics in which the *population* behaves like a team of collaborating agents and the collaboration mechanism is generally inspired by a natural phenomenon. The ant colony approach, for example, suggests that simple interactions among simple agents may produce a very well-structured organizational behavior. The agents are not even aware of the overall system they are working for, what other agents are doing or how their interaction effects the system behavior [9]. Among other examples are *particle swarm optimization* [14], *electromagnetism-like mechanism* [7] and *evolutionary diffusion optimization* [21].

1.3 Overview of MANGO

In this paper we present our project entitled **M**ulti**A**gent E**N**vironment for **G**lobal **O**ptimization (MANGO). MANGO is a platform that supports the development of multi-agent system architectures for solving global optimization problems. The optimization problems solved by MANGO are not divided into sub-problems so that different agents can solve different parts of the problem. Instead, each agent solves the same whole problem definitions loaded on the system. Similar to the A-teams approach, agents in the proposed environment are autonomous and asynchronously execute deterministic or stochastic, single-point or multi-point optimization methods. But unlike A-teams, the cooperation mechanism is quite flexible to allow any type of hybrid implementation. The cooperation is via messaging and generic cooperation protocols are designed and used. The agents introduce themselves to the system through a registry mechanism but no agent exactly knows the algorithms other agents own. Through communication, agents share partial solutions and coordinate their search.

The rest of this paper is organized as follows. Section 2 explains the system architecture of MANGO with an emphasis on its design, implementation and communication mechanisms. Section 3 shows a proof-of-concept study to demonstrate how MANGO can be used for global optimization. Section 4 summarizes our work on MANGO and future directions.

2. SYSTEM ARCHITECTURE

MANGO is a JADE-based middleware that simplifies the implementation of agents and provides a distributed environment for hosting global optimization agents and enabling communication between them. JADE is a software platform for the development and run-time execution of peer-to-peer applications that are based on the agents paradigm [4]. The following sections give a detailed design of the system, and explain the system's current implementation.

2.1 System Design

The proposed system provides a distributed environment for agents to execute their own search algorithms and at the same time, com-

municate and collaborate with each other via messaging. Several characteristics are considered in the system design.

Usability: MANGO is designed to support the development of distributed MAS for global optimization by users with little agent programming experience. A user accesses the environment over the Web and loads the executable of an agent on the environment. MANGO provides a medium for the agent to be run on the user machine (which requires the JADE platform and JVM be installed on the machine), or on its servers. Ongoing work involves the development of a friendly user interface for displaying the current agents in the system, the interactions between the agents and their current outputs.

Autonomous Cooperation: The agents in our system are autonomous. If they desire, they can cooperate with other agents in the system. They communicate with each other by sending and receiving messages. The communication is done asynchronously, so each agent has a mailbox for its messages to be stored when they arrive, until the agent decides to process them. Using a generic protocol, an agent may share its solutions to problems, request other agents in the system to perform particular searches for it as well as provide requested services to other agents. The aim of the platform is to allow (and encourage) collaboration between agents, so that better solutions can be found faster.

Flexible Organization Models: Depending on the design choice of the user, agents in the system can form diverse organizations. MANGO is flexible enough to host a variety of agent organizations such as: master-slave, team, hierarchical, peer-to-peer, or just agents that only search and do not communicate, or a combination of these organizations.

2.2 Implementation

JADE is used to implement the backbone of the environment and the multiagent systems. JADE is a middleware that simplifies the implementation of MAS and is compliant with the FIPA specifications. It is a software framework that hides all complexity of the distributed architecture to application developers, who can focus their software development on the logic of optimization MAS rather than on middleware issues, such as discovering and contacting the entities of the system [4]. JADE comes with a set of facility agents such as Agent Management System (AMS) and Directory Facilitator (DF). The Agent Management System (AMS) is the agent that supervises the entire platform. It is the contact point for all agents that need to interact in order to access the white pages of the platform as well as to manage their life cycle. Consistent with the FIPA specifications, each agent instance is identified by an "agent identifier" (AID). Every agent is required to register with the AMS (automatically carried out by JADE at agent startup) in order to obtain a valid AID. The Directory Facilitator (DF) is the agent that implements the yellow pages service, used by any agent wishing to register its services or search for other available services. The JADE DF also accepts subscriptions from agents that wish to be notified whenever a service registration or modification is made that matches some specified criteria [3]. Each agent in our system is implemented by extending the JADE agent template. The agents have to implement the `setup()` method which is intended to include agent initializations. Methods for registering to and deregistering from DF are provided together with the agent template. The actual job an agent has to perform is typically carried out within "behaviours". A behaviour represents a task that an agent can carry out. For an agent to execute a behaviour, the behaviour must be added to the agent (in the `setup()` method) by means of the `addBehaviour()` method. Agents live in containers which are the Java process that provides the JADE run-time and all the services needed for hosting and executing agents [3]. Figure 1 shows the life cycle of an agent in our system.

