In *Evolutionary Multi-Criterion Optimization*
by Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele (Eds.).
In Lecture Notes in Computer Science (LNCS), Volume 2632, © Springer, Berlin, Heidelberg, 2003.

Solving Hierarchical Optimization Problems Using MOEAs*

Christian Haubelt[1], Sanaz Mostaghim[2], Jürgen Teich[1], and Ambrish Tyagi[2]

[1] Department of Computer Science 12
Hardware-Software-Co-Design
University of Erlangen-Nuremberg
{christian.haubelt, teich}@cs.fau.de
[2] Computer Engineering Laboratory (DATE)
Department of Electrical Engineering and Information Technology
University of Paderborn
{mostaghim, tyagi}@date.upb.de

Abstract. In this paper, we propose an approach for solving hierarchical multi-objective optimization problems (MOPs). In realistic MOPs, two main challenges have to be considered: (i) the complexity of the search space and (ii) the non-monotonicity of the objective-space. Here, we introduce a hierarchical problem description (chromosomes) to deal with the complexity of the search space. Since *Evolutionary Algorithms* have been proven to provide good solutions in non-monotonic objective-spaces, we apply genetic operators also on the structure of hierarchical chromosomes This novel approach decreases exploration time substantially. The example of system synthesis is used as a case study to illustrate the necessity and the benefits of *hierarchical optimization*.

1 Introduction

The increasing complexity of typical search spaces demands new strategies in solving optimization problems. One possibility to overcome the large computation times is the hierarchical decomposition of the search as well as the objective space. The decomposition of optimization problems was already mentioned in [1] and formalized in [2]. While [1] only shows experimental results, Abraham et al. [2] discuss a very special kind of search space which possesses certain monotonicity properties that do not hold in SoC design.

Since Multi-Objective Evolutionary Algorithms (MOEAs) [3] provide good results in non-monotonic optimization problems, we propose an extension of MOEAs towards hierarchical chromosomes. By using hierarchical chromosomes, we capture the knowledge about the search space decomposition. The concept of hierarchical chromosomes is based on the idea of regulatory genes as described in [4] where the activation and deactivation of genes is used to adapt to non-stationary functions. In this paper, the structure of the chromosome itself affects

* This work was supported in part by the German Science Foundation (DFG), SPP 1040.

the fitness calculation, i.e., also the structure of the chromosomes is subject to the optimization. Therefore, two new genetic operators are introduced: (i) *composite mutation* and (ii) *composite crossover*.

In this paper, we consider the example of system synthesis as a case study which includes binding and allocation problems to illustrate the necessity and benefits of hierarchical optimization. When applying hierarchical optimization to system synthesis, we have to consider two extensions: a) *hierarchical problem decomposition* and b) *hierarchical design space exploration*. In previous approaches such as [5], [6], etc., both the application specification and the architecture are modeled non-hierarchically. Here, we introduce a hierarchical approach to model embedded systems. This hierarchical structure could be coded directly in hierarchical chromosomes. We will show by experiment that for our particular problem the computation time required for the optimization decreases substantially.

This paper is organized as follows: Section 2 introduces the problem of hierarchical optimization. Also first approaches to code these problems in hierarchical chromosomes including the genetic composite operators are presented. Afterwards, we apply this novel approach to the task of system synthesis. The benefits of hierarchical optimization in system synthesis are illustrated in Section 4.

2 Hierarchical Optimization Problems

This section describes the formalization of hierarchical optimization problems. Starting from (non-hierarchical) multi-objective optimization problems, we introduce the notion of *hierarchical decision and hierarchical objective spaces*.

2.1 Multi-Objective Optimization and Pareto-Optimality

First, we give a formal notation of multi-objective optimization problems.

Definition 1 (Multi-Objective Optimization Problem (MOP)). *A* multi-objective optimization problem *(MOP) is given by:*

$$\text{minimize } o(x), \text{ subject to } c(x) \leq 0$$

where $x = (x_1, x_2, \ldots, x_m) \in X$ is the decision vector and X is called the search space. Furthermore, the constraints $c(x) \leq 0$ determine the set of feasible solutions, where c is k-dimensional, i.e., $c(k) = (c_1(x), c_2(x), \ldots, c_k(x))$.

The *objective function* o is n-dimensional, i.e., we optimize n objectives simultaneously. There are q constraints c_i, $i = 1, \ldots, q$. Only those *decision vectors* $x \in X$ that satisfy all constraints c_i are in the set of feasible solutions, or for short in the *feasible set* called $X_\mathrm{f} \subseteq X$. The image of X is defined as $Y = o(X) \subset \mathbb{R}^n$, where the objective function o on the set X is given by $o(X) = \{o(x) \mid x \in X\}$. Analogously, the *objective space* is denoted by $Y_\mathrm{f} = o(X_\mathrm{f}) = \{o(x) \mid x \in X_\mathrm{f}\}$.

