Chapter 3

Juan R. González, Carlos Cruz, Ignacio G. del Amo, and David A. Pelta

Abstract. This work presents the results obtained when using a decentralised multi-agent strategy (Agents) to solve dynamic optimization problems of a combinatorial nature. To improve the results of the strategy, we also include a simple adaptive scheme for several configuration variants of a mutation operator in order to obtain a more robust behaviour. The adaptive scheme is also tested on an evolutionary algorithm (EA). Finally, both Agents and EA are compared against the recent state of the art adaptive hill-climbing memetic algorithm (AHMA).

3.1 Introduction

There is a very active research on the field of Dynamic Optimization Problems (DOPs) [2, 3, 5, 14]. DOPs are problems where the fitness landscape changes with time and its main interest relies on its closeness to real world where the problems are rarely static (trade-market prediction, weather forecast, robot motion control, . . . ) . Since the problem changes generally occur on a gradual manner, the algorithms for DOPs try to reuse the information obtained in previous stages to better solve the problem at its current state and to try and track the movement of the optima. This approach is generally better and faster than re-optimizing.

Most of the research for DOPs has been focused on evolutionary algorithms (EAs), but we were able to improve the results of EAs on several usual test problems, such as the moving peaks [3], by using an algorithm with multiple agents cooperating on a decentralised manner to improve a set of solutions on a matrix [7, 9, 10]. This algorithm, called simply Agents, has only been tested on continuous DOPs.
Therefore, the first goal of this work is to present a new version of Agents for combinatorial DOPs to see if it can obtain results as good as in the continuous case. Additionally, the performance of an heuristic depends on the parameters used and no specific parameters can work well across different problems and instances [11]. Moreover, the optimal parameters may vary as the search process is conducted. The difficulties increase with DOPs, since the problem is going to change and these changes may affect the validity of the learning done. Therefore, a second goal of this paper is to test a simple adaptive scheme for learning among several mutation operator variants and to see if it can improve the results even on this difficult scenario of combinatorial DOPs. To further test this adaptive scheme it will also be applied to a standard evolutionary algorithm (EA). An additional third goal of this paper will be to compare both Agents and EA algorithms with the adaptive scheme against the state of the art hill-climbing memetic algorithm (AHMA) [12].

To achieve the previous goals, this paper is structured as follows. Firstly, Section 3.2 presents the algorithms used through the paper: the new version of the Agents algorithm for combinatorial DOPs, the EA implemented to further test the adaptive scheme, and the state of the art AHMA used for comparison. Secondly, in Section 3.3 we describe the adaptive scheme and how it is incorporated to both Agents and the EA. Then, Section 3.4 describes the combinatorial DOPs that will be used to test the algorithms. After that, Section 3.5 describes the experiments done and their results. Finally, the paper concludes at Section 3.6 with the conclusions and future work.

3.2 Algorithms

3.2.1 Agents

The multiagents algorithm presented here is a decentralised cooperative strategy that has been previously applied to continuous dynamic optimization problems [7,9,10]. The strategy makes use of a matrix of solutions and a group of agents that move through the matrix trying to improve the solutions stored on the matrix cells they visit. The cooperation is based on the fact that the solutions that an agent improves may be later improved by other agents that arrive at the same cells on the matrix. Algorithm 3.1 presents the pseudocode of Agents. Basically, the algorithm is run until the stop condition is met, which will normally be when the resources are exhausted (such as the time or the number of evaluations/changes of the objective function). For each iteration, the function detectChanges() is called to reevaluates all the solutions if a change on the problem is detected. The change detection is done by recomputing the fitness value of the best solution and comparing it with the previous one to see if it has changed. Then, for each iteration of the inner loop, an agent is selected, in a circular fashion, and it is moved to the best neighbor solution (in terms of horizontal and vertical adjacency on the matrix) and it tries to improve this solution. The algorithm performs as many iterations of the inner loop as solutions on the matrix. In this way, the number of iterations of the for loop is similar to a generation of the other population-based algorithms considered on the paper.