Solving Global Optimization Problems using MANGO*

Akın Günay[1], Figen Öztoprak[2], Ş. İlker Birbil[2], and Pınar Yolum[1]

[1] Boğaziçi University, İstanbul, Turkey
akin.gunay@boun.edu.tr, pinar.yolum@boun.edu.tr
[2] Sabancı University, İstanbul, Turkey,
figen@su.sabanciuniv.edu, sibirbil@sabanciuniv.edu

Abstract. Traditional approaches for solving global optimization problems generally rely on a single algorithm. The algorithm may be hybrid or applied in parallel. Contrary to traditional approaches, this paper proposes to form teams of algorithms to tackle global optimization problems. Each algorithm is embodied and ran by a software agent. Agents exist in a multiagent system and communicate over our proposed MultiAgent ENvironment for Global Optimization (MANGO). Through communication and cooperation, the agents complement each other in tasks that they cannot do on their own. This paper gives a formal description of MANGO and outlines design principles for developing agents to execute on MANGO. Our case study shows the effectiveness of multiagent teams in solving global optimization problems.

1 Introduction

Many powerful algorithms for solving global optimization problems exist [**?**]. Some of the algorithms propose a single, unique technique to solve a problem, while others propose manually-integrated, hybrid approaches. In both cases, algorithms can be run independently or in parallel. Many such approaches have their own advantages over others. Depending on context, one algorithm can defeat others in finding a solution. One intuitive perspective is to enable these algorithms to exist in teams so that they can complement each other in tasks they cannot do well on their own. Depending on the context, one algorithm can help the other algorithm (e.g., if one gets stuck in a local optimum) to continue operations successfully [3–5].

(Software) Agents are autonomous computations that can perceive their environment, reason, and act accordingly [1]. Agents are most useful when they exist in a multiagent system so that they can interoperate with other agents. Interoperation requires agents to speak a common language to coordinate their activities and to cooperate if they see fit. Autonomy of the agents implies that agent can choose how and with whom they want to interact. These properties of multiagent system make it an ideal candidate to realize teams of algorithms. While multiagent systems have been used in many areas, their use in solving global optimization problems, as we discuss here, is new.

* This research has been partially supported by the Scientific and Technological Research Council of Turkey by a CAREER Award under grant 106M472. A preliminary version of this paper appeared at AAMAS OPTMAS Workshop 2008.

Our proposed software environment is MANGO, which provides the necessary utilities to develop agents that can participate in a multiagent system to solve global optimization problems. MANGO enables agents to find each other through a directory system. MANGO contains an extendible protocol for agents to communicate with each other. The protocol messages are related to solving problems, such as exchanging current best points, signaling areas already explored by others, and so on. Hence, agents can find others and cooperate with them on their own. As a result, agent teams are formed and exploited to solve global optimization problems.

There are three main contributions of this paper: (i) We summarize MANGO, our proposed environment for developing multiagent systems that solve global optimization problems cooperatively. (ii) We provide design principles for developing an agent that can participate in a multiagent environment for solving global optimization problems. (iii) We develop a case study in which agents are developed following the above design principles and show how they can actually solve a given global optimization problem.

The rest of this paper is organized as follows: Section 2 explains MANGO in detail, concentrating on the architecture, services, and messaging. Section 3 identifies the design principles for developing a MANGO agent. Section 4 develops a case study to show how MANGO agents can cooperate to solve global optimization problems. Finally, Section 5 gives our conclusions and future research ideas.

2 MANGO Framework

MANGO is a multiagent global optimization framework. It is implemented in Java and uses Java Messaging Service (JMS) technology. The aim of MANGO is to provide a development and experiment environment for global optimization research. Because of this, contrary to conventional multiagent development environments, the agents that can be developed using MANGO are targeted for solving global optimization problems. Concepts that are crucial for global optimization such as problem definitions, problem solutions, and so on are first class entities in MANGO. In the rest of this section we explain the MANGO environment and working principles of its components in detail.

2.1 Architecture Overview

We present the architecture of the MANGO environment in Figure 1. The fundamental entity of the MANGO architecture is the agent. Every task, such as solving global optimization problems, visualization of results and administrative tasks in the MANGO environment is performed by agents.

