Applied Soft Computing 14 (2014) 577–593

Contents lists available at ScienceDirect

Applied Soft Computing

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

The role of cardinality and neighborhood sampling strategy in agent-based cooperative strategies for Dynamic Optimization Problems

Antonio D. Masegosa *, David Pelta, Ignacio G. del Amo

Models of Decision and Optimisation Research Group, Center for Research on ICT, University of Granada, 18071 Granada, Spain

ARTICLE INFO

Article history:
Received 8 May 2012
Received in revised form 23 June 2013
Accepted 7 August 2013
Available online 17 September 2013

Keywords:
Dynamic Optimization Problems
Agent-based optimization
Hybrid metaheuristics
Cooperative strategies

ABSTRACT

The best performing methods for Dynamic Optimization Problems (DOPs) are usually based on a set of *agents* that can have different complexity (like *solutions* in Evolutionary Algorithms, *particles* in Particle Swarm Optimization, or *metaheuristics* in hybrid cooperative strategies). While methods based on low-complexity agents are widely applied in DOPs, the use of more "intelligent" agents has rarely been explored. This work focuses on this topic and more specifically on the use of cooperative strategies composed by trajectory-based search agents for DOPs. Within this context, we analyze the influence of the number of agents (cardinality) and their neighborhood sampling strategy on the performance of these methods. Using a low number of agents with distinct neighborhood sampling strategies shows the best results. This method is then compared versus state-of-the-art algorithms using as test bed the well-known Moving Peaks Benchmark and dynamic versions of the Ackley's, Griewank's and Rastrigin's functions. The results show that this configuration of the cooperative strategy is competitive with respect to the state-of-the-art methods.

1. Introduction

Dynamic Optimization Problems (DOPs) have attracted the attention of the scientific community in the past decade due to its closeness to real-world situations (trade market prediction, meteorological forecast, robotics motion control, etc. [1–3]). Solving this type of problems is harder than their "static" counterparts because of the presence of time-dependent properties that may affect the fitness function, constraints, number of variables, their domain, etc.

It is not our purpose here to provide a literature review on the topic (the interested reader is referred to [4] and the website http://dynamic-optimization.org for more information) but nowadays, most of the publications in this field consider some assumptions for the resolution of DOPs:

- Changes in the problem can be detected.
- Time-dependent features vary gradually.
- Keeping a good track of the optima is more important than refining the quality of the solutions, especially when the frequency of the changes is high.
- A population of solutions may better track the changes than a unique solution.

Under these assumptions, it is not strange that most of the research is concentrated around population-based methods [2,5] where each member of the population can be considered as an "agent" in a wide sense. For example, an agent could be a very simple entity as a solution (an individual in the case of Evolutionary Algorithms [6,7] or a particle in PSO [8,9]), or a more complex object like a local search operator (mutation operator [10] or a full Tabu Search method [11]).

While methods composed by low-complexity agents are widely extended in DOPs, the use of more sophisticated agents has been less explored. The present work focuses on this last topic and concretely on cooperative strategies where the agents implement trajectory-based algorithms.

Having in mind the need for more competitive methods for solving DOPs, this paper pursues two main objectives. The first one is to analyze two of the features with a higher influence on the performance of cooperative strategies:

- The number of agents composing the strategy, the so called *cardinality*.
- The Neighborhood Sampling Strategy (*NSS*) implemented by the agents.

* Corresponding author. Tel.: +34 673110419.
E-mail addresses: admase@decsai.ugr.es (A.D. Masegosa), dpelta@decsai.ugr.es (D. Pelta).

1568-4946/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2013.08.006

To carry out this study, we depart from the approach proposed in [12] where the authors presented a cooperative strategy composed by a set of Tabu Search agents that are controlled by a rule-based central coordinator. In this contribution, the strategy uses the same type of agent.

To evaluate the influence of the cardinality (first feature), we test various configurations with different number of agents. For the second feature, we consider that all the agents can use the same or different NSS's, leading to a homogeneous or a heterogeneous *composition*, respectively. The analysis is done evaluating different pairs {cardinality, composition} over the well-known Moving Peaks Benchmark [1] and dynamic versions of the Ackley's, Griewank's and Rastrigin's continuous optimization functions.

The second objective of the paper is to assess the performance of the best pair {cardinality, composition} by comparing it versus state-of-the-art algorithms for continuous DOPs.

It is important to remark that we will only consider DOPs having changes in the objective function.

This work extends the research presented in [11–13], where the same cooperative strategy was used. In the first one, the aim was to study the performance of trajectory-based agents in DOPs as well as the benefits of cooperation; the second work focused on the analysis of different coordination schemas with distinct control rules for this cooperative strategy; and the last paper provided a comparison of several state-of-the art algorithms for DOPs that included this cooperative strategy, emphasizing the comparison methodology and the visualization of the results.

