Visualization of heuristic-based multi-objective design space exploration of embedded systems
Taghavi Razavi Zadeh, T.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Visualization of Multi-Objective Design Space Exploration

System-level simulation frameworks that aim for early Design Space Exploration (DSE) create large volumes of simulation data in exploring alternative architectural solutions and mappings. Interpreting and drawing conclusions from these copious simulation results can be extremely cumbersome. In other domains that also struggle with interpreting large volumes of data, such as scientific computing, data visualization has become an invaluable tool to facilitate the data analysis. Such visualization is often domain specific and has not become widely studied and utilized for evaluating the results of computer architecture simulations. Here, results are usually still presented in a table or displayed in a 2D or 3D graphs and very little research has been undertaken in the use of visualization to support and guide DSE.

In this chapter, we introduce our proposed visualization techniques, which are specially designed for understanding the Multi-objective DSE process of embedded systems that are based on heterogeneous Multi-Processor System-on-Chip (MPSoC) architectures. All the developed visualization techniques are incorporated into a tool, called VMODEX\footnote{Visualization of Multi-Objective Design space Exploration}.

We have two main motivations for visualizing the design space exploration process. The first one is to gain insight into the landscape of the design space. That is, understanding the characteristics of the optimum design points, the correlations among multiple design criteria such as performance, cost, energy consumption and so on, and the relationships between the design parameters and their effects on the criteria. These relations are often not linear and a small change may lead to a completely different result. However, our visualization tool allows designers to easily understand these relations. Our second motivation is to understand how the design space is searched by heuristic searching algorithms. Since in real problems the design space is extremely large, it is not possible to evaluate all the design points in this space. Therefore heuristic search techniques such as evolutionary algorithms are often used to prune the design space and trim down an exponential design space into a finite set.
3.1. Related Work

Our tool enables designers to understand the dynamic search behaviors of such algorithms. For instance, it clearly shows which parts of the design space are not searched at all (no design point is evaluated there), which parts of the design space are searched more often by the MOEA (more design points are evaluated there), which parts of the design space are explored in the later generations, and it illustrates the progress of the searching algorithm in the design space during successive iterations.

The rest of this chapter is organized as follows. In Section 3.1 related work on visualizing the exploration results is discussed. Section 3.2 introduces techniques we have provided for visualizing multi-objective design space exploration. Section 3.3 presents a case study with a Motion-JPEG encoder application to illustrate the benefits of using visualization in the design space exploration process. The data from this case study are used in Section 3.4, which introduces several metrics and their visualization approaches for comparing different subspaces of the explored design space. Finally, section 3.5 concludes this chapter.

3.1 Related Work on Visualizing Exploration Results

Most of the work that has been performed on visualization of computer architecture simulations either focuses on educational purposes (e.g., [5],[7]), tightly couples visualization to one particular architecture simulation environment (e.g., [8],[11]), visualizes only one specific aspect of embedded applications such as memory access patterns [12], cache behavior [13],[15] and data dependencies [16], or only provides some basic support for the visualization of simulation results in the form of 2D (and sometimes 3D) graphs (e.g., [17]). However, in the area of system-level design space exploration, little research has been undertaken on visualization of simulation results and the exploration process.

The work presented in [62,63] provides advanced and generic visualization support, and tries to do so for a wide range of computer system related information. However, these visualizations are not necessarily applicable to computer architecture simulations and in particular to design space exploration, with its own domain-specific requirements. Vista [64] aims at visualization support for computer architecture simulations, but it does not target system-level simulations, which may have a serious impact on the scalability requirements of the visualization, nor does it address the needs for visualization from the perspective of DSE.

In [65], an interactive visual tool is presented to visualize the results from system-level design space exploration experiments. The simulation results are visualized using a coordinated, multiple-view approach, which enables designers to understand the information through different perspectives. It is possible to compare different design points with respect to various characteristics. But this tool does not provide any insight in the searching process as performed by a heuristic method (e.g. MOEA). For example, there is no way to find out which parts of the design space are not searched at all.
This tool is only useful in the case of small design spaces, which can be exhaustively searched.