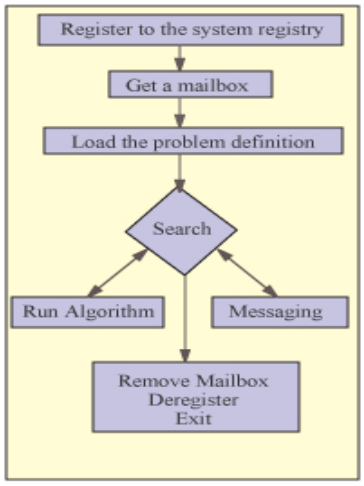

Figure 1: Life cycle of an optimization agent

When a new agent enters MANGO, a new container is created for the agent to execute. A mailbox is created for the agent to store the incoming messages. The agent is also automatically registered to AMS and to DF by the services it provides. The services here correspond to the type of the optimization algorithm an agent executes. For instance, an agent executing a local search by using the gradient information provides a certain type of service. The agent loads the problem definition which specifies the kind of problem to be solved. When the agent starts execution, it runs its search algorithm and communicates with other agents when necessary. When the `setup()` method of the agent finishes execution, the agent exits the system. Its container and mailbox are removed and it is also deregistered from the AMS and DF.

2.3 Communication

The communication paradigm in JADE is based on asynchronous message passing. Thus, each agent has a "mailbox" (the agent message queue) where the JADE run-time posts messages sent by other agents. Whenever a message is posted in the mailbox message queue the receiving agent is notified. However, when, or if, the agent processes the message in the queue is a design choice of the agent programmer.

To support a successful cooperation between optimization agents, MANGO agents send two types of messages: *solution type* and *coordination type*. Each agent may need information about a *solution*, or about a *search space*.

Solution type messages are sent when agents want to inform other agents about a current solution they have found about a problem or to request other agents for their solutions. The other agents may use this solution so they do not need to search an area to find that solution, thus reducing the work done on searching. For a solution type message, the agent needs to send a Solution message content, which is composed of a SearchSpace (an array of Iterates) and a

Cost (an array storing the number of a particular search method calls, number of times first derivative is taken, and number of times second derivative is taken during that search). Each Iterate is composed of a vector and a value, corresponding to the coordinates of a mini-search area and the solution found from there.

Coordination type messages are sent when an agent wants the other agents to *explore*, i.e. to search a specific search space for that agent, or *refrain* the other agents from searching that space. For a coordination type message, the agent needs to send a SearchSpace message content, which is composed of Iterates, where the whole or some specific searched area is stored.

The communication protocol supports the methods to exchange the above messages. All types of messages can be sent to some specific agent(s), to agents providing some specific service(s), or can be broadcasted to all the agents currently running in the system. The communication protocol methods are implemented as behaviours in the JADE platform. Agents that are willing to use any of these protocol behaviours have to add them explicitly in their `setup()` method. In this way, the implementation of the communication protocol is made easy for the developer of an agent.

There are two main types of protocol behaviours: the sender type behaviour, which prepares and sends a message, and the receiver type behaviour, which receives the proper message. The sender protocol behaviours take as argument the message content (i.e the Solution type message), the sender AID, the receiver AIDs, a list of services and a choice. They are designed as generic behaviours, where according to the choice that the sender agent makes, the message can be sent to specific agent(s), to agent(s) registered to DF as providing the services given as an argument to the behaviour, or the message can be broadcasted to all the active searching agents in the system. The behaviour prepares the message package by specifying the language and ontology used, as well as the conversation ID, and the message performative (i.e. INFORM). According to the choice made, either the AMS or the DF is searched for agent IDs, and in each case a list of agent IDs is returned. Then the message is sent to the agent IDs in this list. InformSolution and RequestSolution send Solution type messages, whereas Explore and Refrain send SearchSpace type messages. The receiver protocol behaviours take as an argument only the agent that adds them. Only the messages matching the performative INFORM and the conversation ID set previously in the corresponding sender behaviour are received. The language and ontology of the received messages is also controlled. These are the sender and their corresponding receiver protocol behaviours:

InformSolution-GetInform inform other agents about the current (best) solution of an agent.