An agent is implemented as a regular Java executable that uses MANGO API in order to work in the MANGO environment. Agents communicate through JMS, however MANGO API hides details of the JMS communication by providing its own communication methods in order to simplify development process. MANGO uses the service concept of the service oriented architecture (SOA) [1]. In this manner agents may provide services to other agents. For instance, in a typical MANGO environment one agent may provide a service to solve global optimization problems using a specific algorithm whereas another agent might provide a visualization service to graphically represent

the results of the optimization algorithm. A third agent might use these two services in combination to solve its global optimization problem and to visualize the results. MANGO environment itself provides a directory agent for management and service discovery purposes.

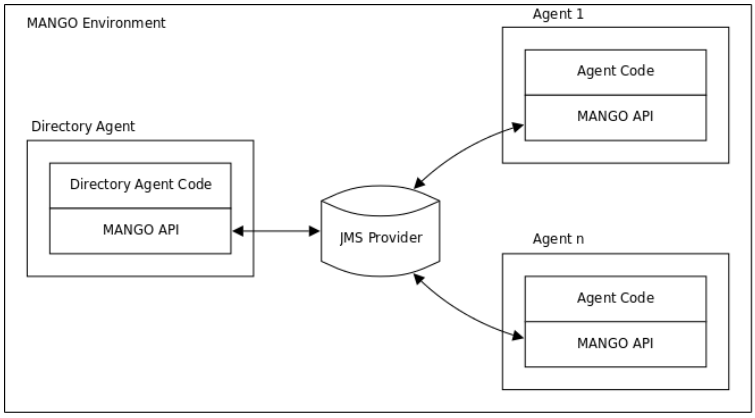


Fig. 1. MANGO Architecture

2.2 Directory Service

In MANGO environment a directory agent is a special agent with administrative and managerial capabilities and each MANGO environment has one directory agent. It keeps track of all other agents and their services. Using this information it also acts as a service matchmaker and provides service discovery service to other agents in the environment. It is also responsible for low level managerial tasks such as maintenance of communication resources.

2.3 MANGO API

The MANGO API is a fully extendible API that provides all necessary facilities for the developers to implement their own agents for the MANGO environment. Figure 2 shows the basic components of the MANGO API. There are four main libraries of classes as the *agent templates, protocol, optimization and service*. Agent templates library provides a set of basic agents with communication capabilities. Developers can implement their own agents through extending these agent templates without considering details such as messaging. Classes in the protocol library are the predefined set of protocols and related messages for agent communication. These protocols are further divided into two libraries as system and optimization protocol libraries. System protocols are mainly used by the agents to communicate with the directory agent for resource management and service discovery purposes. On the other hand optimization protocols

are used between the agents to solve these problems in cooperation. Classes under the optimization library provide facilities for global optimization. These include common problem and solution definitions that allow interchange of global optimization problem knowledge between agents and utility classes used in these solution and problem definitions. The classes in service library provide support to define agent services, both for administration and optimization.

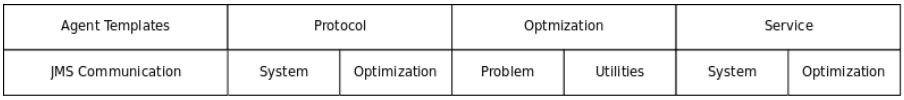


Fig. 2. MANGO API

2.4 Protocols and Messages

MANGO environment provides a set of extendable 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 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 Agent Lifecycle

The lifecycle of an agent starts by registering itself to the directory agent. While registering the new agent to the MANGO environment, the directory agent creates necessary communication facilities for 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. 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. When the agent decides to use services from other agents, it queries the directory agent for the available services. According to the results of this query it communicates with the agents that provide the required service. On the other

hand, if the agent decides to provide its own service to other agent, it must register the service to the directory agent in order to inform other agents about its service. An agent can use any number 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 unregisters all of its services and also itself from the directory agent.

3 Developing a MANGO Agent

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.

