This is the author's version of the work. The definitive work was published in *Proceedings of Design, Automation and Test in Europe (DATE 2008), pp. 158-163, 2008.*
http://www.springer.com/west/home/computer/foundations?SGWID=4-156-22-17372500 ©*Springer-Verlag*
The work is supported in part by the German Science Foundation (DFG), SFB 694.

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 [4–6]. Unfortunately, the worst

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

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 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 that the depth of the trees depends on the order in which nondominated candidates are added to the archive.

A combination of two new data structures, called *dominated trees* and *nondominated trees*, avoids the above mentioned problem and was proposed by Fieldsend et al. [4]. Both data structures are based on the notion of so called *composite points* where each composite point represents a set of so called *constituent points* with a maximum cardinality m, where m is the number of objectives. Composite points can be constructed from a set of vectors by successively determining the maximum (minimum) for each dimension (starting with the first objective) and removing the corresponding vector (a constituent point) from the set. Having m objectives, a maximum of m constituent points contribute to a composite point. Dominated trees allow an efficient *nondominance check* whereas nondominated trees permit an efficient computation of dominated points. In the best case, the nondominance test using dominated trees can be done in $\log_2(|A|/m)$ comparisons between the candidate and the composite points with $|A|$ being the cardinality of the archive. However, in general k additional tests with individual solutions in the archive are required, leading to a complexity of $\mathcal{O}(\log_2(|A|/m)+k\,m)$. Hence, the computational complexity is still dependent on the archive size $|A|$. Before we will present our approach for a nondominance test that is independent from the archive size through representing the region in the objective space weakly dominated by the solutions in the archive by binary decision diagrams, we first will start with a formal introduction to the problem.

3 Problem Formulation

Given the following multi-objective optimization problem:[1]

$$\min f : X \subset \mathbb{R}^n \rightarrow \mathbb{R}^m \tag{1}$$

The goal in multi-objective optimization is to find all *Pareto-optimal solutions* $X_\mathrm{p} \subseteq X$ [9]. A solution x_1 is said to be Pareto-optimal if it is not *dominated* by any solution $x_2 \in X$.

Definition 1 (Pareto dominance (cf. [10])). *For any two solutions x_1 and x_2,*

$$\begin{aligned}
x_2 &\succ x_1 && (x_2 \textit{ dominates } x_1) \textit{ if } \forall i : f_i(x_2) \leq f_i(x_1) \wedge \exists i : f_i(x_2) < f_i(x_1)\\
x_2 &\succeq x_1 && (x_2 \textit{ weakly dominates } x_1) \textit{ if } \forall i : f_i(x_2) \leq f_i(x_1)\\
x_2 &\sim x_1 && (x_2 \textit{ is indifferent to } x_1) \textit{ if } \forall i : f_i(x_2) = f_i(x_1)\\
x_2 &\parallel x_1 && (x_2 \textit{ is incomparable to } x_1) \textit{ if } \exists i, j : f_i(x_2) > f_i(x_1) \wedge f_j(x_2) < f_j(x_1).
\end{aligned}$$

The so called *Pareto-optimal front* is given by $Y_\mathrm{p} = f(X_\mathrm{p}) = \{y \mid y = f(x) \wedge x \in X_\mathrm{p}\}$. Thus the goal in multi-objective optimization can also be stated as: Sort out the nondominated objective vectors $y \in Y_\mathrm{p}$ from a set of all objective vectors $Y = f(X) =$

[1] Without loss of generality, we consider minimization problems in this paper.

$\{y \mid y = f(x) \wedge x \in X\}$. X is called the *decision space*. Y is called the *objective space*. In the following, we will limit our discussion to the objective space.

This so called *nondominance problem* can be divided into two classes (cf. [5]): The *static* nondominance problem is to find the subset of nondominated solutions Y_{p} in a given set Y [11]. The *dynamic* nondominance problem arises during the archive update in Multi-Objective Evolutionary Algorithms: Given a dominant-free archive $A \subseteq Y$ and a sequence of candidate solutions $(y_1, y_2, \ldots, y_l)$. Each candidate solution in this sequence has to be tested for nondominance against the solution in A. If a candidate solution y_i is not weakly dominated by any solution in the archive, y_i has to be added to A. In addition, if y_i dominates solutions from the archive, these solutions have to be removed from the archive keeping it dominant-free.

The most intuitive solution for the dynamic nondominance problem is using a linear list as data structure. In that case, the computational complexity of testing whether a candidate is weakly dominated is linear in the size of the archive, i.e., $\mathcal{O}(|A| \cdot m)$. Removing dominated solutions from A has the same complexity as well.