The paper is structured as follows. Section 2 describes the cooperative strategy, the Tabu Search implemented by the agents and the NSS's proposed. The experimental framework is in Section 3, where the benchmark problems, the state-of-the-art methods, the performance measures and the analysis techniques used are presented. Section 4 contains the results related with the first aim of the paper while Section 5 shows the comparison of the best cooperative strategy obtained versus the state-of-the-art algorithms. Finally, conclusions are given in Section 6.

2. A centralized cooperative strategy for Dynamic Optimization Problems

As we stated before, the cooperative strategy used in this paper was previously presented in [11,12]. In this section, we describe the main aspects of this strategy, the Tabu Search method implemented by the agents and the three Neighborhood Sampling Strategies considered. Further details of the strategy and the Tabu Search method can be found in previous references.

2.1. Description of the cooperative strategy

The strategy consists of a set of agents guided by a central coordinator [12] as shown in Fig. 1. The agents explore the search space using a Tabu Search algorithm, periodically sending performance reports to a coordinator which analyses them and readjusts the behavior of the agents by sending orders to them. The orders indicate the agents the point from which they should restart their search, either to move them into more promising region or to escape from a local minimum. The exchange of data is done using a blackboard model [14]. Concretely, two blackboards are available, one to send performance reports from the agents to the coordinator, and another, to send orders from the coordinator to the agents.

Within the cooperative strategy, the coordinator and the agents maintain certain information about the search (e.g. fitness of the current solution, best solution found so far, etc.). When a change is detected by any of the agents, the rest of them should be informed in order to update or discard their search information. The mechanism

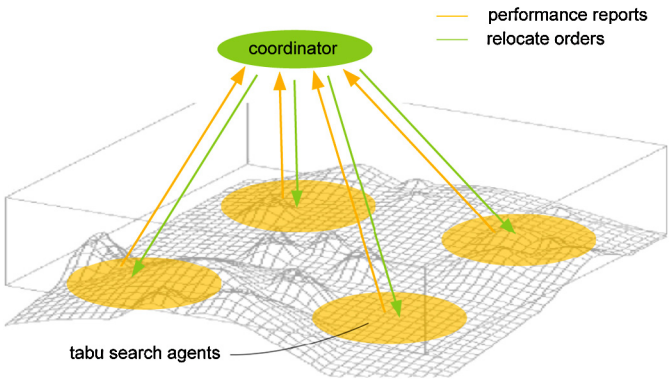

Fig. 1. Scheme of the cooperative strategy.

to detect changes and the system to communicate these changes are described below.

An agent detects fitness function changes by reevaluating its best solution found so far and checking whether its fitness has changed or not. This action is done after each neighborhood exploration to ensure early detection of changes. When an agent i detects a change, it increases a counter cc_i (change counter), that stores the number of changes detected so far (see Fig. 2). Then it will send the cc_i value to the coordinator in the next report. The coordinator also maintains a register, called cc_{global}, with the highest cc_i received so far. After receiving a report, the coordinator determines whether the agent has detected a new change ($cc_i > cc_{global}$) to update cc_{global}; or if it has already perceived the current change ($cc_i = cc_{global}$) or not ($cc_i < cc_{global}$). In the former case, the coordinator notifies those agents that have missed the change to make all the strategy's components work with current information, as Fig. 2 illustrates.

Now, we can describe the cooperative strategy with the help of Fig. 3. For the sake of clarity, we divide the description in agents and coordinator:

2.1.1. Agents

At the beginning, the agents are initialized and run asynchronously. Each agent first checks if it has received an order or report from the coordinator. This report contains:

- Agent identification
- s_{reloc}
- *changeNotification*

where s_{reloc} can be either a solution from which the agent will restart the search, or an empty value that will be ignored. Regarding *changeNotification*, it is a boolean value set to *True* to inform the agent that a new change was detected.

When the received report has *changeNotification* = *True*, the agent updates its cc_i counter and restarts memories having "obsolete" information as: components of the optimizer (tabu list), the fitness of the current solution, the best solution found so far (s_{best}) and a list where it keeps the local minima found since the last report. These local minima are solutions that could not be improved by the agent after exploring the corresponding neighborhoods a certain number of times. Before running the optimizer, the agent checks if it has to relocate its search ($s_{reloc} \neq \emptyset$) and in this case, it sets its current solution to s_{reloc}. After a given number of fitness function evaluations, the agent stops the search process, updates the memories, records the performance information in a report and sends it to the coordinator. These performance reports include the following information: