Multiagent Cooperation for Solving Global Optimization Problems: An Extendible Framework with Example Cooperation Strategies

Fatma Başak Aydemir, Akın Günay
Department of Computer Engineering, Bogazici University, Bebek, 34342 Istanbul, Turkey
Figen Öztoprak, Ş. İlker Birbil
Faculty of Engineering and Natural Sciences, Sabancı University, 34956 Istanbul, Turkey.
Pınar Yolum
Department of Computer Engineering, Bogazici University, Bebek, 34342 Istanbul, Turkey

Abstract: This paper proposes the use of multiagent cooperation for solving global optimization problems through the introduction of a new multiagent environment, MANGO. The strength of the environment lays in its flexible structure based on communicating software agents that attempt to solve a problem cooperatively. This structure allows the execution of a wide range of global optimization algorithms described as a set of interacting operations. At one extreme, MANGO welcomes an individual non-cooperating agent, which is basically the traditional way of solving a global optimization problem. At the other extreme, autonomous agents existing in the environment cooperate as they see fit during run time. We explain the development and communication tools provided in the environment as well as examples of agent realizations and cooperation scenarios. We also show how the multiagent structure is more effective than having a single nonlinear optimization algorithm with randomly selected initial points.

Keywords: Multiagent Systems, Global Optimization, Cooperation

1. Introduction. Global optimization is a broad term defining the problem of finding the minimum of a given function on a given domain of feasible solutions. Global optimization problems are known to be NP-hard, thereby deterministic solution methods may not be suitable unless the problem is relatively small. While it is difficult in general to guarantee providing exact solutions or to evaluate whether a given solution is close to being globally optimal, it is still necessary to attack these problems effectively since they arise in many diverse application areas such as molecular distance geometry, neural network training, and space trajectory optimization. Important reviews of global optimization, its popular solution methodologies and applications can be found elsewhere [24, 30, 29].

In this study, we consider global optimization problems given by

$$\begin{aligned} &\text{minimize} && f(x), \\ &\text{subject to} && x \in \mathcal{F}, \end{aligned} \tag{1}$$

where the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is a continuous non-convex function without a special structure, and the feasible set $\mathcal{F} \subseteq \mathbb{R}^n$ is defined by simple bounds on the components of decision variable x. These problems are important to examine because for unconstrained real-life problems it is generally possible to derive bounds on the values of decision variables. Also, for problems with functional constraints, the problems of the form (1) may arise as sub-problems of upper level solution methodologies. Solving a general global optimization problem clearly requires an extensive search on the set of feasible solutions. Thus, it is very suitable for the application of various decomposition schemes, which can be described by a group of tasks and task interrelationships.

We start by considering the case of population-based heuristics. In its simplest form each individual of the population has simple identical tasks —such as a single function evaluation. However, the interaction mechanism can be quite sophisticated, e.g.,[5, 11, 7]. The next idea would be to define identical but possibly more functional tasks, as in the case of a multi-start strategy that executes several local searches

concurrently starting from different points and finally selects the solution with the minimum objective function value. This approach corresponds to a scheme where several instances of an algorithm run independently and share information in a prescribed way. It has been argued that the resulting (parallel) algorithms may perform better in terms of solution quality thanks to the cooperation among individual search instances [12, 6].

Search algorithms may perform differently on a given problem instance; they can be very successful on some problems and may fail to make progress on some others. Thus, an idea following the above scheme would be to run different search algorithms in parallel and let them cooperate. This is an approach that is referred as parallel hybrid algorithms. In this case, the tasks are no more identical, but the interaction is again predefined. This scheme allows several possibilities to combine deterministic methods with stochastic and heuristic approaches, and some examples are reported to perform quite well [36, 21, 16].

While parallel hybrid algorithms cover various combinations of algorithms, the interactions between the algorithms have several drawbacks:

- ◇ Lack of heterogeneity: The algorithms generally exchange the same message with predefined entities. The message could be the best solution found or an area to be searched. However, differentiations based on the identities of the participants are not, and cannot be made.
- ◇ Lack of autonomy: The algorithms are expected to handle the messages that are received in the same way. An algorithm cannot autonomously decide that it is going to ignore a certain message from a particular algorithm.
- ◇ Lack of flexibility: The interaction is limited to the predefined patterns enforced by the system designer. An algorithm cannot decide to send a different type of message at run time. We believe that a more adaptive and efficient scheme would be possible by allowing the interaction to emerge during the solution process. For example, an algorithm that does not usually send out the solutions it has found, may decide at run time to do so, because it has found a really promising point.

In order to overcome these limitations, we propose to embody each algorithm inside a software agent. Agents are autonomous computations that can perceive their environment, reason, and act on it as well as communicate with others accordingly [15]. Agents are most useful when they exist in a multiagent system so that they can inter-operate with other agents. Inter-operation requires agents to share a common communication protocol to coordinate their activities and to cooperate if they find it to their advantage. The autonomy of the agents implies that the agents can choose how and with whom they want to interact. This autonomy also enables freedom during execution. Agents need not have to be developed by the same people. Agents from different vendors can inter-operate in a suitable environment. An agent can be reactive or proactive, depending on the situation. Reactive agents respond to the events happening around it, whereas proactive agents decide what needs to be done and perform an action without being probed.

Since agents interact on demand, multiagent communication is usually asynchronous. That is, an agent sends a message to another when it feels necessary. Similarly, an agent receives a message during its execution rather than waiting for messages at predetermined points. Many times, a multiagent system can be viewed as an organization where each agent has a specific role. These roles can be assigned prior to execution or the agent can adapt a role during execution. The roles ease the execution of the system

by prescribing specific tasks or behaviors to agents [9]. These properties of a multiagent system make it an ideal candidate to realize teams of algorithms.

This paper has three contributions. First, we show how a global optimization algorithm can be embodied inside an agent to act autonomously and cooperatively with other agents. Second, we develop a multiagent environment (MANGO) for executing such agents. The MANGO environment provides the necessary utilities to develop agents that can participate in a multiagent system to solve global optimization problems as well as an extensible protocol for agents to communicate with each other. Third, we depict how multiagent cooperation can be used to tackle global optimization problems. Multiagent cooperation is shown in examples where agent teams are formed and exploited to solve global optimization problems in MANGO. The teams employ a cooperation scenario that specifies an interaction pattern among agents. MANGO is a distributed environment; therefore it suits well for implementing global optimization methods on a distributed system, as well as solving global optimization problems with distributed data. Also, MANGO is designed as a modular system so that different global optimization strategies can be employed by running the same MANGO agents with different roles or different interactive behaviors.

1.1 Examples. As pointed above, it is possible to describe most of the existing general global optimization methods by a group of concurrently executable tasks and task interrelationships. MANGO can be used for the execution of all such existing methods, as well as providing all the flexibility for the design of new global optimization strategies. The below three examples provide further clarification.

(i) As the first example, let us consider the well-known multi-start scheme. An existing approach is to maintain a history of local search results, and employ that information in choosing the starting points for the following local search runs. This scheme can be realized on MANGO by introducing a non-hierarchical system of agents, each of which runs a local search algorithm. The agent communication consists of sharing the local solutions obtained. Upon completion of a local search, the agent sends the final solution point to all the other agents as an information message. When an agent receives such a message, it adds the enclosed local solution point to its list of *known solutions.* Each agent uses that list of known solutions to decide on a point that will be used to start the next local search run.

(ii) Another well-known strategy that can be realized on MANGO is (parallel) branch-and-bound. This requires the design of agents with two different roles: a leader agent and a group of subordinate search agents. The leader partitions the search domain and sends request messages to the subordinate agents asking them to apply their search mechanism on the specific partitions it has assigned to them. The leader agent manages the search tree by using the replies of the subordinates—fathoms existing branches or starts new ones according to the results obtained in sub-domains. The overall search can start with a small number of active search agents, and the leader may ask more search agents to become active if there are more branches to be searched. This asynchronous communication among the leader and the search agents well suits to efficiently implement this as a multiagent system.

(iii) Finally, let us give an example on how MANGO can be used to design new strategies. We will consider again the local search agents of the multi-start example, but this time they are allowed to use the information they receive in different ways. Note that each agent can receive information

messages from others in the middle of the execution of their algorithms. Therefore, their lists of known solutions can change dynamically from one iteration of the local search algorithm to the next. The local search agents could use the additional information sent by the others to adapt dynamically to a model or update parameters they use, or to terminate a local search early and jump to another starting point. The main goal here is to explore better solutions of the global optimization problem at hand. Hence, it is reasonable to consider modifications of the local search methods in ways to improve their global performance. For example, a local search agent could use its list of known solutions to revise its step computation so that the steps are rotated towards other useful directions as the iterates approach to the points in the list. In this way, the agent could increase its chance of ending up at a different local solution. We will return to this example in Section 4 to discuss in detail how MANGO agents are suitable to design such extensions that aim to improve the performance of the global search.

1.2 Related Work. While multiagent systems have been used in many areas, their use in solving global optimization problems, as we discuss here, is new. There are a number of different studies on multiagent optimization systems. However, to the best of our knowledge, the most related idea to our work is that of asynchronous teams (A-teams). An 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 coordination or planning mechanism. Each agent applies an algorithm or a modification operation 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 [37]. The approach has been successfully adapted for and applied to particular problems, among these are some specific nonlinear optimization problems [32, 33]. To support the implementation of this approach, moreover, a general-purpose JADE-based A-team environment (JABAT) has been developed [2, 1]. However, our intuition in this work is significantly different than that of A-teams. In the A-teams environment, all agents employ the same interaction behavior, whereas, in this work, our aim is to enable agents to play different roles and have various interaction patterns. Moreover, in A-teams, the interaction among the agents does not affect the included algorithms at the control level, i.e., the cooperation applies through the input data of the algorithms but the original mechanisms of these procedures are not modified. However, in this work, agents can behave differently based on the messages that they receive.

Similar to A-teams, the main idea of the *distributed constraint optimization problem (DCOP)* approach is that individual agents have limited capabilities so they can only partially contribute to the solution of the global problem. In the DCOP approach, each agent owns a constraint or a variable and the values of the variables are assigned in coordination [40]. Various algorithms have been developed to solve DCOP. An important one is ADOPT, which is a polynomial space algorithm, where agents' execution, either in parallel or asynchronously, guarantees finding the optimal solution [22]. The approach has been successfully applied to a variety of combinatorial optimization problems and several variants and extensions have been developed [19]. The problem attacked in this paper is different than DCOP. Here, our focus is on solving global optimization problems, which cannot be represented as a constraint optimization problem. Further, our approach supports heterogeneity and autonomy of the agents to a larger extent. That is, we enable agents to exchange a richer set of messages and more importantly allow agents to

respond to messages in ways that are most profitable for them. Hence, contrary to agents applied in DCOP approaches, agents in our platform are free to decide whether to answer to queries or follow the directives in the messages as they see fit.

The rest of this paper is organized as follows. Section 2 explains MANGO in depth, detailing important features such as its protocol, library structure and so on. Section 3 describes the anatomy of a MANGO agent and gives guidelines for developing one. Section 4 provides different scenarios that have been developed using MANGO, outlining various interesting results that emerge due to flexible interactions. Finally, Section 5 discusses our work in relation to existing literature and provides directions for future research.

2. MANGO Framework. MANGO is a Java-based multiagent global optimization framework that provides an environment to deploy software agents that cooperatively solve global optimization problems and an API to develop these agents. The essential principle of MANGO is to use cooperation [17] among agents to facilitate the search for a global optimum of a given problem. In order to cooperate effectively agents should communicate with each other using a communication language, which establishes the common understanding over the exchanged messages [8]. In this respect MANGO provides a communication mechanism in which agents can exchange messages to share their information with others to solve global optimization problems. The exchanged messages are specifically designed to share information related to global optimization problems (e.g., a point in the search space and its objective value), which provides a basic language to the agents to support cooperation. An advantage of this language is its extensible nature. MANGO provides fundamental language elements, which can be combined or extended to come up with a richer languages when necessary. Such a language may apply an ontology in order to establish a common understanding to agents on the language elements (i.e., new message types or message contents) [35]. Besides communication, coordination is also required to establish cooperation between agents. There are various coordination mechanisms in the multiagent systems literature such as task sharing and result sharing [10, 31, 41]. In task sharing, a problem is decomposed into smaller sub-problems mostly during design time and each sub-problem is assigned to an agent by a central agent or the designer. Depending on the problem, both the task division and the assignment of tasks to agents with different capabilities can be difficult. MANGO does not offer a static task division capability. For agents are assumed to be autonomous, MANGO does not provide a dedicated method for task assignment, either. However, MANGO provides a yellow page service, which allows agents to advertise their services. Agents can realize task sharing by discovering the other agents that provide suitable services to perform the necessary tasks and requesting these services on demand. As a result of task sharing, a problem is concurrently solved by several agents. Then, the agents share their results with others and refine their own solutions with respect to the other agents' findings. Result sharing is considered beneficial over task sharing since it allows agents to cross-check their results. MANGO supports result sharing by providing special message types, which can be used by the agents to share their results with others. However, to preserve autonomy, MANGO does not enforce agents to share their results or to use the shared results by other agents. In general MANGO does not enforce a specific coordination mechanism for agent cooperation and the agent developers are free to adopt any coordination mechanism that fits best to their specific requirements. On the other hand, MANGO provides basic facilities, such as a directory service, yellow pages service and specific message types to help the developers to implement their coordination

mechanism.

The MANGO environment can be described as follows: autonomous agents interact with each other to find an optimal solution to a given global optimization problem. Agents are built on MANGO API, so the designer is not bothered with technicalities such as entering and leaving system, discovering other agents, sending and receiving low level messages. Agents implement different algorithms, some of which can be found in MANGO libraries, to solve a given problem. Agents exchange messages regarding the global optimization problem. Different message types carrying different information regarding to the solution are implemented specifically for global optimization domain. Also, new message types can be introduced whenever needed. Since the agents are autonomous, they are free to choose the recipient and the content of the message they send. They may also disregard messages depending on their designs. Agents use the received information to find better solutions and send information to help other agents. Once the agents are developed, they are deployed in the MANGO environment. The MANGO environment is a distributed environment that provides registry and yellow pages services to the agents. The environment is distributed so that agents running on different machines interact with each other.

2.1 MANGO Environment. A MANGO environment is a distributed system of agents as we show in Figure 1. An agent is implemented as a Java program that is developed using MANGO API. Agents may run on the same computer or may reside on different computers, which are distributed over the network. Every task, such as running an optimization algorithm, visualization of optimization results and administrative issues, is performed by agents. In general, each agent performs a specific task. However, there is no limitation on the number of tasks that an agent can perform in parallel.