In the following, we will show how to use Binary Decision Diagrams (BDDs) as data structure to represent the region in the objective space weakly dominated by the solutions in the archive. However, by using BDDs we will not substitute the linear list but rather give a support to it. A Binary Decision Diagram (BDD) is a data structure that can be used to represent Boolean functions [7, 12]. By extending the linear list archive with BDDs, the costs for adding a new candidate increase. On the other hand, the test if a candidate is weakly dominated by a solution in the archive and whether it should be added to the archive is independent from the archive size.

4 Using BDDs for a fast Nondominance Test

In addition to save the nondominated solutions $y \in A$ in the linear, we encode the region that is weakly dominated by these solutions in a single BDD. To test a candidate vector for weak dominance, the binary encoding of its objective values is used to traverse this BDD. This traversal returns *true*, i.e., the BDD is satisfied, if the candidate solution is weakly dominated by at least one solution in the archive A. Otherwise, the traversal returns *false*.

A BDD is a data structure to represent Boolean functions as rooted directed acyclic graphs. Each node in the BDD has either exactly two or none successor. Nodes without successors are called *terminal nodes* and are labeled *true* or *false*. All other nodes are called *decision nodes* and are labeled with a binary variable x of the Boolean function. One edge from the node to one of its successors is labeled 0, the other edge connecting the second successor is labeled 1. This labeling corresponds to the assignment to x. Hence, a path from the root of the BDD to a terminal node is an assignment to the variables of the Boolean function. Moreover, the label of the terminal node determines if the Boolean function is satisfied (*true*) or not satisfied (*false*) under the given variable assignment. On each path from the root to one terminal node each variable appears at most once. The corresponding BDD of the Boolean function $(\overline{x_0} \wedge \overline{x_1} \wedge \overline{x_2}) \vee (x_0 \wedge x_1) \vee (x_1 \wedge x_2)$ is shown in Figure 1.

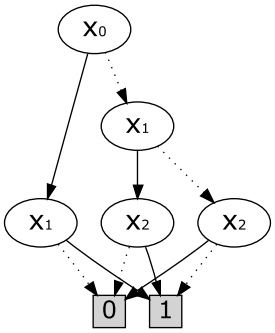

Fig. 1. A Binary Decision Diagram (BDD) of the Boolean function $(\overline{x_0} \wedge \overline{x_1} \wedge \overline{x_2}) \vee (x_0 \wedge x_1) \vee (x_1 \wedge x_2)$. A dotted (solid) edge corresponds the case where the decision variable is 0 (1). The variable order is (x_0, x_1, x_2).

To clarify the goal of our methodology we assume the example from Figure 2. Given is a three-dimensional problem (x, y, z) in which the values are natural numbers in the range from 0 to 15. Therefore, for each dimension exactly four binary variables are needed to encode a natural number, i.e., for the dimension x we have x_3, x_2, x_1, x_0 where x_3 is the most significant bit and x_0 the least significant bit. For the set $A = \{(15, 12, 4), (6, 12, 8), (2, 2, 14), (2, 5, 13)\}$ of nondominated solutions the BDD is given in Figure 2. To test if a candidate vector is weakly dominated by any vector from A we have to use its binary representation to traverse the BDD. For instance the binary representation of the candidate vector $v = (8, 12, 8)$ is $(x_0 = 0, x_1 = 0, x_2 = 0, x_3 = 1, y_0 = 0, y_1 = 0, y_2 = 1, y_3 = 1, z_0 = 0, z_1 = 0, z_2 = 0, z_3 = 1)$ and by traversing the BDD with that assignment the result is $true$ which means that v is weakly dominated by some vector in A.

By using BDDs, it is mandatory that the objective values are encoded by a binary representation. Although our methodology does not limit the objective values to be natural numbers, we will assume that the objective space is given by $Y \subset {\mathbb{N}_0}^m$. Additionally it is recommended that the upper and lower bounds for the values in each dimension are known, such that the minimal number of variables can be used in the BDD. With the known upper bound hi_i and lower bound lo_i for the values in each dimension $i \in \{1, ..., m\}$ the number of required binary variables is given by:

$$\sum_{i=1}^{m} \lceil \log_2(hi_i - lo_i + 1) \rceil \tag{2}$$

Before each objective value is converted to its binary representation, it is normalized by subtracting the corresponding lower bound. To decrease the number of required binary variables even more, all objective values occurring in the i-th dimension are divided by their greatest common divisor. An effect on the number of required binary variables will only take place if the greatest common divisor is greater than 1.