Since we are dealing with multi-objective optimization problems, there is generally not only one global optimum, but a set of so-called *Pareto-points* [7]. A Pareto-optimal solution x_p is a decision vector which is not worse than any

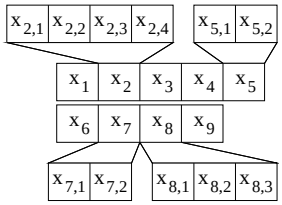

Fig. 1. Example of a hierarchical decision vector consisting of a non-hierarchical part $\mathrm{x}_1, \mathrm{x}_3, \mathrm{x}_4, \mathrm{x}_6, \mathrm{x}_9$ and a hierarchical part $\mathrm{x}_2, \mathrm{x}_5, \mathrm{x}_7, \mathrm{x}_8$.

other decision vector $\widetilde{x} \in X$ in all objectives. The set of all Pareto-optimal solutions is called the *Pareto-optimal set*, or the *Pareto-set* X_p for short. An approximation of the Pareto-set X_p will be termed *quality set* X_q subsequently.

2.2 Hierarchical Search Spaces and Hierarchical Objective Spaces

In the following, we assume that the search space has a hierarchical structure, i.e., each element x_i of a decision vector x itself may be a vector $(x_{i1}, x_{i2}, \ldots, x_{ik_i})$. In general, the decision vector x consists of a non-hierarchical and a hierarchical part, i.e., $x = \left(x^\mathrm{n}, x^\mathrm{h}\right)$. Each element $x^\mathrm{h}_j \in x^\mathrm{h}$ itself may be a decision vector consisting of a non-hierarchical and a hierarchical part. Hence, this structure is not limited to only a single level of hierarchy.

Example 1. Fig. 1 shows the example of a hierarchical decision vector x. The non-hierarchical part x^n of the decision vector is given by the elements $\mathrm{x}_1, \mathrm{x}_3, \mathrm{x}_4, \mathrm{x}_6$, and x_9. The hierarchical part x^h of x consists of four elements $\mathrm{x}_2, \mathrm{x}_5, \mathrm{x}_7$, and x_8.

By using hierarchical decision vectors, we must reconsider the objective functions, too. On the top-level, the objective function is given by: $o(\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_9)$. Since we do not assume monotonicity for the objective functions, we must introduce a decomposition operator $\otimes$ to construct the top-level objective function from deeper levels of hierarchy, i.e., $o(\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_9) = o^\mathrm{n}(\mathrm{x}_1, \mathrm{x}_3, \mathrm{x}_4, \mathrm{x}_6, \mathrm{x}_9) \otimes o^\mathrm{h}(\mathrm{x}_2, \mathrm{x}_3, \mathrm{x}_4, \mathrm{x}_6, \mathrm{x}_9)$, where o^n denotes the partial objective function for the non-hierarchical part of the decision vector. o^h denotes the partial objective function for the elements of the hierarchical part of the decision vector. In general, $o(x) = o^\mathrm{n}(x^\mathrm{n}) \otimes o^\mathrm{h}(x^\mathrm{h})$ where $o^\mathrm{h}(x^\mathrm{h}) = \bigotimes_{x^\mathrm{h}_i} o(x^\mathrm{h}_i)$.

Abraham et al. name three advantages of hierarchical decomposition [2]:

1. The size of each search space of a decision vector $x^\mathrm{h}_i \in x^\mathrm{h}$ in the hierarchical part x^h of the decision vector is smaller than the top-level search space.
2. The evaluation effort for each decision vector $x^\mathrm{h}_i \in x^\mathrm{h}$ of the hierarchical part x^h is low because these elements are less complex.
3. The number of top-level decision vectors to be evaluated is a small fraction of the size of the original search space, when using a hierarchical search space.

The last advantage refers to the fact that a feasible decision vector must be composed of feasible decision vectors of the elements in the subsystem. Thus,

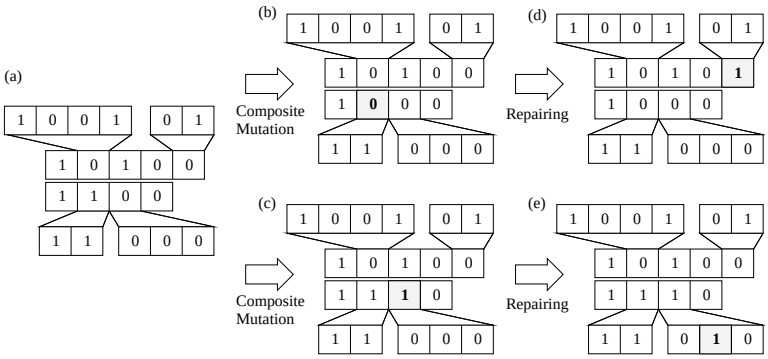

Fig. 2. Example of composite mutation in hierarchical chromosome.

we are able to restrict our search space dramatically. Unfortunately, we cannot generally assume that a Pareto-optimal top-level decision vector is composed of Pareto-optimal decision vectors of the elements in the hierarchical part. Abraham et al. define necessary and sufficient conditions of the decomposition function of the objectives which guarantee Pareto-optimality for the top-level decision vector depending on the Pareto-optimality of elements in the hierarchical part of the decision vector [2]. Pareto-optimal solutions of a top-level decision vector are only composable of Pareto-optimal solutions of the decision vector $x_i^{\mathrm{h}} \in x^{\mathrm{h}}$ iff the decomposition function of each objective function is a monotonic function.

Although this observation is important and interesting, many optimization goals unfortunately do not possess these monotonicity properties. In fact, they depend on the decomposition operator $\otimes$.