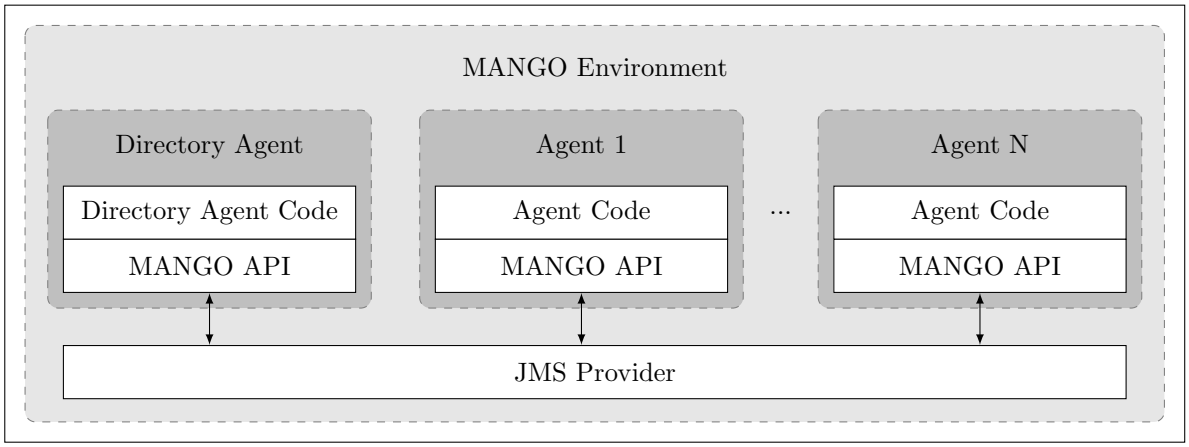


Figure 1: MANGO Environment

The major emphasis of the MANGO framework is cooperation among agents to solve problems. In a MANGO environment, cooperation is realized over the service concept of the service oriented architecture (SOA) [34]. Accordingly, agents provide services to other agents to combine their specific capabilities to solve a common problem. Consider the (parallel) branch-and-bound strategy as we discussed previously. In such a strategy, each search mechanism provided by a search agent is a *service*. The leader agent, who requests these services as it needs, is called the *service consumer* and the search agents, who provide the services (search mechanisms), are called the *service providers*.

Realization of services requires communication. In a MANGO environment, communication is imple-

mented in two levels. The low-level communication, which deals with network structures and protocols, is done over Java Messaging Service (JMS) [14]. We achieve this by integrating a JMS provider implementation into MANGO environment. The JMS provider implementation is mainly responsible from management of network resources that are required for communication. Note that MANGO framework is independent from any specific JMS implementation. Any JMS implementation that is compatible with the JMS specification can be used in a MANGO environment. The high-level communication, which deals with the exchange of agent messages, is build on top of JMS and supports two different messaging paradigms. The first paradigm is based on interrupts. In interrupt based paradigm, when a message is sent to an agent, this agent is interrupted immediately to process the sent message. The second paradigm is based on the use of mailboxes. In this paradigm, the messages that are sent to an agent are stored in a mailbox and the agent is free to check its mailbox whenever it finds it appropriate. Interrupt based messaging favors handling of messages over the execution of agent's tasks. On the other hand, mailbox messaging favors execution of agent's own tasks over the handling of messages. In a MANGO environment agents are free to chose the suitable messaging approach according to their own priorities.

Each MANGO environment includes a special type of agent, which is called the directory agent (DA). DA is mainly responsible from management of communication resources. DA provides two types of services to other agents. The first type of services are related to the management of the JMS communication resources. For instance, when a new agent joins to a MANGO environment, the DA requests the necessary communication resources from the JMS provider on behalf of the new agent. In this way, other types of agents, such as problem solvers, focus on their primary tasks. The second type of service provided by the DA is the directory service that allows agents to discover other agents and their provided services in a MANGO environment. When an agent needs a service, it queries the DA about who can provide the required service and if such a service is provided by an agent, the DA replies the necessary contact information of the provider.

2.2 MANGO API. The MANGO API is a fully extensible API that provides fundamental facilities to the developers to implement their own agents to run in a MANGO environment. The MANGO API is a composition of the five libraries. These are *agent templates*, *communication*, *optimization*, *service* and *log* libraries.

Agent templates library provides a set of agent classes that can be used as a base by developers to develop their own agents. The agent classes in the agent templates library provide fundamental data structures and constructs to developers, by hiding low level details of the JMS based communication and agent administration in the MANGO environment.

The classes of the communication library provides a set of predefined protocols to facilitate messaging between the agents. There are two different family of protocols in the MANGO API. The first family of protocols is the system protocols. These protocols are used for system and administration purposes usually between agents and DA. For instance there is a protocol for service discovery, which specifies how an agent can query the DA to find an agent that provides a certain service. The second family of protocols is the optimization protocols. These protocols are used between agents to cooperatively solve an optimization problem. For instance there is a protocol that can be used by an agent to request the best solution found for a global optimization problem from another agent. However, the agent developers are not restricted only to use the protocols provided by the MANGO API. Hence they can extend provided

protocols or create their own protocols from scratch according to their needs.

The optimization and service libraries provide facilities to represent common concepts of optimization and SOA domains. The optimization library provides classes to define global optimization problems and problem solution formats, which can be used by the agents to achieve a common understanding about the global optimization problems. Similarly, the service library provides facilities to define the services in a common way for all agents. The log library provide basic logging facilities for developers both for debugging agent execution and also for reporting results of optimization processes. The logs are generated in XML format; hence, they are extensible and also processable on various platforms like MANGO itself.

Other than the five libraries of the MANGO API mentioned above, we provide two auxiliary libraries in MANGO Framework. The first of these libraries is the optimization problem library, in which implementations of a set of global optimization problems exist. These global optimization problems are implemented in accordance with the definition of the problem classes given in the MANGO API, hence they are ready to use in a MANGO environment. The second auxiliary library provides a set of optimization algorithms such as trust-region method, quasi-Newton methods, random search, and so on, which can be used by agents in a MANGO environment. [1]

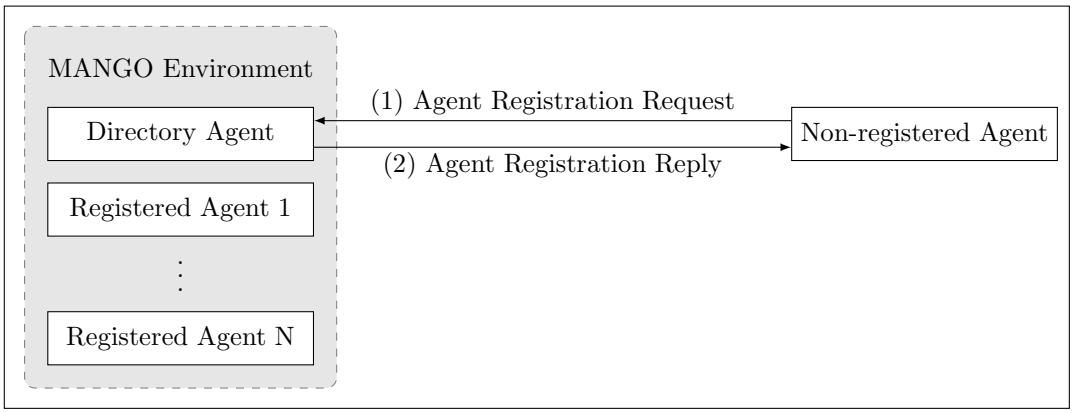


Figure 2: Registration of a new agent to MANGO

2.3 Agent Lifecycle. The lifecycle of an agent starts by registering itself to the MANGO environment via the directory agent as we show in Figure 2. The new agent first sends a request for registration to the directory agent. To register the new agent, the directory agent creates necessary communication facilities and sends a reply to inform the new agent. After the registration process, the new agent is ready to act in the MANGO environment. In general an agent can act in three different ways after this point as we show in Figure 3. First, it can use services provided by other agents in the environment to perform its own tasks. Second, it can provide services to other agents. Third, it can do both in parallel. The services that are of interest here are related to the particular algorithm an agent is employing. When the agent decides to use services of other agents, it queries the directory agent for the available services. According to the results of this query, the agent communicates with those agents that provide the required service. On the other hand, if the agent decides to provide its own service to others, it must register the service to the directory agent in order to inform other agents about its service. An agent can use any number