Outgoing Messages: The second component 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. That is, generally 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.

- Needed Services: 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.
- Played Roles: An agent may decide to take a leader role in the multiagent system and influence the others by suggesting areas to explore or refrain from. The choice of taking this role is up to the agent, but is 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.

Incoming Messages: The third decision point is related to if and how the agent is going to handle incoming messages. 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 it is prompted for the best solution it 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 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 representative.
- Agent Sending the Message: Over the course of exaction, 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, if an agent sends frequent explore messages to a second agent, the receiving agent may mark the sender agent as a "spammer" and decide to ignore messages coming from that agent.

4 A Case Study

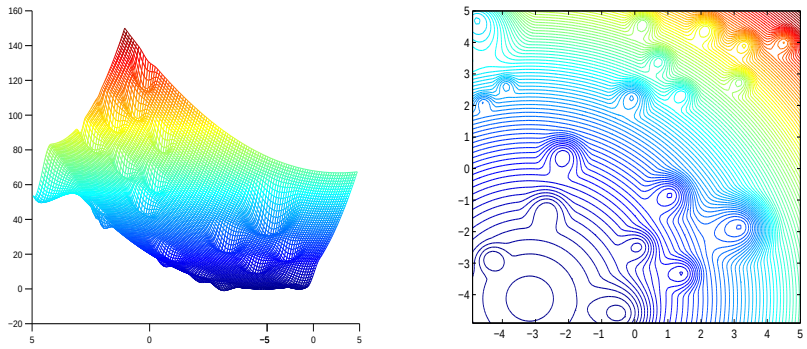
In this section, we provide an illustrative system that has been established to solve a global optimization problem with three cooperative agents in MANGO.

Motivations for the example: The strategies in this example are motivated by two challenges in global optimization. An important issue that makes very well-known efficient local optimization methods useless for global optimization problems is that the objective function may have multiple local minima. A local optimization method finds one of them, which may or may not be the global one, depending on the initial point from where it has started its search. A first idea to overcome this problem to a certain degree is starting local search from several initial points. But the obvious drawback of this straightforward approach is that many searches may end up in the same local minimum point, i.e., the same local minimum may be rediscovered for several times.

Position of the global minimum is another issue. When the global minimum lies at the bottom of a large basin, i.e., the attraction region, it is relatively easier to find it out since an initial point is near to that large attraction region with a higher probability. A situation at least as hard as a narrow attraction region is a narrow attraction region placed within the attraction region of another local minimum. In this case, there is a significant risk of ending up at the more attractive local minimum point even when we start very close to the global minimum. If we escape from the larger attraction region not to rediscover it, then we may never approach to the global minimum and waste our efforts in irrelevant faraway parts of the search space.

The problem: We select a two-dimensional problem for our illustration. The problem is produced by the GKLS generator [2]. It has 20 local minima and finding the global minimum is quite hard in the sense that it has a relatively small attraction region and it is located within the attraction region of another local minimum (see Figure 3).

The agents: The three agents we run in this example are all local optimization agents [**?**]. We name the three agents as *BFGS*, *TR*, and *PTR*. The BFGSAgent applies BFGS



(a) Mesh plot of the objective function (b) Contours of the objective function

Fig. 3. The problem

quasi-Newton algorithm: a line search method which progresses by taking steps through directions that provide decrease in the objective function value, so it calculates a direction vector and a step size at each iteration. TRAgent applies a trust region method. It generates a model function which is a quadratic approximation to the original objective function. It accepts that the model function is a good approximation within a *trust region*, a Δ-radius ball, and it minimizes the model function in that region. It updates the model function and the trust region radius at each iteration. Finally, PTRAgent applies a perturbed trust region method: It applies a trust region algorithm like TRAgent. But it works more sensitive, i.e., the maximum radius value allowed is small. Also, it follows a perturbed direction in some iterations to increase the chance of finding an unvisited minimum point. That is, the iterates are not moved along the direction as in the regular trust region method; instead, the trust region direction is distorted randomly.
Cooperation strategies: The cooperation strategies applied by three agents are illustrated in Figure 4. They follow three basic ideas to improve their performance:

- penalize approaching to already discovered local minima (BFGS)
- do not enter regions searched by others (TR)
- conduct a more sensitive search in the region searched by the other agents, so that a possible global optimum near a local one is not missed (PTR)

In this context, the cooperation procedures applied by each agent are summarized as follows:

- BFGSAgent

  ListenMessage: If an INFORM_SOLUTION message is received, then add the received point x_r to the *penalized* list so that approaching to x_r increases the objective function.