2.3 Chromosome Structure for Hierarchical Decision Vectors

Due to the non-monotonicity property of the decomposition of the objective function, heuristic techniques are preferable for the optimization of hierarchically coded problems. Furthermore, Evolutionary Algorithms (EAs) seem to be good at exploring selection and assignment problems. Since we are dealing with multi-objective optimization problems, we propose a special chromosome structure particular to our problem for Multi-Objective Evolutionary Algorithms.

In this paper, we focus on problems where the hierarchical part x^{h} of the decision vector corresponds to a selection problem, i.e., the selection of an element $x_i^{\mathrm{h}} \in x^{\mathrm{h}}$ implies the selection of at least one subelement $x_{i,j}^{\mathrm{h}} \in x_i^{\mathrm{h}}$. For example, consider Fig. 1. The selection of element x_2 implies the selection of at least one of the elements $\mathrm{x}_{2,1}, \mathrm{x}_{2,2}, \mathrm{x}_{2,3}$, or $\mathrm{x}_{2,4}$. Many problems like the selection of scenarios supported by a network processor can be coded that way [8]. This results in a number of network processors optimized for different scenarios.

The advantage of such a strategy should be clear. As in the example of the network processors, we are not only interested in a single type of network processor. In fact, we would like to design a diversity of processors special to different

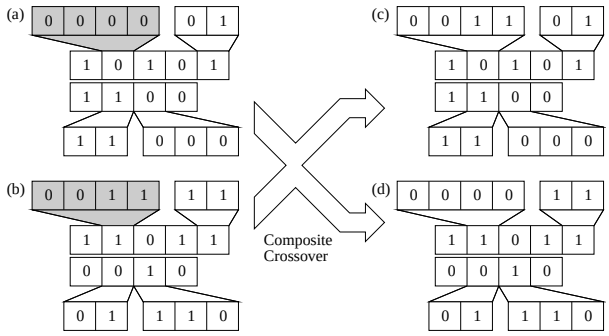

Fig. 3. Example of composite crossover applied to a hierarchical chromosome resulting in one valid and one invalid decision vector.

packet streams in type and number. The knowledge about an optimal implementation of a simple network processor may help to design more sophisticated types.

The novelty of this approach is that not only parameters are optimized but also the structure of such a problem. Note, this differs from [4] where redundant chromosomes are provided for adapting to non-stationary environments. Thus, we need to define two new genetic operators regarding the structure of the chromosomes, namely (i) *composite mutation* and (ii) *composite crossover*.

Composite Mutation The hierarchical nodes $x_i^{\mathrm{h}} \in x^{\mathrm{h}}$ in a hierarchical chromosome directly encode the use of associated elements $x_{i,j}^{\mathrm{h}} \in x_i^{\mathrm{h}}$ of the decomposition of the problem. The composite mutation of a hierarchical chromosome changes a selection bit in a selection list L_{S} from selected to deselected, or vice versa. As a result, leaves of the chromosome are selected or deselected. Fig. 2 shows the two cases that may occur during composite mutation.

Here, we assume that at least one element $x_i^{\mathrm{h}} \in x^{\mathrm{h}}$ has to be selected and if an element x_i^{h} is selected at least one associated subelement $x_{i,j}^{\mathrm{h}} \in x_i^{\mathrm{h}}$ must be selected, too. Fig. 2(a) shows one valid assignment to our decision vector.

In Fig. 2(b), we see the deselection of an element $x_i^{\mathrm{h}} \in x^{\mathrm{h}}$. Since no element $x_i^{\mathrm{h}} \in x^{\mathrm{h}}$ of the hierarchical part x^{h} is selected, the assignment is invalid. Hence, in Fig. 2(d) one of the hierarchical elements x_i^{h} is selected. In the second case (Fig. 2(c)), an element $x_i^{\mathrm{h}} \in x^{\mathrm{h}}$ is selected leading to an invalid assignment (no associated subelement is selected). Thus, we randomly choose one of the associated subelements $x_{i,j}^{\mathrm{h}} \in x_i^{\mathrm{h}}$ (Fig. 2(e)).

Composite Crossover The second operator is called composite crossover. An example of how composite crossover works is shown in Fig. 3. Two individuals (Fig. 3(a) and (b)) are cut at the same hierarchical node in the chromosome. After that we interchange these two subvectors. This results in two new decision vectors as depicted in Fig. 3(c) and (d). This operation again may invalidate one

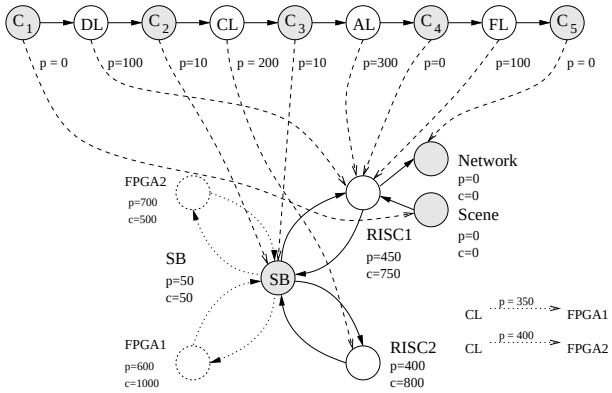

Fig. 4. Specification graph for an MPEG4 coder.

or both assignments (as in Fig. 3(d)). We can repair this infeasibility by randomly choosing one of the associated subelements of a new selected hierarchical element.