Using the binary encoding of the objective values, we start to construct the Binary Decision Diagram which encodes the region in the objective space weakly dominated

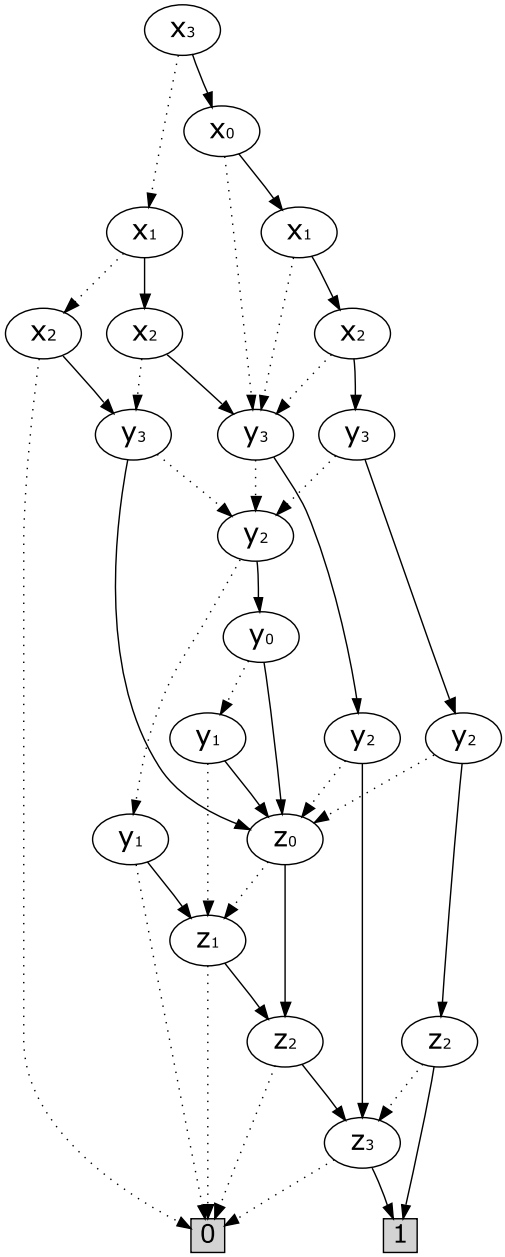

Fig. 2. An example of the BDD that represents the weakly dominated space in a three-dimensional (x, y, z) problem. The variable order is $(x_3, x_0, x_1, x_2, y_3, y_2, y_0, y_1, z_0, z_1, z_2, z_3)$. A dotted (solid) edge corresponds the case where the decision variable is 0 (1).

Algorithm 1 The function bdd_greater&equal constructs a BDD with the variables $var_i[k_i - 1], \ldots, var_i[0]$. The BDD is satisfied, i.e., it returns *true*, if a binary encoded objective value assigned to the variables $var_i[k_i-1], \ldots, var_i[0]$ is greater than or equal to the given constant value represented by a binary number $c_i = (c_i[k_i - 1], \ldots, c_i[0])$. Otherwise, it will return *false*. If $var_i = (var_i[k_i - 1], \ldots, var_i[0])$ is a list of variables with the length k_i, the first element $var_i[0]$ is the least significant bit and the last element $var_i[k_i - 1]$ is the most significant bit.
The algorithm operates from the least to the most significant bit. The if-condition determines whether the k-th position of c_i is 1 or 0. If the if-condition is *true* and $c_i[k]$ is 1, it is mandatory that the corresponding variable $var_i[k]$ is also 1 in order to fulfill the greater or equal condition. Therefore, the variable is appended by a logical AND ($\wedge$). If $c_i[k]$ is 0 a 1 for the corresponding variable $var_i[k]$ fulfills the greater or equal condition albeit the variable assignment of the less significant bits. Therefore, the variable in the else-branch is appended by a logical OR ($\vee$).

```
bdd_greater&equal(var_i, c_i)
{
    bdd b = true;

    for(k = 0; k < k_i; k++){
        if(c_i[k] == 1) {
            b = b ∧ var_i[k];
        } else {
            b = b ∨ var_i[k];
        }
    }

    return b;
}
```

by the archive A. In the following, we assume that we use k_i variables to encode the i-th objective value. We start by introducing an algorithm that constructs a BDD with k_i binary variables $var_i[k_i-1], \ldots, var_i[0]$. This BDD returns *true* if the binary encoding of the i-th objective value of a candidate solution is assigned to the variables var_i and the value is greater than or equal to a given constant value represented by the binary number $c_i[k_i - 1], \ldots, c_i[0]$. Otherwise, it will return *false*. One can think of c_i being the i-th objective value of a solution stored in the archive. That means the BDD covers the statement

$$2^{k_i-1}var_i[k_i - 1] + \cdots + 2^0 var_i[0] \geq 2^{k_i-1}c_i[k_i - 1] + \cdots + 2^0 c_i[0].$$

The construction of this BDD is shown in Algorithm 1.