[1] All of these MANGO libraries as well as documentation for implementation is available at the MANGO Project Web site: https://algopt.sabanciuniv.edu/projects/mango

of services provided by other agents at any point of its lifecycle. Similarly, an agent can provide any number of services and do this at any point of its lifecycle. The lifecycle of an agent ends when the agent deregisters all of its services and also itself from the directory agent.

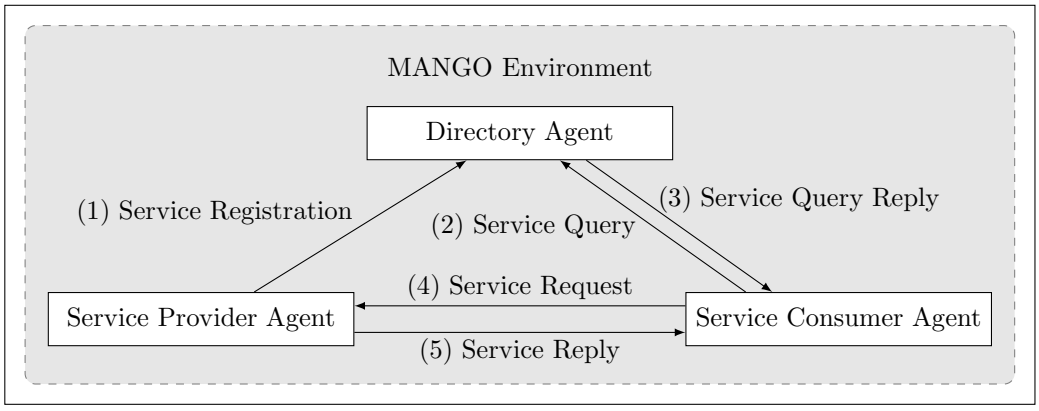


Figure 3: Use of agent services in MANGO

2.4 Protocols and Messages. MANGO environment provides a set of extensible protocols to coordinate communication between agents. Each protocol specifies a set of message types as classes to specify the content to be exchanged by the agents during the execution of the protocol. We divide the protocols into two categories as system and optimization protocols. System protocols are mainly used between individual agents and the MANGO environment. Some examples of system protocols are agent-register-protocol executed when a new agent joins to the environment, service-register-protocol executed when an agent starts to provide a new service, and service-discovery-protocol executed when an agent searches for a new service. On the other hand optimization related protocols provide simple message exchange blocks that can be combined in order to realize complex high-level cooperative optimization strategies explained in Section 4. Some examples of these optimization protocols are solution-request protocol executed when an agent requests the solution of a specific problem from another agent and refrain-request protocol executed when an agent informs another agent not to search a specific region in a given search space, and explore-request protocol executed when an agent requests another agent to explore a certain area.

2.5 Contrast to Existing Multiagent Middleware. There are some other middleware to develop multiagent systems. One of the leading middlewares for multiagent systems is JADE [3]. JADE is an open-source project that enables the building of agents in JAVA. JADE provides a good underlying communication infrastructure, visualization tools, and so forth. The main strength of JADE is its general purpose nature. Independent of the application in hand, one can use JADE to build a multiagent system. However, this strength is also a caveat for specific applications. That is, for solving global optimization problems, we need well-defined data structures to refer to basic constructs, such as regions, solutions, and so forth. Indeed we have also experienced these deficiencies when we tried to improve upon our prior work [18]. MANGO, on the other hand, is specifically targeted for global optimization purposes. It defines necessary message types and data structures that can be used in communication for solving global optimization problems. Other multiagent development middleware such as JACK [39] and Jason [4] are more specialized multiagent modeling and reasoning methodologies, which make them unsuitable to use

to solve global optimization problems.

There are also multiagent simulation frameworks such as MASON [20] and Repast [26]. These frameworks are developed for simulation of various systems using multiagent modeling methodologies and they are widely used for the simulation of social phenomena. Since these tools do not provide general purpose programming facilities, they are also not suitable to solve global optimization problems.

3. Developing a MANGO Agent. Using the above MANGO framework, MANGO agents can be developed. Each MANGO agent in the environment aims to solve a global optimization problem and thus can be considered as a search agent. The development requires first to design the agents based on design principles and then to realize them on the framework.

3.1 Agent Design Principles. When a MANGO agent is being designed, there are three decision points that need to be considered.

Optimization Algorithm: The first point is the agent's main algorithm for attacking the global optimization problem. This algorithm may be any known or newly-developed algorithm for solving a global optimization problem. The agent designer decides on this algorithm and implements it in the agent. The MANGO environment comes with a set of algorithms that can be used when developing new agents.

Outgoing Messages: The second point is related to when and with whom the agent is going to communicate during its execution. The communication is necessary for various reasons, but most importantly for coordination. That is, it is beneficial for an agent to position itself correctly in the environment. For example, two agents may not want to be searching the same area since probably if they search two different areas they may find a solution faster. Conversely, they may want to focus on a certain area rather than diverging.

- ◇ Need-Oriented: The questions of when and with whom to communicate are strictly related to the optimization algorithm that the agent is using. If the agent's own algorithm cannot handle certain tasks, the agent would need others' services to handle these. For example, if the agent's optimization algorithm cannot perform local search well, the agent may find it useful to find other agents that can offer local search service. As explained before, whether an agent does offer this service can be found out by querying the directory agent that keeps track of the services associated with each agent. After finding out the agents that offer the service, the agent may contact one of them to receive the service. Alternatively, an agent that can do local search well may be interested in finding out new areas to search when it finishes its local search. Hence, it may be interested in finding others that can suggest new areas to search.
- ◇ Role-Oriented: An agent may decide to take a leader role in the multiagent system and influence the others by suggesting areas to explore or refrain. The choice of taking this role is up to the agent but also affected by the particular algorithm the agent is executing. That is, some algorithms can identify potential *good* areas quickly; and thus it is reasonable for the agent to take this role and to inform others about the potential of these areas. Conversely, an agent may be designed to play a leader or a follower role during design time as in the second example of Section 1.1. In the setting of this example, the leader agent is intended to manage the overall search whereas the rest of the agents have only local tasks. The algorithm of the leader is able to partition the solution space in a specific way and process the information collected from different parts, i.e., the leader assigns lower bounds to the objective function value attainable in each

partition, eliminates some partitions based on these bounds, and does a new partitioning for the remaining area. The local search agents apply their algorithms only as *requested* by the leader: within the region it tells, using the parameters it decides, and stop according to the termination criteria it asks.