3 Case Study: Hierarchical Optimization in System Synthesis

This section illustrates a way to code hierarchical optimization problems. As case study, we use the example of system synthesis. Since Evolutionary Algorithms have been proven to be a good optimization technique in system design [5], we extend this approach by proposing a hierarchical chromosome structure, the objective space decomposition, and the genetic composite operators.

3.1 Specification of Embedded Systems

Blickle et al. propose a graph-based approach for embedded system optimization and synthesis [5]. We introduce this model as a starting point and derive our enhanced hierarchical model subsequently based on this basic model.

The specification model [5] consists of two main components:

- A given functional specification that should be mapped onto a suitable architecture of hardware components as well as the class of possible architectures are described each by means of a universal directed graph $g(V, E)$.
- The user-defined mapping constraints E_{m} between tasks and architecture components are specified in a specification graph g_{s}. Additional parameters which are used for formulating the objective functions and further functional constraints may be assigned to either vertices or edges of g_{s}.

Example 2. The example used throughout this paper is an MPEG4 coder. The problem graph is shown in the upper part of Fig. 4. We start with a given scene (C_1). This scene is decomposed into audio/visual objects (AVO) like images,

video, speech, etc. using the decomposition layer (DL). Each AVO is coded by an appropriate coding algorithm (CL). In the next step (Access Unit Layer. AL), the data are provided with time stamps, data type, etc. The FlexMux Layer (Flexible Multiplexer, FL) allows to group streams with the same quality of service requirements and sends the data to the network (C_5). Between two data flow dependent operations, we insert an additional vertex in order to model the required communication.

The processes of the problem graph are mapped onto a target architecture shown in Fig. 4 (lower part), consisting of two RISC processors, two field programmable gate arrays (FPGAs), and a single shared bus (SB). Additionally, the processor RISC1 is equipped with two special ports.

The mapping edges (dashed edges) relate the vertices of the problem graph to vertices of the architecture graph. The edges represent user-defined mapping constraints in the form of a relation: "can be implemented by". For the purpose of better visibility, additional mapping edges are depicted in the lower right corner of Fig. 4. The mapping edges are annotated with additional power consumptions which arise when a mapping edge is used in the implementation. Furthermore, all resources in Fig. 4 are annotated with allocation cost and power consumptions. These values have to be taken into account if the corresponding resource is used in an implementation of the problem.

3.2 Implementation and the Task of System Synthesis

An implementation, being the result of a system synthesis, consists of two parts:

1. the *allocation* α that indicates which elements of the problem and architecture graph are used in the implementation and
2. the *binding* β, i.e., the set of mapping edges which define the binding of vertices in the problem graph to components of the architecture graph.

The term implementation will be used in the same sense as formally defined in [5]. It is useful to determine the set of feasible allocations and feasible bindings: Given a specification graph g_s and an allocation α, a *feasible binding* is a binding β that satisfies the following requirements:

1. Each mapping edge $e \in \beta$ starts and ends at an allocated vertex.
2. For each problem graph vertex being part of the allocation, exactly one outgoing mapping edge is part of the binding β.
3. For each allocated problem graph edge $e \in (v_i, v_j)$:
   - either both operations v_i and v_j are mapped onto the same vertex
   - or they are mapped on adjacent resources connected by an allocated edge to establish the required communication.

With these definitions, we can define the feasibility of an allocation: A *feasible allocation* is an allocation α that allows at least one feasible binding β.

We define an implementation by means of a feasible allocation and binding.

Definition 2 (Implementation). *Given a specification graph g_s, a* (valid or feasible) implementation *is a pair (α, β) where α is a feasible allocation and β is a corresponding feasible binding.*

Example 3. The dashed mapping edges shown in Fig. 4 indicate a binding for all processes: $\beta = \{(C_1, \text{Scene}), (\text{DL}, \text{RISC1}), (C_2, \text{SB}), (\text{CL}, \text{RISC2}), (C_3, \text{SB}), (\text{AL}, \text{RISC1}), (C_4, \text{RISC1}), (\text{FL}, \text{RISC1}), (C_5, \text{Network})\}$. The allocation of vertices in the architecture graph is: $\alpha = \{\text{SB}, \text{RISC1}, \text{RISC2}, \text{Network}, \text{Scene}\}$. Given this allocation and binding, one can see that our implementation is indeed feasible, i.e., α and β are feasible.

With the model introduced previously, the task of system synthesis can be formulated as a multi-objective optimization problem.

Definition 3 (System Synthesis). *The task of* system synthesis *is the following multi-objective optimization problem:*

minimize $o(\alpha, \beta)$,
subject to:
  α is a feasible allocation,
  β is a feasible binding,
  $c_i(\alpha, \beta) \leq 0,\ \forall i \in \{1, \ldots, q\}$.

The constraints on α and β define the set of valid implementations. Additionally, there are functions $c_i, i = 1, \ldots, q$, that determine the set of feasible solutions. All possible allocations α and bindings β span the design space X.

The task of system synthesis defined in Definition 3 is similar to the optimization problem defined in Definition 1 where α and β correspond to the decision vector x. In the following, we show how to solve this (non-hierarchical) optimization by using Evolutionary Algorithms as described in [5].