By Definition 1, a solution x_1 is weakly dominated by a solution x_2 if $\forall i : f_i(x_2) \leq f_i(x_1)$. For a given solution x_2 it is possible to construct a BDD that returns *true* if a candidate solution x_1 is weakly dominated by x_2. Otherwise, it will return *false*. As

Algorithm 2 The function bdd_weakdominated constructs a BDD that returns *true* if the binary encodings of the objective values $f(x_1)$ of a candidate solution x_1 are assigned to $var = (var_0, \ldots, var_{m-1})$ and x_1 is weakly dominated by x_2 with its objective values $f(x_2) = c = (c_0, \ldots, c_{m-1})$. Otherwise, it will return *false*. m is the number of objectives.
In particular, weak dominance is detected if the candidate vector is greater or equal to c in all m dimensions. Therefore, the greater or equal condition has to be fulfilled in each dimension. This is reached by appending the single dimension conditions with a logical AND ($\wedge$).

```
bdd_weakdominated(var, c)
{
    bdd b = true;

    for(i = 0; i < m; i++){
        b = b ∧ bdd_greater&equal(var_i, c_i);
    }
    return b;
}
```

we are only interested in improving the set of solutions in the archive A, dominated and moreover weakly dominated candidate vectors can be disregarded. The binary encoding of $f(x_2)$ is given by $c = (c_0, \ldots, c_{m-1})$ with m being the number of objectives and $c_i = (c_i[k_i - 1], \ldots, c_i[0])$. The BDD is constructed with the variables $var = (var_0, \ldots, var_{m-1})$ which are the binary encodings of the m objective values where $var_i = (var_i[k_i - 1], \ldots, var_i[0])$. Following Definition 1, if all values $f_i(x_1)$ are greater than or equal to the objective values $f_i(x_2)$ for each dimension x_1 is weakly dominated by x_2. For this reason the BDDs constructed by the bdd_greater&equal function from Algorithm 1 have to be connected by applying the logical AND operation. This is shown in Algorithm 2.

Finally, we can construct a BDD that will validate if a candidate solution x_1 is weakly dominated by any solution in the archive A. This can be easily done by combining the BDDs constructed by Algorithm 2. For each solution in the archive A a BDD is created by Algorithm 2 and connected by a logical OR (see Algorithm 3). If the resulting BDD is interpreted as the entire region in the objective space that is weakly dominated by the solutions in the archive, the OR operation is equivalent to a union of the regions weakly dominated by each solution. Note that the weakly dominated region can only grow monotonously in the dynamic nondominance problem. Thus, we do not need to remove any solutions from the BDD even if they are dominated by new candidate solutions. This is illustrated in Figure 3.

With the ability to iteratively add new solutions to our BDD archive, it is possible to test any candidate solution x_1 if it is weakly dominated by any solution in the archive. If the BDD is satisfied by the assignment the variables of the binary encoding of the objective values $f(x_1)$, x_1 is weakly dominated by at least one solution in the archive A. Testing the satisfiability of a BDD is done by traversing it with the given variable

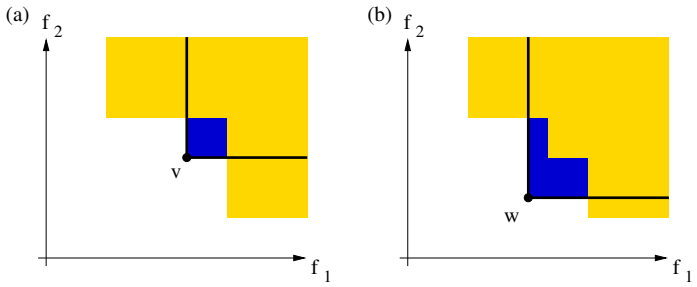

Fig. 3. Objective space with two dimensions. The entire weakly dominated region is built by the union of the regions weakly dominated by each solution vector in the archive. (a) A new candidate solution v incomparable to any other solution in the archive is added. (b) A new candidate solution w which dominates solutions in the archive is added. Note that the dominated solutions need not be removed from the BDD.

assignment. As in BDDs each variable from the root to terminal node appears at most once, the costs of this operation are linear related to the number of used variables. As the number of used variables only depends on the number of objectives (m) and the ranges of the objective values, this test has a computational complexity $\mathcal{O}(m)$ which is independent from the number of solutions in the archive A. On the other hand, the size of the BDD can grow exponentially in the worst case. But even if it does not, we should expect that constructing the BDD or adding solutions to the BDD will be a time consuming operation. Algorithm 3 shows the two main functions for the BDD archive, i.e., adding and testing a candidate solution with its objective values represented by the vector $v = (v_0, \ldots, v_{m-1})$ with $v_i = (v_i[k_i - 1], \ldots, v_i[0])$.

The BDD archive only encodes the region weakly dominated by the solutions in the archive. There is no easy way to extract the solutions $y \in A$ from the BDD. Thus, it can not completely replace the data structure for storing the solutions. Hence, the BDD gives support to the archive in determining whether a candidate solution should be added or not, and this independently from the archive size. As adding solutions to the BDD causes runtimes much greater than adding solution to a linear list, the rate between added and denied candidates should be small.