RequestSolution-GetRequestSolution request the current (best) Solution of other agents.

Explore-GetExplore invite other agents to explore a particular search space.

Refrain-GetRefrain signal other agents not to explore a particular search space. This can be thought as the dual of Explore behaviour.

All the message performatives are set as INFORM, because agents are free to decide what should be done with the received messages, and whether to reply to the sender or not. Communication by messaging preserves the autonomy of the agents.

2.4 Interaction Scenarios

In this section we illustrate how communication between agents plays an important part while optimization problems are solved in MANGO using two example scenarios. Consider a "Genetic Algorithm" (GA) type agent, meaning that this agent can perform search using a genetic algorithm. While searching, the agent wants to inform other GA type agents about the solution it has found (Figure 2). The agent first searches the DF to find the other agents of its type, and sends the message containing the solution to those agents. The other agents receive the message and decide whether to process the message in their subsequent iterations. For the second scenario

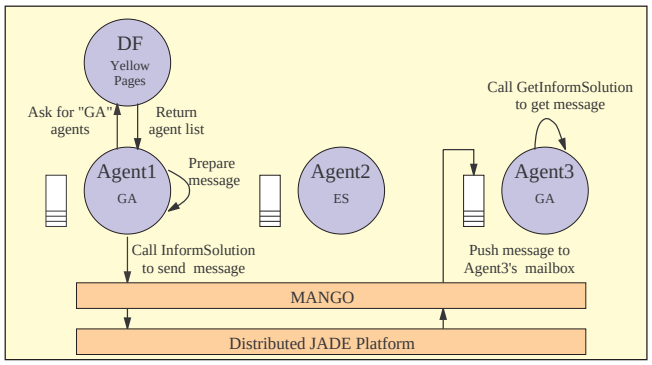

Figure 2: Scenario 1

again consider a "Genetic Algorithm" type agent (Figure 3). This time, the agent is in need of a service from an agent, which provides an exact local search procedure. Hence, it queries the DF to find the agents providing this service. If there are any "Exact Search" (ES) agents, it sends a message from the Explore behaviour with the intended search space. The agents that receive the message are free to decide how to respond to the message (i.e. explore and return an answer, or ignore the message, and so on).

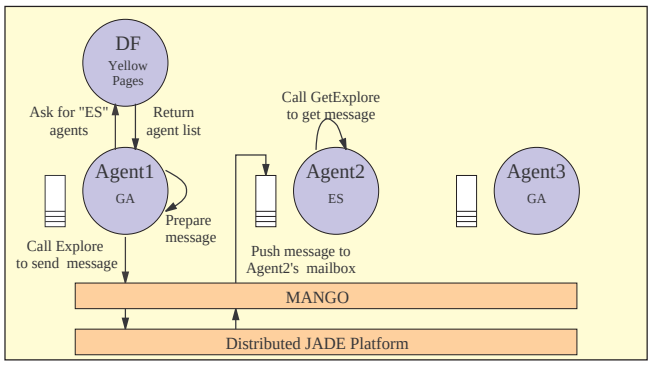

Figure 3: Scenario 2

3. PROOF OF CONCEPT

In this section, we illustrate the basic mechanism of MANGO on a relatively simple cooperative agent-based optimization system example. The system is composed of three homogeneous agents

and a single type of cooperation behavior. Each agent implements the Electromagnetism-like Mechanism (EM) heuristic with identical parameters. In this example, our agents intrinsically know the definition of the problem to be optimized.

To be able to describe the cooperation behaviour of EM agents, we should first briefly explain the heuristic they implement. EM is a population-based stochastic global optimization method that was originally proposed for solving continuous global optimization problems with bound constraints [7]. The algorithm starts with an initial solution set (particles) and then, an attraction-repulsion mechanism is used iteratively to move those particles towards optimality. Four basic procedures are applied: Initialize(), Local(), CalcF(), and Move(). In Initialize(), m particles are randomly located in the feasible space of the problem. In each iteration, these particles (except the best one) Move() by a random step length in the direction of the total force vector calculated by CalcF(). As in the attraction-repulsion mechanism of the electromagnetism theory, the force exerted on a particle via other particles is inversely proportional to the distance between the particles and directly proportional to the product of their charges. The best particle attains the maximum charge and the signs of the force vectors are assigned by comparing the objective function values of the related particles. Thus, a particle with a better objective function value attracts the other particles with a positive sign force vector and a particle with a worst objective function value repels other particles. Local() is applied to the particle with the best objective function value to let it search its neighborhood for locations with better function values. The algorithm terminates when MAXITER iterations are completed.