Incoming Messages: The third decision point is related to if and how the agent is going to handle incoming messages. Note that since the communication is asynchronous, the agent will receive incoming messages during the execution of its algorithm. Incoming messages can be handled by the agent in two different ways. The first way is, interrupting the agent immediately when a message is received. In this case, the agent stops execution of its algorithm to handle the incoming message. The second way is, storing incoming messages without interrupting the agent. In this case, the agent checks for incoming messages whenever appropriate.

One naive approach is to always answer or follow the incoming messages. For example, if an agent receives an explore message, it can always jump to the areas that is being suggested for exploration. Or, whenever prompted for the best solution the agent has found, it can return its current best solution. However, the following play an important role in how the incoming messages can be handled intelligently.

- ◇ Exploration State: The exploration state corresponds to how well the agent has explored the environment. This is important in answering questions, since an agent may prefer not to answer a question if it has not explored the environment well or conversely prefer not to follow orders (such as refrain messages) if it has explored the environment carefully. For example, in the beginning of the execution, when the agent did not have enough time to search properly, it may decide not to answer incoming messages related to the best solutions it has found, since its solution may not be accurate.
- ◇ Agent Sending the Message: Over the course of execution, an agent may model other agents based on the types of messages they are sending. Based on this model, an agent may decide how and if it is going to handle a message. For example, after certain iterations an agent may decide to ignore messages from a particular agent, if these messages have become too restrictive for the receiving agent to explore the region.

3.2 Example MANGO Messages. MANGO framework comes with a set of existing messages in its protocol and further enables more messages to be added if necessary. We introduce two example messages that are also used in the following examples.

- ◇ INFORM_SOLUTION: This message takes a solution to the current problem as a parameter. Generally, an agent might want to share its current best solution through this message type. An agent that receives this message might interpret the message in various ways. For example, an agent may consider the received solution as a local minima and avoid it for future explorations.
- ◇ REFRAIN_REGION: This message takes as a parameter a region represented as a ball. When the agent converges to a point x_f, starting from a point x_0, it can send the ball with center x_f and radius $\|x_f - x_0\|$ inside a REFRAIN_REGION message. The intended meaning of this message is that the ball has been already explored by the agent, so others are better off searching elsewhere to increase the chance of finding the global minima. Note that the receiving agents are free to honor or ignore the message. This freedom is what makes multiagent cooperation flexible and different than typical non-autonomous approaches (e.g., meta-heuristics). If an agent decides

to honor the message (i.e., follow it), this is what it does. Instead of spending its effort within such discouraged regions, it would prefer to start a new local search in a possibly unvisited part of the solution space. Hence, it would take the ball and add it to its set of *refrained regions* ($\mathcal{R}$) set or merge with existing elements of this set. Whenever its current iterate x_k falls inside one of the balls in $\mathcal{R}$, i.e,

$$\|x_k - x_c\| \leq r, \quad \text{for some } \mathcal{B}(x_c, r) \in \mathcal{R},$$

it would stop its ongoing run and start a new run from a different initial point.

3.3 Examples of Agent Realizations. To understand how the above principles can be applied, we develop three different agents: Agent B, Agent T and Agent R. For each agent below, we describe the three important points: its algorithm, treatment of outgoing messages and incoming messages

Agent B: This agent applies a BFGS quasi-Newton algorithm with line-search. This local optimization method uses first order derivative information. It progresses by taking steps that provide an *acceptable* decrease in the objective function value. At each iteration, a step is obtained by first computing a direction vector along which the existence of an acceptable step is guaranteed, and then deciding for a step size along that direction. Under certain assumptions, the algorithm is guaranteed to converge to a local solution of (1) starting from an arbitrary initial point. For a detailed explanation of the method see, for example, [25]. To obtain a more practical algorithm, we have also imposed a bound on the maximum number of iterations. Therefore, the algorithm terminates either by recovering a local optimum or exceeding the maximum number of iterations. In fact, Agent B applies a further modification to the BFGS quasi-Newton algorithm as a result of its interaction with other agents, which shall be explained below. Also, since (1) has bound constraints in our case, it projects the steps computed by the original algorithm to the feasible region whenever the next iterate falls outside. We should also note that the agent repeatedly executes its local algorithm, i.e., when it converges to a local solution or exceeds the maximum number of iterations, it starts a new algorithm instance from a different initial point.

Within a system consisting of the above mentioned three types of agents, Agent B has two types of outgoing messages. The first one is a REFRAIN_REGION message sent to all others, i.e., agents of type Agent T and Agent R. The second type of message that this agent can send is an INFORM_SOLUTION message. By design, it only sends this message to Agent T to notify its current best solution. This example depicts that an agent's outgoing message behavior may differ. In different settings, Agent B can send one of these two types, or both.

The only type of incoming message that Agent B is interested in is INFORM_SOLUTION message. When the agent receives this message, it assumes that the received point x_r is a local minimizer. So, not to converge one more time to one of the already discovered local solutions, it adds x_r to a list of *known local minimizers*, and tries to stay away from those points using a penalty function as its objective. The penalty function ϕ is obtained by adding a penalty term to the original objective function f so that approaching to the known local minimizers increases the value of ϕ, i.e.,

$$\phi(x, P) = f(x) + c \sum_{y \in P} \frac{1}{\|x - y\|^2 + \epsilon}, \tag{2}$$

where P is the set of known local minimizers, ϵ is a small positive number, and c is a constant multiplier. In this way, the minimization algorithm applied by Agent B is expected to direct it towards different

local solutions, providing a more extensive search in the overall solution space. If Agent B receives any other type of message, it simply ignores it.

Agent T: Agent T applies a trust-region algorithm, another local search method that also guarantees convergence to a local solution of (1) under some assumptions. Like Agent B, we also impose an upper bound on the maximum number of iterations that could be spent by the algorithm. At each iteration, the trust-region algorithm computes a step via solving a simpler optimization problem that is defined by a model function m and a trust-region parameter Δ. The model function is an approximation to the original objective function f, which is accepted to be a good approximation within a Δ-radius ball, $\mathcal{B}(x_k, \Delta)$, called the *trust-region*. The computed step approximately minimizes m in $\mathcal{B}(x_k, \Delta)$ and is an *acceptable* step if the step also achieves a proportional decrease in f, i.e., m is approved to be a good approximation to f within $\mathcal{B}(x_k, \Delta)$. The model function and the trust region radius are updated at each iteration. For a detailed explanation of the trust-region methods we again refer to [25]. Agent T does not modify the step computation procedure of the original algorithm but may stop a run before converging to a solution by using the information it receives from other agents. If there is no interaction, the agent restarts its algorithm from a random initial point after the termination of each single run, as in the case of Agent B. It also handles the bound constraints of (1) the same way as Agent B does.

In its interaction with Agents B and R, the only type of message Agent T sends is INFORM_SOLUTION message. It sends messages only to Agent B to share its current best solution when it converges to a local minimizer x_f or exceeds a predetermined maximum number of iterations.

Agent T processes two types of incoming messages: REFRAIN_REGION and INFORM_SOLUTION. It assumes that the REFRAIN_REGION messages are valid and follows these messages as explained in Section 3.2. If this agent receives an INFORM_SOLUTION message (i.e., another agent's current best solution), then Agent T acknowledges the message content solution point x_r, and starts its next run from this point if the agent confirms that x_r is not a local minimizer of (1). This case is likely when the sender of the INFORM_SOLUTION message is Agent R.

Agent R: Agent R uses a simple random search as its algorithm. In a single run, the agent evaluates the value of the objective function at a set of points that are uniformly sampled from the feasible region. The agent repeatedly executes this procedure. Its *current solution* is the point with the minimum objective value within all the evaluated sample points up to that moment; consequently, the current solution is updated if a better solution is obtained in the most recent run.

When its current solution is updated, Agent R sends this point to Agent T via an INFORM_SOLUTION message. On the incoming side, Agent R receives and follows REFRAIN_REGION messages. This agent is very quick in exploration and provides diversification; nonetheless, it may be very inefficient in finding a final solution.

Note the flexibility of the interactions among agents. First, the communications need not be symmetric. For example, Agent R sends messages to Agent T, but only receives messages from Agent B. Second, an agent can send certain types of messages to a particular agent but not others. For example, Agent B sends INFORM_SOLUTION to Agent T but REFRAIN_REGION to Agent R. Such variations on communication can be easily adapted in our environment with minimal modifications on the agent and no modifications on the agent's algorithm.

Another important point to note is that while agents are built to cooperate, they can still operate

without cooperation. That is, none of these agents have to receive messages to start working or need to send messages to continue. The cooperation is only an added value to the agents. If the other agents in the system fail for some reason, the remaining agents can still operate. In a similar vein, addition of agents to the system would not need any modifications on the agents' realizations. For example, we could have several Agent Rs in the system and they would all receive messages that are directed to them. This enables agents to enter or leave the system as it suits them.

4. Computational Study. We next present several cooperation scenarios in the proposed environment to illustrate an actual implementation of the sample agents introduced in Section 3.3. Our main purpose in this section is to show the possible effects of communication on individual agents as well as on the overall success of the cooperation for solving a problem. For ease of exposition, we shall start with a base scenario involving a simple cooperation among the agents. Then, we shall alter this scenario by adding other communication channels among the agents and point out some important observations.

We test our scenarios on several global optimization problems from the literature. Among those problems, we have selected three problems with different attributes, such as; dimension, structure and application domain. Table 1 gives the details of our test problems. The first two problems are given by More *et al.* [23]. These problems are reformulations of systems of nonlinear equations as optimization problems. Therefore, the known global optimal objective function values for both problems are zero (see third column of Table 1). These two problems have multiple minima and involve rather flat regions causing performance deterioration for gradient-based methods. The last problem is related to finding the lowest energy configuration of a molecular system. This particular problem is taken from Lennard-Jones clusters with 15 atoms [38]. Here, we note that 3 times the number of atoms gives the problem dimension as shown in the second column of Table 1. To bound the feasible region for sampling, we have assumed the problems are box constrained with the variable bounds given in the last column. Naturally, we make sure that the global optimum for each problem resides within the imposed bounds.

Table 1: The problem details and parameters.

Problem Name	Dimension	Global Optimum	Imposed Variable Bounds
`Rosenbrock`	2	0.0000	[-100, 100]
`Broyden Tridiagonal`	10	0.0000	[-100, 100]
`LJCluster-15`	45	-52.3226	[-5, 5]

In the subsequent part we solve these test problems for different cooperation scenarios. Since we use random sampling, we report for each problem the average statistics over 10 runs. All runs are terminated after a duration (wall clock time) proportional to the problem dimension has elapsed. That is, for each problem, the time to complete a run is taken as the minimum of 5 seconds times the dimension and 5 minutes. Clearly, this setting is to the advantage of the runs taken for individual agents because the entire computing resource is then dedicated to a single agent. We also note that all results are obtained on a dual core personal computer with an Intel Core i5 processor and 4 GB of RAM.

We first start with the results obtained with the individual agents, when they do not cooperate with other agents in the system. These results demonstrate the performances of the agents when they are started from randomly selected points. Since it is very common to use local search methods in such a multi-start setting for solving global optimization problems, these results illustrate what would most of the decision makers do in practice. We shall later use these results for comparing against the results

obtained with the communication scenarios. Table 2 gives the average objective function values obtained by two agents separately. Since Agent R applies a simple random search, these results clearly show that its performance is quite poor.

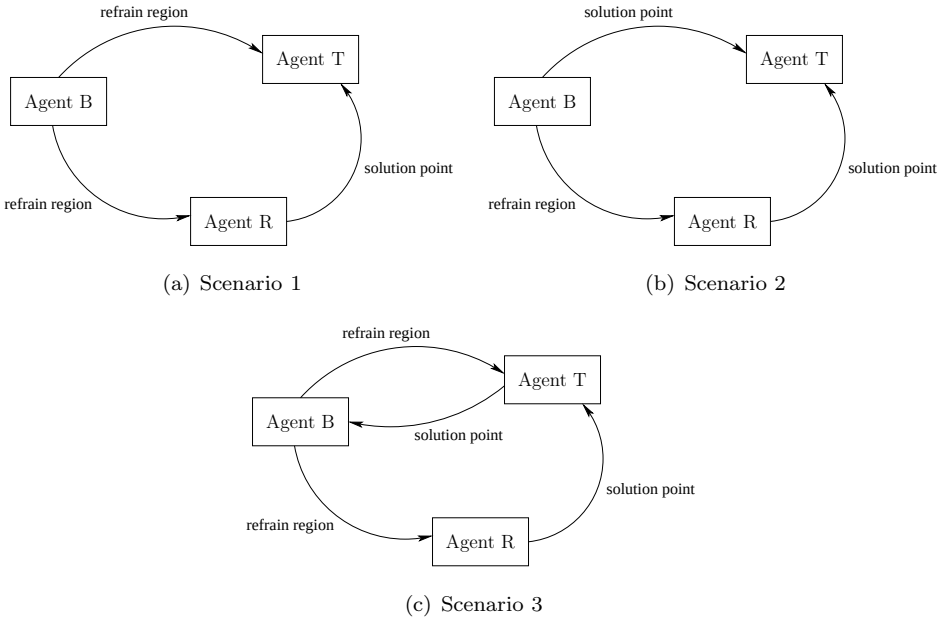
In Table 2 the figures that we shall later use for comparison are marked with boldface letters. As the figures show, both Agent B and Agent T are able to solve the first problem. However, Agent T finds the global optimum solution within a fewer number of function evaluations on average than Agent B does. Therefore, Agent T is used for comparison. When we check the last two problems, we observe that only one agent can find a solution individually. For problem 2, Agent B fails to find the global optimum solution but Agent T does converge to the global optimum. However, the performances of the agents are reversed for the last problem, and Agent B converges to the global optimum whereas Agent T fails. For all problems, Agent R does not converge to the solution, therefore, it has the worst performance.

Table 2: The average objective function values obtained by the individual agents for the test problems over 10 runs.

Problem Name	**Agent B**	**Agent T**	**Agent R**
`Rosenbrock`	0.0000	**0.0000**	0.1779
`Broyden Tridiagonal`	0.0745	**0.0000**	2.15E6
`LJCluster-15`	**-52.2270**	-47.2386	-6.6852

Next we discuss the cooperation scenarios as illustrated in Figure 4. In Table 3, we compare the results obtained with the scenarios against the individual results that are summarized in Table 2. The third column of Table 3 gives the average objective function value obtained by the most successful agent, which is the one reported in the last column. Likewise, the fourth column shows the percentages related to the average number of calls to the objective function and the fifth column presents the percentages related to the time spent until the best objective function value is recovered. These figures are given relative to the number of objective function calls required by the individuals. Thus, the values for the individual runs are omitted and the corresponding cells are marked with (-) signs. For instance, consider the problem `LJCluster-15`. In Scenario 1, Agent B is the most successful agent (see last column) in terms of average objective function value. As shown in the third column, Agent B required on average 82% of the number of function calls used by the individual runs for the same problem. However in the latter two scenarios the successful agents (Agent T for Scenario 2, Agent B for Scenario 3) have required on average slightly more function calls than do the individual runs (4% and 5%, respectively). The fifth column gives a similar percentage comparison relative the individual runs in terms of wall-clock time.