3.3 Chromosome Structure for System Synthesis

We start with the chromosome structure of an EA for non-hierarchical specification graphs as described in [5]: Each individual consists of two components, an *allocation* and a *binding*, as defined in the previous section. The mapping task as described in [5] can be divided in three steps:

1. The allocation of resources is decoded from the individual and repaired with a simple heuristic,
2. The binding is performed, and
3. The allocation is updated in order to eliminate unnecessary resources and all necessary edges in the architecture graph are added to the allocation.

One iteration of this loop results in a feasible allocation and binding of the vertices and edges of the problem graph g_p to the vertices and edges of the architecture graph g_a. If no feasible binding could be found, the whole decoding of the individual is aborted.

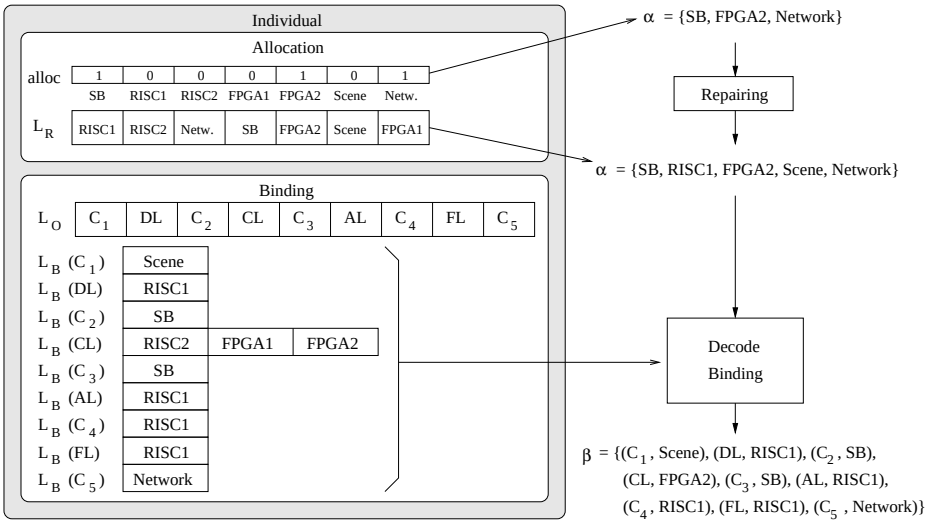

Fig. 5. An example of the coding of an allocation.

The allocation of resources is directly encoded as bit string in the chromosome. This simple coding might result in many infeasible allocations. A simple repair heuristic only adds new resources to infeasible allocation until each process could be bound to at least one resource [5]. The order in which additional resources are added is automatically adapted by a *repair allocation priority list* L_{R}. This list also undergoes genetic operators.

The problem of coding the binding lies in the strong inter-dependence of the binding and the allocation. As crossover or mutation might change the allocation, a directly encoded binding could be meaningless for a different allocation. Hence, an allocation independent coding of the binding is used: A binding order list L_{O} determines the next problem graph vertex to be bound. For each problem graph vertex, a priority list L_{B} containing all adjacent mapping edges is coded. The first edge in L_{B} that gives a feasible binding is included in the binding. Note that it is possible that no feasible binding is specified by the individual. In this case, β is the empty set, and the individual will be given a penalty value as its fitness value.

Finally, resources that are not used will be removed from the allocation. Furthermore, all edges $e \in E_{\mathrm{a}}$ in the architecture graph g_{a} are added to the allocation that are necessary to obtain a feasible binding.

3.4 Hierarchical Modeling

Many applications are composed of alternative functions and algorithms. For example, the coding layer of the MPEG4 coder could be refined by several different coding schemes. We can apply the same refinement also to the architecture graph in order to model different IP cores placed on an FPGA, etc. Thus, hierarchy

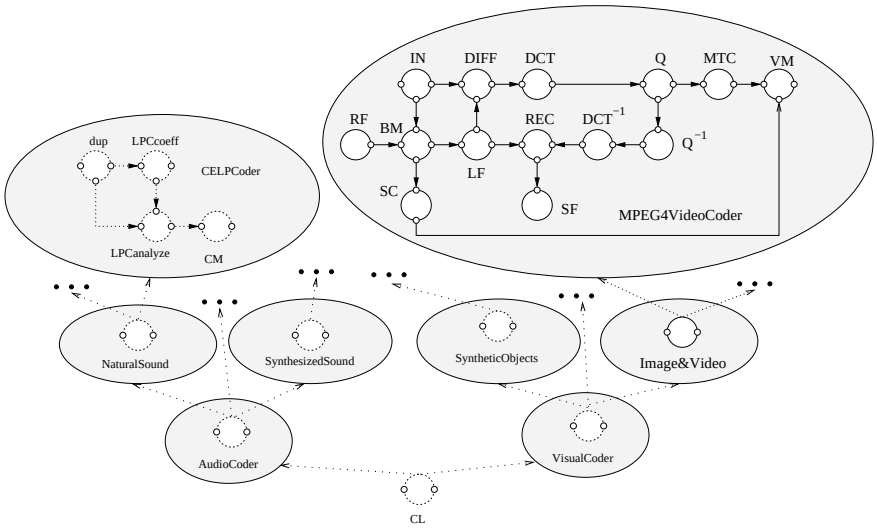

Fig. 6. Refinements of the coding layer of the MPEG4 coder shown in Fig. 4.

is used to model the designer's knowledge about possible refinements and the decomposition of the system.