There are only a few research efforts addressing the visualization of Evolutionary Algorithms (EAs). The most common method for analyzing how EA evolves is monitoring individual fitness values, and creating fitness-versus-generations graphs. Although such straightforward graphs show the quality of the solutions considered during search process, they can only provide a limited amount of information. Due to the large number of individuals in a population, it is not possible to display all of their fitness values. Therefore, displays are usually restricted to the best individual fitness and average population fitness for each generation. Genetic information regarding the position of the discovered solutions in the search space cannot be obtained from these graphs. Other approaches also usually use standard visual representations such as bar charts, line graphs, scatter plots, etc. These graphs can show only one feature of EAs and therefore multiple views are needed to comprehensively understand the EA process.

More complex techniques have focused on how to display the progress of the Multi-Objective Evolutionary Algorithm (MOEA) in design (parameter) space or objective space. Usually, they use 2D or 3D plots in which either variables or objectives are shown. Therefore, two separate views are needed to show the distribution of the solutions in both parameters and objectives spaces. Furthermore, due to the large number of dimensions in practical problems, techniques such as Sammon Mapping should be used to transform higher dimensional search spaces into smaller ones. Hart and Ross proposed the GA VEL visualization tool that provides a means to examine how the genetic operations (crossover and mutation) assemble the optimal solutions, and a way to trace the ancestry of individuals. GA VEL only shows the information that is relevant to the formation of the best solutions. All individuals that do not play any part in propagating individual genes to the final optimal solutions are disregarded. So, it does not provide any information on how the problem space is explored and how an entire population behaves or changes across generations.

In a visualization method is proposed, which shows the individual structure, the fitness function and the objective function in a matrix view. For each generation, an $n \times m$ matrix is used where $n$ is the number of individuals in the generation and $m$ is the number of bits in an individual. Each cell of the matrix is related to a bit of the individual. If the value of a bit is one the corresponding cell is represented in red. The blue cell indicates that the value is zero. The brightness of each row (individual) represents the fitness value. The hue of individuals is changed by their objective values. Although this visualization approach shows three different kind of information in a single view, it has some drawbacks. As they mentioned in their paper, it can be used for showing only binary-coded evolutionary algorithms. Furthermore, the structure of solutions in the parameter space is represented in their encoding format. So, understanding the correlation between the problem space and individual space is difficult. It is not easy to find out how EA explores the parameter space during its generations.

The multi-objective DSE visualization, which is presented in this thesis, is based on
3.2 Multi-Objective DSE Visualization

In this section, we introduce our proposed visualization techniques, which are specially designed for understanding the Multi-objective DSE process of embedded systems that are based on MPSoC architectures. The results of system-level design space exploration of computer architectures are used as the input data to the visualization system. For each design point, its parameter and objective values should be given. The parameter values define the characteristics of a multi-processor architecture instance (e.g. number of processors, types of the processors) and the objective values represent the performance of a design point with respect to the design criteria. The goal of our visualization tool is to help embedded systems designer to 1) gain insight
into the landscape of the design space, and 2) understand how the design space is searched by heuristic searching algorithms.

The multi-objective DSE visualization, which is presented in this thesis, is based on tree visualization. In comparison with the existing tools such as parallel coordinate, scatterplot matrix and influence explorer, our proposed DSE tree is more structured in terms of dominance relation between design points. In multi-objective optimization problems, the concept of dominance is generally used to compare the quality of different solutions and determine the trade-off among multiple objectives. As we will describe in the following sections, in the DSE tree, design points are distributed in the three levels (i.e. global Pareto, local Pareto, non-Pareto) based on their dominance relations with respect to each other. Thus, just by looking at the tree, one can understand how good a solution is in comparison to other solutions. However, in the existing tools, the dominance relation is not considered for representing the solutions. They just plot each individual design point independently from other points. Furthermore, in our proposed DSE tree, the design points are clustered in a way that nicely matches with natural way of thinking of embedded systems designers. In this thesis, we refer to the clusters as subspaces. By modeling the design space as a tree, it is divided into exclusive subspaces, such that each subspace represents a unique instance of the architecture platform. On the other hand, solutions inside a subspace have exactly the same architecture components but the way that the application is mapped onto those components is different. This kind of clustering of design point is quite helpful for embedded systems designers to explore their design problems. They are mainly interested to investigate how objective values are being changed by modifying the mapping of applications within a unique architecture instance (solutions in the same subspace). Moreover, we have provided several techniques in our tool that allow designers to evaluate and compare the properties of the discovered design points in different subspaces (different architecture instances) from various perspectives.

