Symbolic Archive Representation for a Fast Nondominance Test*

Martin Lukasiewycz, Michael Glaß, Christian Haubelt, and Jürgen Teich

Hardware-Software-Co-Design
Department of Computer Science 12
University of Erlangen-Nuremberg, Germany
{martin.lukasiewycz, glass, haubelt, teich}@cs.fau.de

Abstract. Archives are used in Multi-Objective Evolutionary Algorithms to establish elitism. Depending on the optimization problem, an unconstrained archive may grow to an immense size. With the growing number of nondominated solutions in the archive, testing a new solution for nondominance against this archive becomes the main bottleneck during optimization. As a remedy to this problem, we will propose a new data structure on the basis of Binary Decision Diagrams (BDDs) that permits a nondominance test with a runtime that is independent from the archive size. For this purpose, the region in the objective space weakly dominated by the solutions in the archive is represented by a BDD. We will present the algorithms for constructing the BDD as well as the nondominance test. Moreover, experimental results from using this symbolic data structure will show the efficiency of our approach in test cases where many candidates have to be tested but only few have to be added to the archive.

1 Introduction

Multi-Objective Evolutionary Algorithms [1,2] using elitism to prevent nondominated solutions from being deleted during generations can be proven to converge to the true Pareto front [3]. Moreover, elitism increases the probability of creating better offspring [1]. Hence, keeping nondominated solutions in an archive A is an important issue in multi-objective optimization. In general there are two strategies for handling archives: (1) using so called *constrained archives* requires a method for *limiting* the number of nondominated solutions in the archive. (2) so called *unconstrained archives*, i.e., archives for storing an unlimited number of nondominated solutions, rely on efficient data structures. Constrained archives are afflicted with the problem of *shrinking* the Pareto front or *oscillating* between different approximations of the Pareto front [4]. On the other hand, keeping an archive dominant-free has a large influence on the computational complexity of the optimization and, thus, narrowing the benefits from using huge or even unconstrained archives.

As a remedy to this problem, several data structures have been proposed in the recent years. Most of these data structures are tree-based [4,5,6]. Unfortunately, the worst case behavior of the nondominance test using these data structures is similar to the complexity of using linear lists, i.e., a new solution, called *candidate*, has to be compared with

* Supported in part by the German Science Foundation (DFG), SFB 694.

S. Obayashi et al. (Eds.): EMO 2007, LNCS 4403, pp. 111–125, 2007.

each solution already in the archive A resulting in $|A|$ comparisons. In this paper, we will present a novel data structure on the basis of Binary Decision Diagrams (BDDs) [7]. Instead of explicitly storing all members in the archive, we encode the region in the objective space weakly dominated by the nondominated solutions as a BDD. A new solution can be tested for nondominance by traversing the BDD. This operation returns a value *true* or *false* and is independent from the archive size, thus allowing a faster nondominance test than any up to now reported archive data structure.

The rest of the paper is organized as follows: Section 2 discusses data structures for archive representation and Section 3 will formally state the problem this paper is dedicated to. In Section 4 our novel symbolic data structure based on BDDs together with the most important algorithms will be presented. First experimental results from comparing our symbolic representation with a linear list data structure will be discussed in Section 5 before we conclude the paper in Section 6.

2 Related Work

While implementing an archive A as linear list requires in the worst case $|A|$ tests to check the nondominance of a candidate resulting in a complexity of $\mathcal{O}(|A| \cdot m)$ in a m-dimensional objective space, tree-like data structures showed improved runtimes: In [6], Mostaghim and Teich proposed the use of so called *quad trees* (cf. [8]). A quad tree is a tree-based data structure where each node has at most 2^m successors where m is the number of objectives. A new vector can be inserted in the quad tree if it is not dominated by any node in the tree. Therefore, a nondominance test is done against the root. If it is not dominated by the root it will be tested against all nodes in the k-th subtree of the root. Here, k is the binary encoding of the $\geq$-relation of the vector's components to the root's components. The dominance test is recursive, i.e., the new solution is next tested against the root of the k-th subtree. If the k-th subtree does not exist the new vector will be inserted. A more sophisticated algorithms is needed in order to keep the data structure *dominant-free*, e.g., if the new solution dominates nodes already in the quad tree. Mostaghim and Teich present experimental results from a comparison of quad trees with linear lists. As a result quad trees outperform linear lists in case of large populations and small archives.

In [5], Schütze proposed the use of a data structure based on m-ary trees, called *dominance decision trees*, as well as algorithms to test for dominance and tree update. Each node has at most m successors where again m is the number of objectives. For the k-th successor of a given node the following properties hold: The first $k-1$ objectives fulfill the $\leq$-relation between the k-th successor and the node. The k-th objective of the k-th successor is greater than the k-th objective of the given node. In [5], several experimental results from comparing dominance decision trees with quad trees and linear lists are presented. In many cases, the dominance decision tree outperforms the linear list and quad trees. Considering problem instances with more than three objectives, the quad trees perform better.

Both data structures have some common disadvantages: The worst case computation time is similar to the case using linear lists, i.e., $\mathcal{O}(|A| \cdot m)$. This is due to the fact