To model these refinements, we propose a hierarchical extension to the previously introduced model of a specification graph. This hierarchical model [9] is based on hierarchical graphs. Each vertex v in the architecture or problem graph can be refined by a set of subgraphs associated with v. If a subset of these subgraphs is selected as a refinement of a vertex v, we are able to flatten our model, i.e., to replace the vertex v by the selected subgraphs.

Example 4. Fig. 6 shows possible refinements of the coding layer of the MPEG4 coder shown in Fig. 4. There are two types of codings: audio and visual coding. The audio coder subgraph consists only of a single vertex and no edges. In the next level of the hierarchy, for example we can refine the audio coder $\mathrm{v_{ac}}$ by different subgraphs $(\mathrm{NaturalSound}, \mathrm{SynthesizedSound}, \ldots)$. One of the coding schemes for natural sounds is the CELP algorithm (Code Excited Linear Prediction) depicted in the upper left corner of Fig. 6.

Thus, a hierarchical specification graph consists of three parts:

1. a hierarchical problem graph $g_\mathrm{p}(V_\mathrm{p}, E_\mathrm{p})$
2. a hierarchical architecture graph $g_\mathrm{a}(V_\mathrm{a}, E_\mathrm{a})$, and
3. a set of *mapping edges* E_m map leaf problem graph vertices to leaf architecture graph vertices.

3.5 Hierarchical Chromosomes

In Section 3.3, an Evolutionary Algorithm coding for allocations and bindings of non-hierarchical specification graphs was revised. Here, we want to present

EA-based technique for hierarchical graphs. Therefore, we consider an EA that exploits the hierarchical structure of the specification by encoding the structure in the chromosome itself. In order to extend the presented approaches, we have to capture the hierarchical structure of the specification in our chromosomes first. Fig. 7 gives an example for such a chromosome.

The depicted chromosome encodes an implementation of the problem graph first introduced in Fig. 6. The underlying architecture graph is the one shown in Fig. 4. The structure of the chromosome in Fig. 7 resembles the structure of the problem graph given in Fig. 6. The leaves of the chromosome are nearly identical to the non-hierarchical chromosome structure described in Section 3.3 except for the lack of the allocation and allocation repairing list. These have moved to the top-level node of the chromosome (see Fig. 7). Note that the allocation and allocation repairing list form the non-hierarchical part (x^{n}) of the chromosome.

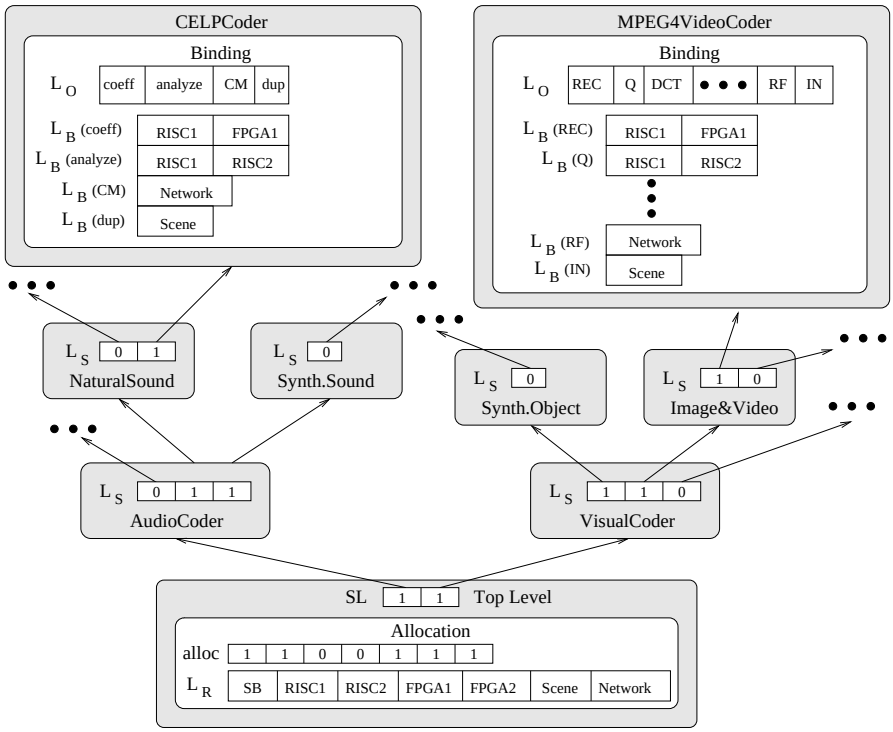

Fig. 7. Hierarchical chromosome structure for the problem graph given in Fig. 6 and the architecture graph given in Fig. 4.

Each hierarchical node in the chromosome resembles a subgraph of the underlying problem graph. For each hierarchical vertex in the corresponding subgraph, the hierarchical node contains a selection list L_{S}. Each entry in this list

describes the use of a subgraph in the implementation. Thus, the selection list L_S corresponds to the hierarchical part (x^h) of the chromosome. If we encounter a subgraph that does not include hierarchical vertices, we encode it by using a non-hierarchical chromosome as described above. The binding order list and the binding priority list form the non-hierarchical part (x^n) at this level of hierarchy.