In the base scenario as shown in Figure 4(a), Agent B sends refrain messages to both Agent R and Agent T. The two receiving agents then try to avoid those regions exploited by Agent B. Aside from the refrain messages from Agent B, Agent T also receives some promising solution points to start with from Agent R. In this scenario, we expect Agent T to recover the global optimum quicker than it does when it works individually. As the average numbers of objective function calls in the fourth column of Table 3 show, for the first two problems, Agent T indeed finds the global optimum with less than half of the function calls it uses individually. As we observed with the individual runs, in the last problem Agent B is still the one that converges close to the global optimum. This result is expected because Agent T is refrained from the regions that are exploited by Agent B; hence, the success of Agent B in the vicinity of the global optimum keeps Agent T away from those regions. Although Agent B seems to find this



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

Figure 4: Communication scenarios

Table 3: The average statistics over 10 runs for all communication scenarios.

		Average OF*			Obtained By
		Value	**Calls (%)**	**Time (%)**	
Individuals	`Rosenbrock`	0.0000	-	-	Agent T
	`Broyden Tridiagonal`	0.0000	-	-	Agent T
	`LJCluster-15`	-52.2270	-	-	Agent B
Scenario 1	`Rosenbrock`	0.0000	43%	67%	Agent T
	`Broyden Tridiagonal`	0.0000	32%	57%	Agent T
	`LJCluster-15`	-52.1326	82%	79%	Agent B
Scenario 2	`Rosenbrock`	0.0000	25%	42%	Agent T
	`Broyden Tridiagonal`	0.0000	41%	54%	Agent T
	`LJCluster-15`	-52.3226	104%	84%	Agent T
Scenario 3	`Rosenbrock`	0.0000	23%	38%	Agent T
	`Broyden Tridiagonal`	0.0000	73%	93%	Agent T
	`LJCluster-15`	-52.2321	105%	109%	Agent B

*Objective function

solution faster than it does individually both in terms of wall-clock time and the number of objective function calls (columns 4 and 5), the quality of solution deteriorates slightly. However, notice that Agent B does not receive information from the other agents in this scenario. Thus, this decrease in the solution quality can be attributed to the decrease in the computing power that is allocated to Agent B since it still cooperates with the other agents. Using the same reasoning, we can also infer that Agent B converges very quickly to a value quite close to the optimal objective function value of this particular problem. Then, Agent B makes many function calls to refine this objective function value if the time-limit permits.

Having observed the success of Agent B for the last problem, we next construct Scenario 2, where we try to lead Agent T to the global optimum in *all* problems. As illustrated in Figure 4(b), we accomplish this outcome simply by replacing the refrain message sent from Agent B to Agent T with a solution point message. This change then encourages Agent T to start with those points, which have been recognized as promising but not properly exploited by Agent B particularly when, it terminates its procedure after

exceeding the maximum number of iterations. The last column corresponding Scenario 2 in Table 3 shows that Agent T succeeds to find the global optimum in all problems. Moreover, Agent T finds the exact global optimum in all 10 runs at the expense of a slight increase in the average number of objective function calls but an ample decrease in the wall-clock time. When it comes to the first two problems, as in Scenario 1, we observe a significant decrease in the average numbers of objective function calls as well as in the wall-clock times.

Figure 4(c) shows the last and the most versatile scenario in terms of communication. Unlike the previous two scenarios, Agent B now receives a feedback from Agent T. This feedback pays back for the last problem and Agent B finds a solution closer to the global optimum than the solutions it finds individually and in Scenario 1. However, this improvement comes with a small increase in the average number of objective function calls and the wall-clock time. On the other hand, we obtain the fastest convergence to the global optimum for the first problem with this scenario. Although not as good as the previous two scenarios, the global optimum for the second problem is found within fewer number of function calls than is the individual runs. More remarkably, the wall-clock time to obtain this solution has significantly improved.

One important concern with distributed systems is related to their scalability. This concern can be posed as how much the system performance degrades when the size of the system increases in terms of number of threads and processed jobs. When discussing the scalability of MANGO in such a sense, we need to consider the increase in size with respect two different entities: agents and messages. Since each agent is a stand-alone Java program, the number of agents that can be run on a single machine depends on the individual specification of the machine. Nonetheless, as the MANGO architecture permits running each agent on a different computer, the increase in the number of agents is not expected to cause a significant problem. Therefore, our focus in terms of scalability of physical resources is with the number of exchanged messages. Each message in the system passes through a central messaging system to reach its destination. If agents generate significantly more messages than that can be consumed by other agents, we expect to have delays and performance loss, even when the number of agents is low. This observation is an obvious one, since in our implementation the agents shall then seize their major function of problem solving and devote all their resources to processing messages. This drawback can easily be avoided by limiting the number of messages sent and received by the agents, or processing the incoming messages only when the agent sees fit.

Another important question about scalability is related to the performance improvement in solving a global optimization problem. That is, if we increase the number of agents when solving a particular problem, does this effort necessarily improve the success of the agents for finding the global optimum of a particular problem? Table 4 shows the average statistics over 10 runs for the ninety-dimensional cluster problem `LJCluster-30`. As before, we have assumed that each variable comes from the bounds $[-5, 5]$. The objective function value of the global optimum for this particular problem is -128.2515. The number of function calls are given relative to the numbers obtained with 2 agents (row 3). The results in the table demonstrate that as we increase the number agents up to a certain value we do obtain performance increase. When we reach 10 agents in total, we finally recover the global optimum in the best run. However, as we continue increasing the number of agents, the performance deteriorates. This decrease in the performance can be attributed to two reasons: (i) the agents could be overwhelmed by processing excessive message passing, (ii) too much information creates pollution. The latter reason may

be complemented with further explanation: As each agent tries to direct the others to different parts of the feasible region, it is quite possible that, particularly in earlier iterations, agents receive conflicting messages and hence, fail to explore the feasible region properly. Consequently, we note that one should not indiscriminately assume that the performance shall increase as the number of agents increase. The optimal parameter for the number of agents is clearly problem-dependent and unfortunately, requires some parameter fine-tuning.

Table 4: The average statistics over 10 runs for problem LJCluster-30.

Number of Agents	**OF* Value**		**OF Calls (%)**		**Time (%)**		**Obtained by**
	Best	Average	Best	Average	Best	Average	
1 B, 1 T	-127.4218	-126.6161	-	-	-	-	Agent T
2 B, 2 T	-128.0966	-126.2599	60%	95%	59%	147%	Agent B
5 B, 5 T	**-128.2515**	-126.9845	79%	65%	157%	128%	Agent B
10 B, 10 T	-127.4218	-126.5504	11%	25%	66%	146%	Agent T

*Objective Function

5. Conclusion. The MANGO project is a part of an overall effort targeted at developing optimization algorithms that benefit from the concurrent production of information. Given the challenges in global optimization, we believe that the design of several different successful algorithms of this type can be achieved. Our initial experiments show that cooperation is most useful when agents complement each other in terms of their algorithms. For example, an agent that performs a local search works well with an agent that does not. Hence, whenever there is a need for a local search, the first agent can be consulted. Similarly, the characteristics of the algorithms is influential for a good cooperation scheme. For example, some algorithms execute for a long time in each iteration. Agents that execute such algorithms might have little room for cooperation compared to others that finish iterations more quickly. Overall, it would be interesting to identify cooperation strategies that would be well-suited for various problem types. Our initial results on performance of MANGO are encouraging. However, we plan to study the performance of MANGO on real large-scale global optimization problems and cover some application domains in our future research. Since most real problems have special features, one has to design problem-specific agent cooperation.