In our experiments, we extended a linear list archive by our BDD data structure in such a way, that for each candidate solution a test for weak dominance is carried out with the BDD archive. If the candidate solution is weakly dominated, it is rejected. If the candidate solution is incomparable to or dominates solutions in the archive, it has to be added to the linear list as well as to the BDD archive. Furthermore, the dominated solutions have to be removed from the linear list archive.

5 Experimental Results

In this section, we will present experimental results from using our BDD archive in combination with a linear list archive. One of the key factors for the success of the BDD archive implementation is the performance of the used BDD library. The chosen

Algorithm 3 The main functions that are provided by the BDD archive. var is containing the lists of variables for each dimension. The region that is weakly dominated by the archive A is encoded in the BDD b.
Adding a new nondominated vector equals a union on the weak dominated region. Therefore, new weak dominated regions are added by applying a logical OR where $v = (v_0, \ldots, v_{m-1})$ with $v_i = (v_i[k_i - 1], \ldots, v_i[0])$ corresponds the added candidate vector.
By traversing the BDD with a binary representation of the candidate vector $v = (v_0, \ldots, v_{m-1})$ with $v_i = (v_i[k_i - 1], \ldots, v_i[0])$ weak dominance is detected by a resulting *true*. Otherwise, the traversal returns *false*.

```
var;
bdd b=false;

add(v){
    b = b ∨ bdd_weakdominated(var, v);
}

is_weakdominated(v){
    return b.traverse(v);
}
```

library for the tests is Buddy 2.4 [13]. In order to create appropriate test functions, we use adapted versions of some DTLZ functions from [14] (compare Table 1).

First, we will analyze how the size of the archive affects the size of the BDD focusing on the dynamic variable reordering. The reorder algorithm we used is called Sifting [15] and is activated each time the BDD size doubles. Figure 4 shows that dynamic variable reordering has a huge effect on the BDD archive. By using the reorder algorithm the size of the BDD is halved, but it even has a bigger effect on the runtime. The reordering algorithm itself is time consuming. It can be recognized in Figure 4 on the right as a vertical characteristic. On the other hand, the runtime of the reordering algorithm is just a fraction of the whole runtime. This is due to the minimization of the BDD size. Thus, it is recommended to use dynamic variable reordering, which is also used in all following test cases.

In the following all four test functions were used and an appropriate average was calculated over 100 test runs. Figure 5 shows that the BDDs in our test cases are never growing exponentially. The difference in BDD size and time consumption is insignificant between the four test functions as they are all in the same order of magnitude. With a growing archive the size of the BDDs shows in fact an increase that is similar to the logarithmic function. Adding solutions to the archive seems to be a constant time operation independent of the archive size if the curves are considered as linear.

In many cases an increasing number of variables in a BDD leads to a growth of the BDD. In our test cases the number of variables increases if the range of one dimension increases or if the number of dimensions is increased. Therefore, we examined the effect of additional bits for the encoding of the objective values. For this purpose, the

Test Function 1 (TF1)

$$\begin{aligned}
&\text{min } f_1(z) = \tfrac{1}{2}(1+g(z_m))z_1 z_2 \cdots z_{m-1}\\
&\text{min } f_2(z) = \tfrac{1}{2}(1+g(z_m))z_1 z_2 \cdots (1-z_{m-1})\\
&\vdots \quad \vdots\\
&\text{min } f_{m-1}(z) = \tfrac{1}{2}(1+g(z_m))z_1(1-z_2)\\
&\text{min } f_m(z) = \tfrac{1}{2}(1+g(z_m))(1-z_1)
\end{aligned}$$

Test Function 2 (TF2)

$$\begin{aligned}
&\text{min } f_1(z) = (1+g(z_m))cos(z_1\pi/2)\cdots cos(x_{m-2}\pi/2)cos(x_{m-1}\pi/2)\\
&\text{min } f_2(z) = (1+g(z_m))cos(z_1\pi/2)\cdots cos(x_{m-2}\pi/2)sin(x_{m-1}\pi/2)\\
&\text{min } f_3(z) = (1+g(z_m))cos(z_1\pi/2)\cdots sin(x_{m-2}\pi/2)\\
&\vdots \quad \vdots\\
&\text{min } f_m(z) = (1+g(z_m))sin(x_1\pi/2)
\end{aligned}$$

Test Function 3 (TF3)

$$\begin{aligned}
&\text{min } f_1(z) = (1+g(z_m))(1-cos(z_1\pi/2)\cdots cos(x_{m-2}\pi/2)cos(x_{m-1}\pi/2))\\
&\text{min } f_2(z) = (1+g(z_m))(1-cos(z_1\pi/2)\cdots cos(x_{m-2}\pi/2)sin(x_{m-1}\pi/2))\\
&\text{min } f_3(z) = (1+g(z_m))(1-cos(z_1\pi/2)\cdots sin(x_{m-2}\pi/2))\\
&\vdots \quad \vdots\\
&\text{min } f_m(z) = (1+g(z_m))(1-sin(x_1\pi/2))
\end{aligned}$$