Despite the modified internal structure, our hierarchical chromosome resembles exactly the non-hierarchical chromosome by still encoding an allocation and a binding. The main difference lies in the variable number of problem graph vertices allocated and bound in any given individual. We therefore still can use the same evolutionary operations, namely mutation and crossover. However, we propose two additional genetic operators making use of the hierarchical structure of the chromosome which have been introduced in Section 2.3.

In summary, composite mutation is used in order to explore the design space of different allocations of leaf graphs of the problem graph (flexibility). The second genetic operator, composite crossover, is used for the same purpose, allowing larger changes as when using the composite mutation operator only.

4 Experimental Results

This section presents first results obtained by using our new hierarchical chromosome structure with the multi-objective evolutionary algorithm SPEA2 [10].

As example, we use the specification of the MPEG4 coder layer. Due to space limitations, we omit a detailed description. A detailed description of this case study including possible mappings of all modules as well as the results can be found in [9]. The specification consists of six leaf graphs for the coding layer, each representing a different coding algorithm. Our goal is to implement at least one of these algorithms while minimizing cost, power, and maximizing flexibility. The underlying architecture consists of 15 resources. The search space for this example consists of more than 2^{200} points.

4.1 Objective Space

Here, we introduce the three most important objectives during system synthesis namely the *cost*, *power consumption*, and *flexibility* of an implementation.

Implementation Cost The *implementation cost* $\text{cost}(i)$ for a given implementation $i = (\alpha, \beta)$ is given by the sum of cost of all allocated resources $v \in \alpha$. Note, due to the resource sharing, the decomposition function of the implementation cost is a non-monotonic function.

Power Consumption Our second objective, the overall power consumption is the sum of the power consumptions of all allocated resources and the additional power consumptions originating by binding processes to resources.

Example 5. The implementation described in Example 3 possesses the overall power consumption $\text{power}(i) = 1620$.

Flexibility The third objective, the reciprocal of a system's flexibility, captures the functional richness an implementation possesses. Without loss of generality, we only treat system specifications throughout this paper where the maximal flexibility equals the number of leaf graphs. For a comprehensive illustration of a system's flexibility, see [11].

4.2 Parameters of the Evolutionary Algorithm

All results were obtained by using the SPEA2 algorithm [10] with the following parameter: In our experiments, we have chosen a population size of 300 and an archive size of 70. The specific encoding of an individual makes special crossover and mutation schemes necessary. In particular, for the allocation α uniform crossover is used, that randomly swaps a bit between two parents with a probability of 0.25. For the lists (repair allocation priority lists L_{R}, binding order lists $\mathtt{L}_O$ and the binding priority lists $L_{\mathrm{B}}(v)$), order based crossover (also names position-based crossover) is applied (see [12]).

For the composite crossover in hierarchical chromosomes, single-point crossover with a probability of 0.25 is chosen. Composite mutation is done on the selection list of each node of the hierarchical chromosome, and by swapping one element of the selection list with a probability of 0.2.

4.3 Exploration Results

As due to complexity reasons, we do not know the true Pareto-set, we compare the quality sets obtained by each approach against the quality set obtained by combining all these results and taking the Pareto-set of this union of optimal points. This set consists of 38 Pareto-optimal design points. A good measure of comparing two quality sets A and B is then to compute the so-called *coverage* $\mathcal{C}(A,B)$ as defined in [13], where $\mathcal{C}(A,B) = \frac{|\{b \in B | \exists a \in A : a \succeq b\}|}{|B|}$.

Hierarchical Chromosomes By using hierarchical chromosomes, the coverage of the Pareto-set increases fast. After $t = 1100$ generations we achieved a coverage of $\mathcal{C}(o(X^{\mathrm{hc}}_{\mathrm{q},t}), o(X_{\mathrm{p}})) \approx 0.83\%$ (average over the results from 5 different runs). As Fig. 8 shows, the hierarchical EA produces only a few Pareto-optimal points at the beginning. This is due to the fact, that the EA is also responsible for the exploration of allocations in the problem graph. The hierarchical EA could reach a full coverage of the Pareto-set when run sufficiently long.

Non-Hierarchical EAs Here, we compare our new approach against a non-hierarchical approach. In this non-hierarchical exploration algorithm, we explore the design spaces individually for all $2^k - 1$ possible combinations where k is the total number of leaf subgraphs in the problem graph. Therefore, we perform six different exploration runs for each individual leaf subgraph, $\binom{6}{2} = 15$ runs for combinations that select exactly two leaf subgraphs, etc. All in all, there are

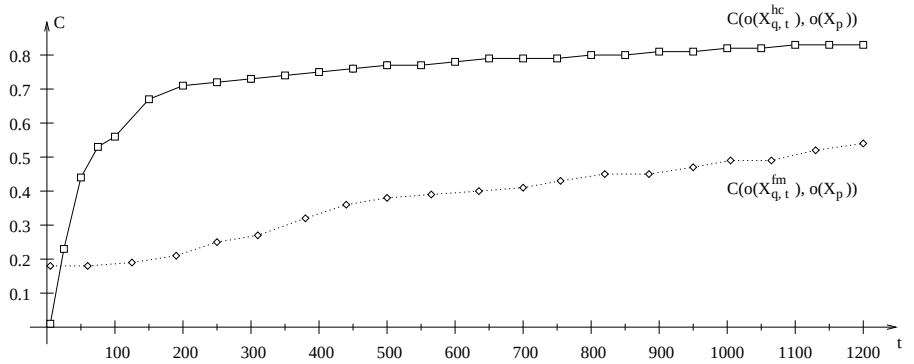

Fig. 8. Coverage of the Pareto-optimal implementations found after a given number of generations compared to the Pareto-set.