Note that in this thesis, we use the conventional tree diagram to describe the concepts of our tree model of the design space. However, to more effectively utilize the screen space, other alternatives for representing the tree structures such as hyperbolic tree, cone tree, space tree, etc. could also be used.

3.2.1 Modeling the Design Space as a Tree

As it is conceptually shown in Figure 3.1, we model the design space as a tree. In this section, we explain how a design space can be modeled as a tree. Table 3.1 presents the set of mathematical symbols that are used for constructing the tree model of the design space. The tree has three segments: the parameters, architecture-dependent objectives and design points. Equation 3.1 formally describes tree segments.

\[
T_{DS} = Seg_{par} + Seg_{arch-dep-obj} + Seg_{dp}
\]  

(3.1)

The parameters and design points segments are explained in this section while the architecture-dependent objectives segment will be described in the next section.
Table 3.1: Table of symbols for the tree model of the design space

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{DS}$</td>
<td>Tree model of the design space</td>
</tr>
<tr>
<td>$Seg_{par}$</td>
<td>Parameters segment</td>
</tr>
<tr>
<td>$Seg_{arch-dep-obj}$</td>
<td>Architecture-dependent objectives segment</td>
</tr>
<tr>
<td>$Seg_{dp}$</td>
<td>Design points segment</td>
</tr>
<tr>
<td>$</td>
<td>Seg</td>
</tr>
<tr>
<td>$LVL_{par}$</td>
<td>Abbreviation for parameter</td>
</tr>
<tr>
<td>$n$</td>
<td>Total number of parameters</td>
</tr>
<tr>
<td>$DP$</td>
<td>The set of all evaluated design points</td>
</tr>
<tr>
<td>$GP$</td>
<td>The set of global Pareto optimal points</td>
</tr>
<tr>
<td>$LP$</td>
<td>The set of local Pareto optimal points</td>
</tr>
<tr>
<td>$NP$</td>
<td>The set of non-Pareto optimal points</td>
</tr>
<tr>
<td>$Sub$</td>
<td>Abbreviation for subspace</td>
</tr>
<tr>
<td>$k$</td>
<td>Total number of subspaces in the $T_{DS}$</td>
</tr>
<tr>
<td>$map(p)$</td>
<td>The mapping of the application onto the architecture components for the design point $p$</td>
</tr>
</tbody>
</table>

Parameters Segment

In this segment, each level shows one parameter of the design space, such as the number of processors in the MPSoC platform. So, the number of levels in this segment is equal to the total number of parameters in the design space; formally:

$$Seg_{par} = LVL_{par_1} + LVL_{par_2} + ... + LVL_{par_n}$$

$$|Seg_{par}| = n$$  \hspace{1cm} (3.2)

For better understanding the process of modeling the design space as a tree, an example is shown in Figure [3.2]. In the tree illustrated in this figure, the design space has four parameters thus there are four levels in the parameters segment, which are:

1. **Number of processors level**: this level shows the number of processors (independent of their type), which are allocated for executing the application tasks. In this particular example, the desired maximum number of processors is two and therefore there are two nodes at this level indicating one or two used processors.

2. **Processors type level**: nodes at this level represent different combinations of processor types. Each node becomes a child of the node at the previous level that shows the number of processors used in its combination. In this example, the platform architecture consists of one Application Specific Instruction Processor (ASIP) and two microprocessors (mPs). Note that only those combinations are shown, which lead to the platform instances that are capable of executing the
3.2. **MULTI-OBJECTIVE DSE VISUALIZATION**

By modeling the design space as a tree, there is no limitation on the number of design variables as each parameter is located at one level of the tree. It should be mentioned that, in principle, the designer has total freedom of ordering the parameters in the levels of the tree. However, putting more important parameters higher up in the tree facilitates the information organization in such a way that it produces sub trees, which are more likely to show a better view of the design space characteristics. Because the more important design points (according to the design parameters) are clustered in only one sub tree, the designer can easily select that sub tree to investigate and compare these design points. On the other hand, by putting more important parameters down in the tree, the design points with the same parameter are distributed in several sub trees.

**Figure 3.2:** Modeling the design space as a tree

- **Application (feasible combinations):** For instance, in this example, using only one ASIP is not sufficient for executing the application and thus there is no node in the tree that indicates one ASIP.