Test Function 4 (TF4)

$$\begin{aligned}
&\text{min } f_1(z) = z_1\\
&\text{min } f_2(z) = z_2\\
&\vdots \quad \vdots\\
&\text{min } f_{m-1}(z) = z_{m-1}\\
&\text{min } f_m(z) = 2m - \textstyle\sum_{i=1}^{m-1}[\frac{f_i(z)}{1+g(z_m)}(1+sin(3\pi f_i(z)))]/m
\end{aligned}$$

where $z = (z_1, \cdots, z_m)$ and $0 \leq z_i \leq 1, i = 1, \cdots, m$
and $g(z_m) = z_m \cdot r$

Table 1. The used test functions are based on the DTLZ test functions from [14]: Test function 1 (based on DTLZ1) converges to a linear front, test function 2 (based on DTLZ2) to a sphere, test function 3 (based on DTLZ2) to an inverse sphere, and test function 4 (based on DTLZ7) has a disconnected set of Pareto-optimal regions. With the random values $z_1, \cdots, z_m$ the functions construct vectors in a m-dimensional space. By scaling the objectives $f_i(z)$ as needed and rounding them to integers appropriate test cases are generated. The function $g(z_m)$ determines the distance of the generated vectors to the Pareto-optimal front. It is scaled with the factor r.

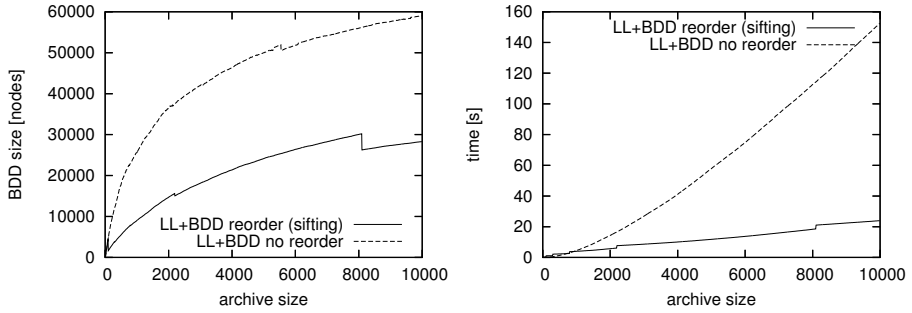

Fig. 4. Example with $m = 3$, $r = 10^{-2}$ and 10 bit encoding per dimension. $10,000$ nondominated solutions from TF1 were added iteratively to the BDD archive. The figures illustrate the size of the BDD and the time consumption with and without dynamic variable reordering.

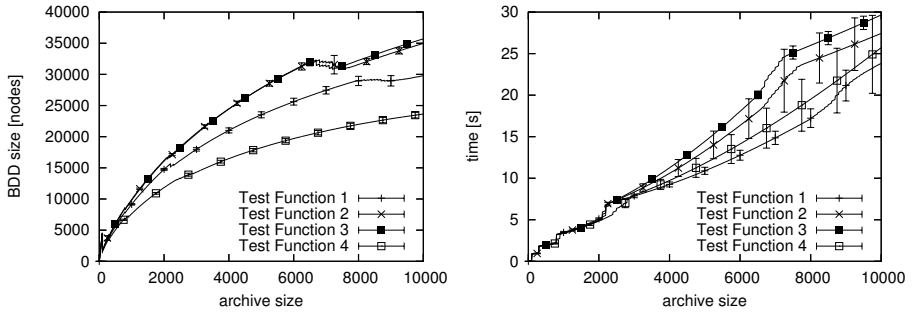

Fig. 5. Example with $m = 3$, $r = 10^{-2}$ and 10 bit encoding per dimension. $10,000$ nondominated solutions from all test functions were added iteratively to the BDD archive. The figures illustrate the size of the BDD and the time required to fill the archive. The vertical bars indicate the standard deviation for 100 runs.

m	bits per dimension	**TF1** time[s]	**TF2** time[s]	**TF3** time[s]	**TF4** time[s]
3	9	3.87 (0.02)	4.41 (0.04)	4.43 (0.04)	4.15 (0.03)
3	12	12.3 (0.98)	12.9 (0.54)	12.4 (0.23)	13.2 (0.43)
3	15	18.08 (1.69)	20.55 (1.13)	20.47 (0.83)	21.36 (0.70)
m	bits per dimension	**TF1** size[nodes]	**TF2** size[nodes]	**TF3** size[nodes]	**TF4** size[nodes]
3	9	8875 (124)	10829 (88)	10729 (138)	7034 (58)
3	12	32646 (213)	35639 (143)	34765 (184)	30892 (299)
3	15	59167 (363)	62569 (301)	61479 (236)	58427 (224)