  SendMessage: If converged to a point x_f, starting from a point x_0, then send the ball with center x_f and radius $\|x_f - x_0\|$ to both TRAgent and PTRAgent as a REFRAIN_REGION message.

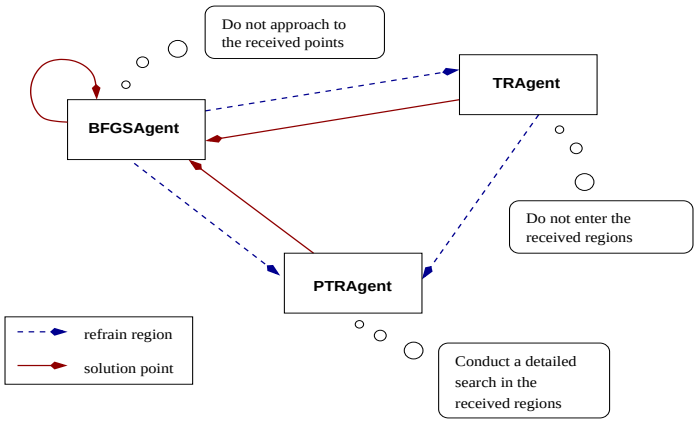


Fig. 4. The cooperation strategies

– TRAgent

ListenMessage: If a REFRAIN_REGION message is received, then add the received region to the *refrained* list so that if case of entrance to that region leave the ongoing search and start a new search from some random point in the solution space.

SendMessage: If converged to a point x_f, then send it to BFGSAgent as an INFORM_SOLUTION message. If the radius of the last trust region is Δ_f, then send the ball with center x_f and radius $k\Delta_f$ to PTRAgent as a REFRAIN_REGION message, $k \geq 1$.

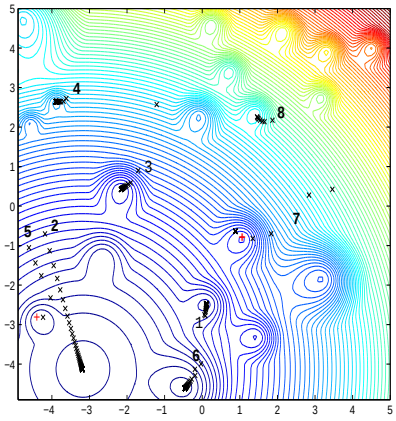


Fig. 5. The penalizing strategy applied by BFGSAgent

– PTRAgent

ListenMessage If a REFRAIN_REGION message is received, then it is added to the *explore* list so that it is going to be searched for a minimum other than the center of that region.

SendMessage If converged to a point x_f, then send it to BFGSAgent as an INFORM_SOLUTION message.

Observations: We next illustrate how those strategies work on the above mentioned problem with 20 minima. In Figure 5, the search paths of BFGSAgent has been marked with $\times$. The points marked by $+$ are the minima that have been already discovered by other agents and sent to BFGSAgent. The consecutive searches conducted by BFGSAgent are numbered. As the figure points out, the search is discouraged to approach to the previously converged points, either by BFGSAgent or by the other agents.

In Figure 6, we illustrate the second idea. The search paths of TRAgent are marked with $+$ signs, the circles are the trust regions. The paths are numbered at their starting points. During its second run, TRAgent has entered to the refrain region sent by the BFGSAgent at its forth step. Thus, it has left the second path at that point and started a new search from another point so that it has spent its effort for discovering another local minimum (at the end of the third path).