- **Number of memories level:** this level shows the number of memories used in a platform instance.

- **Memory types level:** at this level, the type of memories is shown. In this example, there are two types of memory: one Static RAM (SRAM) and one Dynamic RAM (DRAM).

---

3. **Number of memories level:** this level shows the number of memories used in a platform instance.

4. **Memory types level:** at this level, the type of memories is shown. In this example, there are two types of memory: one Static RAM (SRAM) and one Dynamic RAM (DRAM).
3.2. Multi-Objective DSE Visualization

Design Points Segment

This segment includes the design points searched by the MOEA. Here, a design point is defined as a specific instance of the architecture platform as well as a task and communication mapping. Each point is shown as a node, which is a child of its corresponding architecture. This means that, for each solution, its parents at the previous levels show its design parameters. For instance, the solution labeled by “Global Pareto 2” has the following architectural components: two processors, of which one is an ASIP and the other one is mP, and two memories, of which one type is DRAM and another type is SRAM. Design points are distributed in three levels: global Pareto, local Pareto and non-Pareto. There is also another level in this segment, called distance level, which is used for categorizing the non-Pareto points based on their distance from parents. Thus, the design points segment consists of four levels, as follows:

\[
\text{Seg}_{dp} = \text{LV}_{gp} + \text{LV}_{lp} + \text{LV}_{dis} + \text{LV}_{np}
\]

\[
|\text{Seg}_{dp}| = 4
\]

Global Pareto Level

This level shows the global Pareto points found by the MOEA. The solutions at this level are better than all other solutions in the entire design space; formally:

\[
\text{LV}_{gp} = \{p \mid p \in GP\}
\]

\[
GP = \{p \in DP \mid \exists p' \in DP : p' < p\}
\]

The solutions in the global Pareto optimal set has the following two properties:

1. They are non-dominated with respect to each other (no one is absolutely better than another one); formally: \( \forall p, p' \in GP : p \sim p' \)

2. Each point that is not part of the global Pareto set, is dominated by at least one global Pareto point; formally: \( \forall p \in (DP - GP) \exists p' \in GP : p' < p \)

In this thesis, we simply refer to the global Pareto optimal solutions as Pareto optimal solutions.

Local Pareto Level

By modeling the design space as a tree, it is divided into exclusive subspaces. Each subspace represents a unique combination of design parameters (in our case, a unique instance of the architecture platform). On the other hand, solutions inside a subspace have exactly the same architecture components (have the same parents at the parameter levels) but the way that the application is mapped onto those components is different. Figure 3.3 illustrates the concept of subspaces. It shows the same tree as Figure 3.2. However, in this figure, the four subspaces of the design space that contain some evaluated design points are indicated by contour lines and are labeled from \( \text{Sub}_1 \) to \( \text{Sub}_4 \). For instance, all design points in the subspace \( \text{Sub}_4 \)
have two microprocessors and one SRAM memory in their underlying architecture. However, the binding of application tasks and channels to these resources is different. In total, there are 11 subspaces in the example shown in Figure 3.3. Actually, The number of nodes at the last level of the parameters segment (memory type level in Figure 3.3) indicates the number of subspaces. Thus, we can consider the tree model of the design space as a union of several subspaces:

\[ T_{DS} = \bigcup_{i=1}^{k} Sub_i \]

(3.5)

Where \( k \) is the total number of subspaces (in our example \( k = 11 \)).

For each \( p, p' \in Sub_i \) following conditions are true:

1. \( \forall j \in \{1, 2, .., n\} \; par_j(p) = par_j(p') \)

2. \( map(p) \neq map(p') \)

At the local Pareto level, the local Pareto points are shown. In each subspace, the Pareto optimal solutions are called local Pareto. Thus, a design point is called a local Pareto point if within the design points with the same architecture components (same
allocation) but with different mappings (different binding), there is no point dominating that one. So, it is an optimal solution with respect to a specific architectural instance. However, in the entire design space, a design point might exist which dominates the local Pareto point. Equation 3.6 gives a formal definition of the local Pareto concepts:

\[ LV_{lp} = \{ p \mid p \in LP \} \]

\[ LP = \bigcup_{i=1}^{k} LP_i \]

\[ LP_i = \{ p \in Sub_i \mid \exists p' \in Sub_i : p' < p \} \]  