Table 2. Example with $m = 3$, $r = 10^{-2}$, the number of bits per dimension is varied. The number of variables in the BDD is increasing from 27 over 36 to 45. The time consumption and BDD size are listed for adding $2,500$ nondominated solutions from all test functions to the BDD archive. The small numbers indicate the standard deviation for 100 runs.

test functions were analogous scaled. Table 2 shows the increases in size and time consumption. In our testcases the growth of time consumption and BDD size were linear in the number of variables of the BDD, if the number of dimensions is kept constant.

To test which effect will take place with a growing number of dimensions, test cases with a constant number of BDD variables were created. Table 3 shows that an increasing number of dimensions leads to additional runtime and an increased BDD size. Therefore, a constant number of variable is not a guarantee for a constant time consumption and BDD size. Hence, a growing number of dimensions leads to a worse than linear growth of time consumption and BDD size.

m	bits per dimension	**TF1** time[s]	**TF2** time[s]	**TF3** time[s]	**TF4** time[s]
2	18	7.9 (0.81)	7.7 (0.77)	7.5 (0.50)	5.7 (1.03)
3	12	12.3 (0.98)	12.9 (0.54)	12.4 (0.23)	13.2 (0.43)
4	9	23.6 (0.92)	25.1 (2.02)	17.5 (0.92)	21.7 (0.93)
m	bits per dimension	**TF1** size[nodes]	**TF2** size[nodes]	**TF3** size[nodes]	**TF4** size[nodes]
2	18	15593 (576)	14861 (174)	14709 (98)	12535 (143)
3	12	32646 (213)	35639 (143)	34765 (184)	30892 (299)
4	9	64929 (761)	91755 (1456)	79888 (1468)	74875 (940)

Table 3. Example with $r = 10^{-2}$, the number of dimensions m and the the number of bits per dimension was scaled so that the number of variables in the BDD is constantly 36. The time consumption and BDD size are listed for adding $2,500$ nondominated solutions from all test functions to the BDD archive. The small numbers indicate the standard deviation for 100 runs.

In the next test, we compared the performance of a simple linear list archive with a linear list archive extended by our BDD archive. As all test functions had similar traits for the BDD archive, all following test cases are based on TF1. The test is separated in adding nondominated solutions to an empty archive and testing random candidate solutions for weak dominance with the filled archive. Although it is known that the added solutions are nondominated, a test for weak dominance is needed before they can be added to the linear list archive. In Figure 6 this is illustrated in comparison to the archive size. As expected, Figure 6 shows that filling the archive is much slower in the case when using the BDD archive extension. However, using the BDD archive extension, also the check for weak dominance turned out to be a constant time operation for our test case, i.e., it is in fact independent of the archive size.

Finally, we created a dynamic nondominance problem with TF1. The used number of dimensions is three while 10 bits for each dimension are used to encode the binary values. With the factor r, the quality of the adopted optimization algorithm is biased. One million candidate solutions are created and iteratively added to both variants of the archive. Figure 7 shows the result. In both cases the BDD extended linear list archive turns out to be the better solution for the chosen test cases on long term run. The reason is that the archive is getting fuller and the weak dominance test is getting more expensive if the simple linear list archive is used. On the other hand, the number of added solutions to the archive decreases. In the case that r is 10^{-1} the archive grows slower

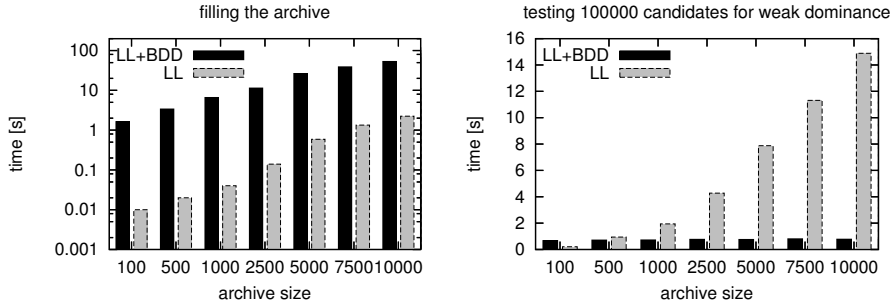

Fig. 6. Example with $m = 3$, $r = 10^{-2}$ and 10 bit encoding per dimension. The used test function is TF1. The archive size is varied. The left figure illustrates the time consumption for filling the archive, the right figure illustrates the time consumption for testing $100,000$ candidate solutions for weak dominance. The used archives are a simple linear list archive and a linear list archive extended by our BDD archive.

compared to the value $r = 10^{-2}$ and the gain of the BDD extended archive is not so clear. But with a growing number of candidate solutions it should get more distinct.