$2^6 - 1 = 63$ combinations of leaf subgraphs, where at least one leaf subgraph has to be chosen. Note that this method is in general not a feasible way to go as the number of EA runs grows exponentially with the number of leaf graphs.

For each of these 63 cases we apply the EA for a certain number of generations for each combination k to obtain the quality set of the different leaf graph selections. Since we use the same number of generations for each combination, we simulate the case where each combination is selected with the same probability. With the given archives, we are able to construct the quality set of the top-level design, denoted by $X^{\mathrm{fm}}_{\mathrm{q},t}$, by simply taking the union of all archives of the combinations and calculating the Pareto-optimal points in the union. Fig. 8 shows the result compared with the Pareto-set of our particular problem.

For our particular problem, we see that the hierarchical EAs is superior to the non-hierarchical exploration. At the beginning, the non-hierarchical EA constructs more Pareto-optimal solutions, since the hierarchical EA has to *compose* problem graphs with Pareto-optimal implementations first. By using the non-hierarchical EA, the maximum coverage of the Pareto-front is $\mathcal{C}(X^{\mathrm{fm}}_{\mathrm{q},1200}, X_{\mathrm{p}}) \approx 52\%$. However, as in the case of the hierarchical chromosomes, it should be possible to find all Pareto-optimal solutions by using this non-hierarchical EA.

5 Conclusions

In this paper, we propose an approach for solving hierarchical multi-objective optimization problems. Two challenges have to be considered: (i) the complexity of the search space and (ii) the non-monotonicity of the objective-space. We propose hierarchical chromosomes in order to deal with the complexity of these search spaces. Furthermore, two new genetic operators, namely *composite mutation* and *composite crossover*, have been introduced in order to not only optimize parameters but also the structure of the chromosome. We applied our novel approach to the problem of synthesis of embedded systems. The results to our

particular problem have shown the benefits of this new idea of *hierarchical optimization* by finding solutions with higher convergence than a non-hierarchical method in less number of generations.

In the future, we would like to show the relevance of this approach on other more general optimization problems.

References

1. Josephson, J.R., Chandrasekaran, B., Carroll, M., Iyer, N., Wasacz, B., Rizzoni, G., Li, Q., Erb, D.A.: An Architecture for Exploring Large Design Spaces. In: Proc. of the Nat. Conference of AI (AAAI-98), Madison, Wisconsin (1998) 143–150
2. Abraham, S.G., Rau, B.R., Schreiber, R.: Fast Design Space Exploration Through Validity and Quality Filtering of Subsystem Designs. Technical report, Hewlett Packard, Compiler and Architecture Research, HP Laboratories Palo Alto (2000)
3. Rudolph, G., Agapie, A.: Convergence Properties of Some Multi-Objective Evolutionary Algorithms. In: Proc. of the 2000 Congress on Evolutionary Computation, Piscataway, NJ, IEEE Service Center (2000) 1010–1016
4. Dasgupta, D., McGregor, D.R.: Nonstationary Function Optimization using the Structured Genetic Algorithm. In Männer, R., Manderick, B., eds.: Proceedings of Parallel Problem Solving from Nature (PPSN 2), Brussels, Belgium, Elsevier Science (1992) 145–154
5. Blickle, T., Teich, J., Thiele, L.: System-Level Synthesis Using Evolutionary Algorithms. In Gupta, R., ed.: Design Automation for Embedded Systems. 3. Kluwer Academic Publishers, Boston (1998) 23–62
6. Dick, R., Jha, N.: MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Cosynthesis of Distributed Embedded Systems. In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 17(10). (1998) 920–935
7. Pareto, V.: Cours d'Économie Politique. Volume 1. F. Rouge & Cie., Lausanne, Switzerland (1896)
8. Thiele, L., Chakraborty, S., Gries, M., Künzli, S.: A Framework for Evaluating Design Tradeoffs in Packet Processing Architectures. In: Proceedings of the 39th Design Automation Conference (DAC 2002). (2002) 880–885
9. Teich, J., Haubelt, C., Mostaghim, S., Slomka, F., Tyagi, A.: Techniques for Hierarchical Design Space Exploration and their Application on System Synthesis. Technical Report 1/2002, Institute Date, Department of EE and IT, University of Paderborn, Paderborn, Germany (2002)
10. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical report, Swiss Federal Institute of Technology (ETH) Zurich (2001) TIK-Report 103. Department of Electrical Engineering.
11. Haubelt, C., Teich, J., Richter, K., Ernst, R.: Flexibility/Cost-Tradeoffs in Platform-Based Design. In Deprettere, E., Teich, J., Vassiliadis, S., eds.: Embedded Processor Design Challenges. Volume 2268 of Lecture Notes in Computer Science (LNCS)., Berlin, Heidelberg, Springer (2002) 38–56
12. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. John Wiley & Sons (2001)
13. Zitzler, E.: Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications. PhD thesis, Department of Electrical Engineering, Swiss Federal Institute of Technology (ETH) Zurich (1999)