Equation 3.6

It is clear that all the global Pareto points are local Pareto points as well. However, not all the local Pareto points are global Pareto points and therefore we use a relation node at the global Pareto level to make a connection between them and the previous level. These nodes are labeled with R in Figure 3.2. Equation 3.7 describes this relation between the global Pareto and local Pareto sets:

\[ p \in GP \Rightarrow p \in LP \]

\[ p \in LP \nRightarrow p \in GP \]  

Equation 3.7

**Non-Pareto Level** All the other design points are placed at the non-Pareto level. Each non-Pareto point is dominated at least by one point in the local Pareto set of its corresponding subspace \((LP_i)\); formally:

\[ LV_{np} = \{ p \mid p \in NP \} \]

\[ NP = \{ p \in DP \mid \exists p' \in LP_i : p' < p \} \]  

Equation 3.8

Each non-Pareto point becomes a child of a local Pareto point, which dominates it. If a design point is dominated by more than one local Pareto point, we calculate the Euclidean distance, in the objective space, between the dominated point and each dominating local Pareto point and the design point becomes the child of the local Pareto point with the smallest distance. A smaller distance means that the points are more similar according to the objectives.

**Distance Level** For easier interpretation and better analysis of the design points, the children of a local Pareto point are categorized into two groups according to their Euclidean distance from their parent. If the distance between a non-Pareto point and its corresponding local Pareto point is more than a certain threshold (determined by the designer), it becomes a child of a "High" distance node, otherwise it becomes a child of a "Low" distance node. Thus, there are two types of nodes at the distance level; formally:

\[ LV_{dis} = \{ d \mid d \in \{ Low, High \} \} \]  

Equation 3.9
3.2.2 Showing Objectives

Various features can be considered as objectives for design space exploration of embedded systems, such as processing time to complete a particular task or application, power consumption, energy consumption, architecture cost, latency, utilization, temperature, physical size, weight and so on. We classify objectives, which are considered in the DSE process, in two groups: primary objectives and secondary objectives. The primary objectives are directly used in optimization process and depending on their values the optimization algorithm walks through the design spaces to find optimum design points with respect to the primary objectives. The secondary objectives are not considered as optimization goals. However, they provide supportive information and help designer to do further analysis on the characteristics of design points.

The primary objectives are directly shown in the DSE tree and therefore just by looking at the DSE tree the designer can see their values. However, the secondary objectives are shown just after the designer requests to see them. If the designer is interested to know about the secondary objectives for some specific solutions, it is possible to select those solutions to see their secondary objective values.

Showing Primary Objectives in the DSE Tree

For representing the primary objectives in the DSE tree, we have divided these objectives into two groups:

1. Objectives that are only dependent on the architectural components, denoted as $obj_{arch-dep}$
2. Objectives that are dependent on the mapping, denoted as $obj_{map-dep}$

Those primary objectives that are only dependent on the architectural components, not on the mapping decision are shown in the architecture-dependent objectives segment ($Seg_{arch-dep-obj}$). An example is the architecture cost as shown in Figure 3.2. These objectives can be computed after the parameters segment, since all components are known. Furthermore, all design points with the same architecture have the same value in terms of these objectives. Therefore, we add an extra segment between the parameters and design points segments, which shows the values of architecture-dependent objectives for different architectures. The definition of the architecture-dependent objectives segment is as follows:

$$Seg_{arch-dep-obj} = LV L_{obj_{arch-dep_1}} + LV L_{obj_{arch-dep_2}} + ... + LV L_{obj_{arch-dep_m_1}}$$

$$|Seg_{arch-dep-obj}| = m_1$$  \hspace{1cm} (3.10)

Where $m_1$ is the number of architecture-dependent objectives. In Figure 3.2, the cost is shown with a different shape; a circle, since it is an objective and not a design parameter. For a better view, the size of the circle becomes bigger as the cost increases.
In the case that more primary objectives exist, which are only dependent on architectural components, such as weight, physical size, etc., extra levels can be added in the segment $Seg_{arch-dep-obj}$ in which each level shows one objective. For better distinguishing different objectives, each one can be shown with a different color and shape, like rectangle, trapezoid, pentagon, etc. In each subspace, the values of architecture-dependent objectives for all design points are the same. For instance, in Figure 3.3, the cost of all design points in $Sub_4$ is 160. Formally, for each $p, p' \in Sub_i$ the following condition is true:

$$\forall j \in \{1, 2, \ldots, m_1\} \quad obj_{arch-dep_j}(p) = obj_{arch-dep_j}(p')$$