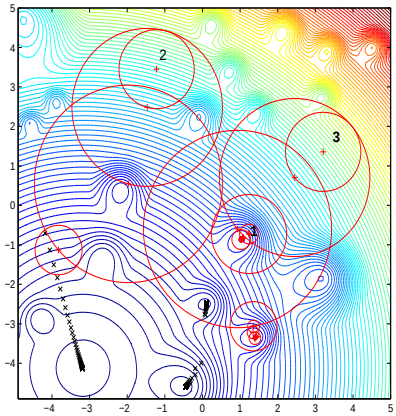


Fig. 6. The leaving strategy applied by TRAgent

Finally, in Figure7, we can see the steps of PTRAgent marked by $*$ signs. It has started a new search in the refrain region sent by BFGSAgent, which has provided finding the global minimum point.

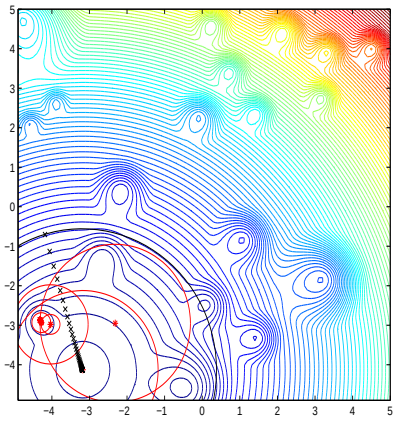


Fig. 7. The research strategy applied by PTRAgent

5 Conclusions

We have introduced a new multi-agent environment for global optimization. The proposed environment provides a flexible mechanism that can be used to design new cooperation strategies among different global optimization algorithms. We have demonstrated on an illustrative example that the design of different cooperation strategies can significantly enhance the performance of individual algorithms. In the future, we intend to focus on different strategies and demonstrate their performances with empirical results.

References

1. Singh, M.P., Huhns, M.N.: Service-Oriented Computing: Semantics, Processes, Agents. John Wiley & Sons, Chichester, UK (2005)
2. M. Gaviano, D. E. Kvasov, D.L., Sergeyev, Y.D.: Algorithm 829: Software for generation of classes of test functions with known local and global minima for global optimization. ACM Transactions on Mathematical Software **29**(4) (2003) 469–480
3. Talukdar, S., Baerentzen, L., Gove, A., Souza, P.D.: Asynchronous teams: Cooperation schemes for autonomous agents. Journal of Heuristics **4**(4) (1998) 295–321
4. Tyner, K., Westerberg, A.: Multiperiod design of azetropic seperation systems i: An agent based approach. Computers and Chemical Engineering **25** (2001) 1267–1284
5. Siirola, D., S.Hauan, Westerberg, A.: Toward agent-based process systems engineering: Proposed framework and application to non-convex optimization. Computers and Chemical Engineering **27** (2003) 1801–1811
6. Yokoo, M. ve Ishida, T.: Search algorithms for agents. In G.Weiss, ed.: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. The MIT Press (1990)
7. Modi, P., Shena, W., Tambe, M., Yokoo, M.: Adopt: asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence **161** (2005) 149–180

8. Lau, H.C., Wang, H.: A multi-agent approach for solving optimization problems involving expensive resources. In: ACM Symposium on Applied Computing. (2005)
9. Ahluwalia, A., Modiano, E.: On the complexity and distributed construction of energy-efficient broadcast trees in wireless ad-hoc networks. IEEE Transactions on Wireless Communications **4**(5) (2005) 2136–2147
10. Rabbat, M., Nowak, R.: Distributed optimization in sensor networks. In: IPSN'04. (2004)
11. Colorni, A., Dorigo, M., Maniezzo, V.: Distributed optimization by ant colonies. In: Proceedings of European Conference on Artificial Life. (1991) 134–142
12. J. Kennedy, R.: Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks. Volume 4. (1995) 1942–1948
13. Birbil, S., Fang, S.C.: An electromagnetism-like mechanism for global optimization. Journal of Global Optimization **25**(3) (2003) 263–282
14. Tsui, K., Liu, J.: Evolutionary diffusion optimization, part i:description of the algorithm. In: Proceedings of Congr. Evolutionary Computation (CEC). (2002) 1284–1290