A common problem of the global optimization algorithms is the risk of being trapped in a local minimum point and missing the global minimizer of the problem. The cooperation behavior applied in this example system is based on a strategy that has been developed to avoid this situation. If an agent identifies a local minimum during its search, it fixes one of its particles there. Unlike the standard particles of the original method, the fixed particles do not move and search; they just repel mobile particles to direct them towards the yet undiscovered parts of the solution space. The agent also broadcasts any local minimizer it finds, so other agents in the system can also mark this point and do not approach there (Algorithm 1). Therefore, none of the agents spends its resources for converging to a local minimum that has already been identified by one of the agents in the system.

The identical agents of our example start applying EM generating different random number sets, so they may concurrently search different parts of the feasible space of their common problem. While they proceed, they cooperate whenever they find a new local minimum point as explained above. After termination, one of the agents (the organizer) collects the results obtained by each agent and calculates the final common performance values, i.e., the GatherResults() procedure.

We test our system by solving *Perm(4,0.005)*, a hard global optimization test problem selected from the set in [17]. This is a four-dimensional continuous bound constrained minimization problem that has many local minima, so that small variations in variable values may cause huge differences in function values and there is a considerable risk of being trapped in a local minimum point instead of converging to the global minimizer.

Algorithm 1 Modified EM Algorithm for cooperative agents

```
m: number of sample points
MAXITER: maximum number of iterations

Initialize()
iteration ← 1
while iteration < MAXITER do
  Local()
  F ← CalcF()
  Move(F)
  for all i ∈ set of Particles do
    if Particle_i has moved to a local minimum point then
      Fix(Particle_i)
      InformSolution(solution in Particle_i)
    end if
  end for
  GetInformSolution()
  if a new set solution vector S is received then
    Add a new fixed Particle located on S to the set of Particles
  end if
  iteration ← iteration + 1
end while
GatherResults()
```

Table 1: Message exchange among agents

Message	Sender-Iter.	Receiver1-Iter.	Receiver2-Iter.
Message 1	EMA1 - 81	EMA3 - 63	EMA2 - 83
Message 2	EMA2 - 94	EMA1 - 94	EMA3 - 76

We name our agents as EMA1, EMA2, and EMA3. Each agent searches the feasible space with 25 particles. The message traffic among agents during a run of the system is summarized in Table 1. In iteration 81, EMA1 finds a local minimum and broadcasts it via *Message 1*. EMA3 receives this message in its 63rd iteration and EMA2 receives the same message in its 83rd iteration. In iteration 94, EMA2 finds another local minimum and broadcasts it by *Message 2*. EMA1 is in iteration 94 and EMA3 is in iteration 76 when *Message 2* is received.

Table 2: Final output

	EMA1	EMA2	EMA3
Final objective function value	1.5688	0.0247	1.5688
Number of function evaluations	644	644	18094
Convergence to the global minimum	no	no	no

In Table 2, the final output of the system is reported in detail. Even though EMA2 obtains a value near to the global minimum, none of the agents can come up with global optimal solution after 400 iterations. Sending every solution to other agents leads to a load on the system, but by cooperation, i.e. by sharing results in this case, the possibility of finding the optimal solution increases.

As this illustrative example indicates, the cooperation mechanism in MANGO is different from the A-team architecture. In A-teams agents cooperate by modifying one another's trial-solutions and these solutions circulate continually [20]. Unlike A-teams, cooperation in MANGO is made explicitly by messaging. According

to their design, agents work autonomously and can find the results by their own. They do not depend on the outputs of other agents. The main idea is that cooperation between agents improves the solutions found and the speed they are found. A-teams are not the appropriate architecture to solve the types of global optimization problems that MANGO aims to solve. Similar to A-teams, agents in MANGO are autonomous and asynchronously execute different searching algorithms, thus providing different services that can be used by other agents.