Those objectives that are dependent on the mapping (binding application tasks and channels to the hardware components) are shown in a design point node. In Figure 3.2, there are two primary objectives that are dependent on the mapping: the processing time (i.e. the time needed to execute the given application) and energy consumption of the whole system. The color of the node itself represents the processing time. Colors are varied from yellow to red with all color grades in between. Nodes with the lowest processing time are yellow and nodes with the highest processing time are red. The size and color of the third dimension of a design point node shows the energy consumption. As the energy consumption increases, the size of the third dimension becomes bigger and its color becomes darker. The color legends for the processing time and energy consumption are shown in Figure 3.4.

Parameter nodes, however, do not represent single design points and therefore do not have the direct notion of processing time or energy consumption. For this reason, there are some options to color the parameter nodes: based on the average, minimum, or maximum of either processing time or energy consumption of the design points in their sub trees. The color of parameter nodes that have no data node (i.e., do not have any DSE data) is white. In Figure 3.2, the minimum processing time is chosen for coloring parameter nodes.

It should be mentioned that although we only show three objectives (architecture cost, processing time and energy consumption) in this thesis, VMODEX is able to easily visualize more than three objectives. For those objectives that are only dependent on the architectural components, extra levels can be added in the architecture-dependent objectives segment. Showing more objectives that are dependent on the mapping is
also easy. Each node has some attributes like shape, orientation, size, color, transparency, texture, border, glow, etc. Each attribute can be assigned to one objective. For the purpose of illustration, Figure 3.5 shows an example of representing a solution with six objectives. In this figure, texture density indicates average utilization (denser texture means higher utilization), the size of the glow shows the temperature (bigger glow means higher temperature) and the size of the trapezoid indicates the weight (bigger trapezoid means heavier weight). The other three objectives are the same as Figure 3.2. Note that the examples shown in Fig. 5 would become the tree nodes in our DSE tree (For the case that we have 6 objectives). Since in the DSE tree the architecture-dependent objectives (e.g. cost and weight) are shown as separate nodes in the segment Segarch−dep−obj, the corresponding nodes are drawn above the 3D rectangle. Figure 3.5(a) represents a solution that is superior to the solution shown in Figure 3.5(b) with respect to the all six objectives. Thus, the solution in Figure 3.5(a) has a lower processing time and energy consumption, and its average utilization is higher. It also produces less heating and it is cheaper and lighter.

As we described above, our visualization tool is extendable to show more than three objectives. However, multi-objective design space exploration problems are usually based on only two or three objectives, typically performance, cost and power. The maximum number of objectives that we found in related work is six [74].

**Showing the Mapping Decision**

In the Sesame simulator as well as in many other system-level simulation frameworks, the application behavior is modeled as a process network. A process network is a computational model of the application and uses a directed graph notation, where each node represents a process and each edge represents a one-way (FIFO) communication channel between two processes. Figure 3.6(a) represents an example process network graph, which has five processes (A−E) and six communication channels (1−6).

We visualize the process network graph in a way that shows the mapping decisions as well. That means that it shows how the application is being mapped to the underlying architecture both in terms of processes and communication channels. The shape and the color of each node in the graph represent the type of the processor executing the
corresponding process. For example, a green rectangle for one processor type and a blue pentagon for another type. If there are multiple processors of the same type in the platform architecture, then they are differentiated using different variants of the same color such as light green and dark green.

If two communicating processes are mapped onto the same processor, then their communications are done internally and therefore communication channel(s) between them are mapped onto the processor in question. In the process network graph these internal communications are represented by a solid line with the same color as the corresponding processor. In the case that a channel is mapped onto an external memory, a dashed line is drawn. The color of the line and style of the dash represents the memory type. Memories with the same type are shown by the same dash style but with different variants of the same color.

Figure 3.6(b) shows how our visualization model shows the process network graph from Figure 3.6(a). As can be seen in this figure, processes A, B, C and channels 1 and 2 are mapped to the same processor (ASIP-1). Process D is executed on the same processor type but on a different processor as process A (ASIP-2). The type of the processor executing process E is different from the others since it is shown with a blue pentagon (mP). Channels 3, 4, 5 and 6 are mapped to memories (not processors) as they are shown with dashed lines. Channels 3 and 4 are mapped to the same memory (DRAM-1). Channel 6 is mapped to another memory but with the same type (DRAM-2) and Channel 5 is mapped onto a different memory type because it has a different dash style and different color (SRAM).