Another important issue relevant to the implementation of agent cooperation with MANGO for large-scale problems is scalability. Our discussion in this paper on the scalability of agent cooperation in MANGO can be further extended by extensive numerical testing and benchmarking. There are already successful studies on scalability of parallel optimization frameworks, such as the Genetic Hybrid Algorithm (GHA) approach that is shown to be scalable when applied to a wide spectrum of problems [27, 28].

As an environment for the development of optimization agents, the current MANGO content can be enriched in several ways. We plan to provide more useful services, e.g. procedures for automatic differentiation, and access to open source algorithms written in languages other than Java such as the ones included in the COIN-OR project. Enabling such a feature makes MANGO widely usable. Currently, MANGO does not require an extensive computational environment. Further, since each agent can run on a different machine, the load is automatically distributed. However, with large teams that exchange too many messages, the computation time can still increase. For such settings, the environment can be extended support recent technologies, such as cloud computing.

Providing more features is not the only way to advance MANGO. The cooperation scenarios described in this paper are illustrative in that they exhibit a variety of messages that can be employed in a cooperation scenario. These scenarios can certainly be specialized, for example, by defining new message types to realize a specific cooperation scenario or advanced further by endowing agents with learning capabilities. The users are encouraged to contribute to the existing MANGO libraries by donating their own libraries.

The current version of MANGO is designed only for continuous problems. However, its structure suggests that an extension for combinatorial optimization problems would not be very hard. As in the case of MANGO, the agents of the combinatorial counterpart may not necessarily implement heuristic procedures only, and the cooperation may result in nice hybrids of stochastic and deterministic methods. Successful implementations of this extension would also need to be careful about any special properties of the problems they attack.

Acknowledgments. This research has been partially supported by the Scientific and Technological Research Council of Turkey by a CAREER Award under grant 106M472. Akın Günay is partially supported by TÜBİTAK National PhD Scholarship (2211) and by the Turkish State Planning Organization (DPT) under the TAM Project, number 2007K120610. Some partial details of MANGO have appeared at KES-AMSTA 2009 [13].

References

[1] D. Barbucha, I. Czarnowski, P. Jędrzejowicz, E. Ratajczak-Ropel, and I. Wierzbowska. JABAT middleware as a tool for solving optimization problems. In N. Nguyen and R. Kowalczyk, editors, *Transactions on Computational Collective Intelligence II*, volume 6450 of *Lecture Notes in Computer Science*, pages 181–195. Springer Berlin / Heidelberg, 2010.

[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 the Sixth International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 144–146, 2006.

[3] F. Bellifemine, A. Poggi, and G. Rimassa. Jade: a fipa2000 compliant agent development environment. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 216–217, New York, NY, USA, 2001. ACM.

[4] R.H. Bordini, M. Wooldridge, and J.F. Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007.

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

[6] 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.

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

[8] F. Dignum and M. Greaves, editors. *Issues in Agent Communication*, London, UK, UK, 2000. Springer-Verlag.

[9] V. Dignum. *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, chapter 1. IGI Global, 2009.

[10] E. Durfee. Distributed problem solving and planning. In Michael Luck, Vladimr Mark, Olga tepnkov, and Robert Trappl, editors, *Multi-Agent Systems and Applications*, volume 2086 of *Lecture Notes in Computer Science*, pages 118–149. Springer Berlin / Heidelberg, 2006.

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

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

[13] A. Günay, F. Öztoprak, Ş. İ. Birbil, and P. Yolum. Solving global optimization problems using mango. In A. Hakansson, N.T. Nguyen, R. Hartung, R.J. Howlett, and L.C. Jain, editors, *Agent and Multi-Agent Systems: Technologies and Applications*, volume 5559 of *Lecture Notes in Artificial Intelligence*, page 783792. Springer, 2009.

[14] M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Haase. *Java Message Service API and Tutorial*. Addison-Wesley Professional, April 2002.

[15] M.N. Huhns and M.P. Singh. Agents and multiagent systems: Themes, approaches, and challenges. In *Readings in Agents*, chapter 1, pages 1–23. Morgan Kaufmann, 1998.

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

[17] S. Kraus. Negotiation and cooperation in multi-agent environments. *Artifical Intelligence*, 94(1-2):79–97, 1997.

[18] L. Kerçelli, A. Sezer, F. Öztoprak, Ş.İ. Birbil, and P. Yolum. MANGO: A multiagent environment for global optimization. In *Proceedings of the AAMAS Workshop on Optimization in Multiagent Systems*, pages 86–91, Estoril, Portugal, 2008.

[19] H. C. Lau and H. Wang. A multi-agent approach for solving optimization problems involving expensive resources. In *Proceedings of the ACM Symposium on Applied Computing*, pages 79–83, 2005.

[20] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. Mason: A multiagent simulation environment. *Simulation*, 81:517–527, July 2005.

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

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

[23] J.J. Moré, B.S. Garbow, and K.E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, 1981.

[24] A. Neumaier. Complete search in continuous global optimization and constraint satisfaction. *Acta Numerica*, (13):271–369, 2004.

[25] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, 2006.

[26] M.J. North, T.R. Howe, N.T. Collier, and J.R. Vos. A declarative model assembly infrastructure for verification and validation. In S. Takahashi, D. Sallach, and J. Rouchier, editors, *Advancing Social Simulation: The First World Congress*, pages 129–140. Springer Japan, 2007.

[27] R. Östermark. A flexible platform for mixed-integer non-linear programming problems. *Kybernetes*, 5/6:652–670, 2007.

[28] R. Östermark. Scalability of the genetic hybrid algorithm on a parallel supercomputer. *Kybernetes*, 9/10:1492–1507, 2008.

[29] P.M. Pardalos, H.E. Romeijn, and H. Tuy. Recent developments and trends in global optimization. *Journal of Computational and Applied Mathematics*, 124(1-2):209–228, 2000.

[30] F. Schoen. Stochastic techniques for global optimization: A survey of recent advances. *Journal of Global Optimization*, 1(3):207–228, 1991.

[31] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, 1998.

[32] 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.

[33] D.S. Siirola, S. Hauan, and A.W. Westerberg. Computing pareto fronts using distributed agents. *Computers and Chemical Engineering*, 29:113–126, 2004.

[34] M. P. Singh and M. N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley & Sons, Chichester, UK, 2005.

[35] S. Staab, R. Studer, H. Schnurr, and Y. Sure. Knowledge processes and ontologies. *IEEE Intelligent Systems*, 16:26–34, 2001.

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

[37] 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.

[38] D.J. Wales and J.P.K. Doye. Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A*, 101(28):5111–5116, 1997.

[39] M. Winikoff. Jack intelligent agents: An industrial strength platform. In G. Weiss, R. Bordini, M. Dastani, J. Dix, and A.F. Seghrouchni, editors, *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, And Simulated Organizations*, pages 175–193. Springer US, 2005.

[40] 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.

[41] G. Zlotkin and J. S. Rosenschein. Negotiation and task sharing among autonomous agents in cooperative domains. In *Proceedings of the 11th international joint conference on Artificial intelligence - Volume 2*, IJCAI'89, pages 912–917, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.