4. CONCLUSION

Hard global optimization problems arise in many areas of engineering. To solve these problems effectively and efficiently, hybrid methods, parallel algorithms, and agent-based approaches have been proposed. In this work, we concentrate on a specific class of global optimization problem (1). To the best of our knowledge, there is not a generic way of dividing these problems into smaller subproblems. It is hard to evaluate how good any obtained solution is; the success in solving a global optimization problem is generally evaluated comparatively and expressed in terms of the *number of function evaluations*. We follow a cooperative agent-based optimization approach to give solutions to this class of problems and present our project entitled MANGO (MultiAgent ENvironment for Global Optimization). MANGO is a JADE-based middleware that simplifies the implementation of agents and provides a distributed environment for hosting global optimization agents and enabling communication between them. The system is designed to meet several requirements such as usability, autonomous agent cooperation and flexible organization models. The agents in the system are autonomous. Depending on the design choice of the user, agents in the MAS can form diverse organizations. The main advantage of MANGO inherited from JADE is the simplification of the implementation of MAS by hiding all the complexity of the distributed architecture to application developers, who can focus on the logic of optimization agents rather than on handling middleware issues.

The main idea of the proposed solution is that cooperation between agents improves the solutions found and the speed they are found. MANGO provides a cooperation mechanism that is quite flexible to allow any type of hybrid implementation. The cooperation is via messaging and generic cooperation protocols are developed. The agents introduce themselves to the system through a registry mechanism but no agent exactly knows the algorithms other agents own. Through communication, agents share partial solutions and coordinate their search. Messaging leads to a load on the system, but by cooperation the possibility of finding the optimal solution increases. Future research will focus on providing a friendly user interface and improving the usability of the system by further supporting the development of distributed MAS. The functionality of agents can be broadened by developing techniques for learning. This would improve the overall performance of the system in finding better solutions to the global optimization problems.

5. REFERENCES

[1] B. Baran, E. Kaszkurewicz, and A. Bhaya. Parallel asynchronous team algorithms: Convergence and performance analysis. *IEEE Transactions on Parallel and Distributed Systems*, 7(7):677–688, 1996.

[2] D. Barbucha, I. Czarnowski, P. Jedrzejowicz, E. Ratajczak, and I. Wierzbowska. Jade-based a-team as a tool for implementing population based algorithms. In *Proceedings of ISDA'06*, 2006.

[3] F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley and Sons, 2007.

[4] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa. Jade: A white paper. *Exp*, 3(3):6–19, September 2003.

[5] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997.

[6] S.I. Birbil. *Stochastic Global Optimization Techniques*. PhD thesis, North Carolina State University, Raleigh, 2002.

[7] S.I. Birbil and S.-C. Fang. An electromagnetism-like mechanism for global optimization. *Journal of Global Optimization*, 25(3):263–282, 2003.

[8] S. Cahon, N. Melab, and E.-G. Talbi. Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10:357–380, 2004.

[9] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In *Proceedings of European Conference on Artificial Life*, pages 134–142. Elsevier Publishing, 1991.

[10] T.G. Crainic and M. Gendreau. Cooperative parallel tabu search for capacitated network design. *Journal of Heuristics*, 8:601–627, 2002.

[11] T.G. Crainic, M. Gendreau, P. Hansen, and N. Miladenovic. Cooperative parallel variable neighborhood search for the p-median. *Journal of Heuristics*, 10:293–314, 2004.

[12] E.H. Durfee. Distributed problem solving and planning. In G.Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1990.

[13] E.H. Durfee and J.S. Rosenschein. Distributed problem solving and multi-agent systems: Comparisons and examples. In M. Klein, editor, *Proceedings of the 13th International Workshop on DAI*, pages 94–104, 1994.

[14] J. Kennedy and R.Eberhard. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[15] H. C. Lau and H. Wang. A multi-agent approach for solving optimization problems involving expensive resources. In *ACM Symposium on Applied Computing*, 2005.

[16] P.J. Modi, W. Shena, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180, 2005.

[17] A. Neumaier. Global optimization test problems. http://www.mat.univie.ac.at/ neum/glopt/test.html, 2007.

[18] D.S. Siirola, S.Hauan, and A.W. Westerberg. Toward agent-based process systems engineering: Proposed framework and application to non-convex optimization. *Computers and Chemical Engineering*, 27:1801–1811, 2003.

[19] E.-G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8:541–564, 2002.

[20] S. Talukdar, L. Baerentzen, A. Gove, and P. De Souza. Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, 4(4):295–321, 1998.

[21] K.C. Tsui and J. Liu. Evolutionary diffusion optimization, part i:description of the algorithm. In *Proceedings of Congr. Evolutionary Computation (CEC)*, pages 1284–1290, 2002.

[22] K. Tyner and A. Westerberg. Multiperiod design of azetropic separation systems i: An agent based approach. *Computers and Chemical Engineering*, 25:1267–1284, 2001.

[23] M. Yokoo and T. Ishida. Search algorithms for agents. In G.Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1990.