6 Conclusions

In this paper, we have shown that extending an archive by a BDD representation of the region weakly dominated by the solutions in the archive can improve the runtime behavior in the dynamic nondominance problem as it occurs in Multi-Objective Evolutionary Algorithms using archives to establish elitism. Using our symbolic representation, the nondominance test of a candidate is independent from the size of the archive. On the other hand, adding new candidates to the archive is more costly, than using other data structures. Our experimental results have shown that using our proposed nondominance test in case of many candidate tests but only few archive updates clearly outperforms an archive based on a linear list.

In future work, we will combine our symbolic data structure with quad trees [6] and dominance decision trees [5] and integrate it in a Multi-Objective Evolutionary Algorithm.

References

1. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons, Ltd., Chichester, New York, Weinheim, Brisbane, Singapore, Toronto (2001)
2. Coello Coello, C.A., Van Veldhuizen, D.A., B.Lamont, G.: Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic Publishers (2002)
3. Rudolph, G., Agapie, A.: Convergence Properties of Some Multi-Objective Evolutionary Algorithms. In: Proceedings of the 2000 Congress on Evolutionary Computation (CEC00), San Diego, California (2000) 1010–1016

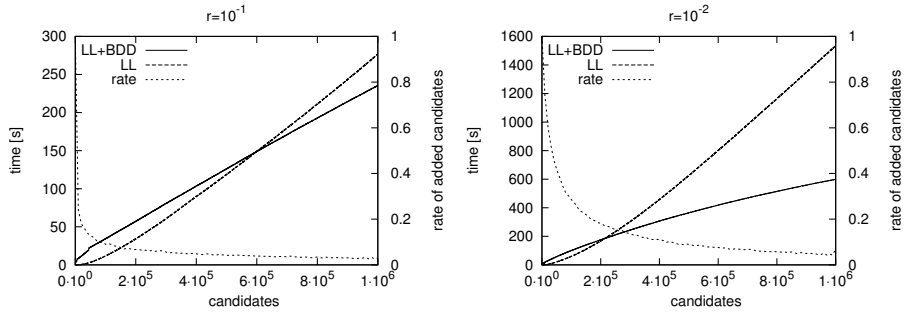

Fig. 7. Example with $m = 3$ and 10 bit encoding per dimension. In the figure on the left the value r is 10^{-1} in the figure on the right $r = 10^{-2}$. $1,000,000$ candidate solutions were generated by TF1 and iteratively tested and, if not dominated, added to the archive. The used archives are a simple linear list archive and linear list archive extended by the BDD archive. Additional to the runtime of the archives the rate of added candidate solutions is stated.

4. Fieldsend, J.E., Everson, R.M., Singh, S.: Using Unconstrained Elite Archives for Multiobjective Optimization. IEEE Transactions on Evolutionary Computation **7**(3) (2003) 305–323
5. Schütze, O.: A New Data Structure for the Nondominance Problem in Multi-Objective Optimization. In: Proceedings of the Second International Conference on Evolutionary Multi-Criterion Optimization, FARO, Portugal (2003) 509–518
6. Mostaghim, S., Teich, J.: Quad-Trees: A Data structure for Storing Pareto-Sets in Multi-Objective Evolutionary Algorithms with Elitism. Advanced Information and Knowledge Processing. In: Evolutionary Multiobjective Optimization – Theoretical Advances and Applications. Springer, London (2005) 81–104
7. Bryant, R.E.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers **C-35**(8) (1986) 677–691
8. Habenicht, W.: Quad Trees, a Data Structure for Discrete Vector Optimization Problems. In: Lecture Notes in Economic and Mathematical Systems, Springer-Verlag (1983) 136–145
9. Pareto, V.: Cours d'Économie Politique. Volume 1. F. Rouge & Cie., Lausanne, Switzerland (1896)
10. Laumanns, M.: Analysis and Applications of Evolutionary Multiobjective Optimization Algorithms. PhD thesis, Eidgenössische Technische Hochschule Zürich (2003)
11. Gupta, P., Janardan, R., Smid, M., Dasgupta, B.: The Rectangle Enclosure and Point-Dominance Problems Revisited. International Journal of Computational Geometry and Applications **7**(5) (1997) 437–455
12. Bryant, R.E.: Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. Technical report, School of Computer Science, Carnegie Mellon University (1992)
13. Buddy: http://sourceforge.net/projects/buddy (2006)
14. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable MultiObjective Optimization Test Problems. In: Proceedings of the 2002 Congress on Evolutionary Computation (CEC02), Piscataway, New Jersey (2002) 825–830
15. Rudell, R.: Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In: ICCAD '93: Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design, Los Alamitos, CA, USA, IEEE Computer Society Press (1993) 42–47