Note that the colors assigned to each processor type or memory type in the visualization of the mapping decision are the same as colors used in the visualization of utilization, latency and processes execution time (these visualizations will be explained later). For instance, in all of them the ASIP processor is shown by a green color. Thus, these visualizations are consistent with each other and the designer can easily understand the effect of different mapping decisions on the utilization of architecture components as well as processes execution times and read/write latencies.
3.2. Multi-Objective DSE Visualization

Showing Secondary Objectives

Since the secondary objectives are not used in the optimization process and do not have any effect on the exploring procedure, we do not show them directly in the DSE tree. However, if the designer is interested to know about the secondary objectives for some particular solutions, it is possible to select those solutions to see their secondary objective values. The secondary objectives provide additional information and help the designer to do more in-depth analysis on the characteristics of design points. VMODEX is designed to show various properties of design points, which are considered as secondary objectives. Thus, the designer is able to look at the evaluated data from different perspectives and thereby gets a comprehensive view on the problem under study. Furthermore, for better understanding and easier analysis, we propose several visualization techniques for showing the secondary objectives. In the following subsections the provided secondary objectives and their visualization methods are described.

Showing Utilization

For better understanding the utilization of each separate hardware component and also for easier comparison between different design points, we have visualized the utilization property. Here, we define utilization as the percentage of the time that a hardware component was busy. For visualizing the utilization, the platform architecture is shown as a directed graph in which each node represents an architectural component and the edges show connectivity between components.

Each node (component) is filled with its corresponding color (see subsection 3.2.2) in a way that the size of the colored part represents the percentage of the time that the corresponding component was busy. The components in the platform architecture that are not allocated are shown by gray borders. The average utilization is written at the bottom of the visualization.

Figure 3.7 shows an artificial example of utilization visualization for a platform architecture consisting of two Application Specific Instruction Processors (ASIPs), two microprocessors (mPs), one Static RAM (SRAM) and two Dynamic RAMs (DRAMs). In this example, the mP-1 is not allocated since it is shown by gray border. The utilization of ASIP-1 and DRAM is 100% while for the Bus it is almost 75% and for the other components it is less than 50%. By visualizing the utilization, besides understanding the utilization of each architecture component individually, the designer is able to see how these components are connected together, operate with each other and which resources are shared.

Showing Processes Execution Time

VMODEX also allows to modify the mapping visualization in such a way that it shows the processes execution time as well. Thus, it also shows the amount of time that each process was executed by the processor onto which it has been mapped. In the mapping visualization, the shape and color of each process in the process network graph represent the type of the processor executing the corresponding process. To add information about how long a processor was busy with
executing a process, instead of fully coloring the process nodes in the graph, the size of the colored part represents the percentage of time that its assigned processor was busy executing it. Figure 3.8 shows the same mapping decision as shown in Figure 3.6(b), but the processes execution time information is added. From this figure we can see that processor ASIC-1 was executing process \( B \) in most of its busy time, while executing process \( A \) and \( C \) took relatively a small amount of time. The processes execution time visualization enables designers to explore the effect of different mappings on the execution time of processes.

**Showing read/ write latency** Similarly, VMODEX is also capable to show the amount of time that each process is waiting for read and write communications. Since in process network graph the communication channels between processes are shown, we visualize it in a way that shows the read and write latencies as well. The color-coding, from blue to red, is utilized to represent the waiting time. In Figure 3.9(a) the color legend for latency is shown. In the process network graph, each process node is divided into two halves. The color of the top part shows the write latency and the color of the bottom part represents the read latency. By clicking on each part, the exact value of latency is shown. The mapping information is also added to the latency visualization as well. Therefore, the designer can easily investigate the effect of different mappings on the latency. The shape and color of the third dimension of each process in the graph represent the type of the processor executing the corresponding

![Figure 3.7: Utilization visualization](image)

![Figure 3.8: Visualization of processes execution time](image)

![Figure 3.9: Latency Visualization](image)
process (similar to the mapping visualization). Figure 3.9(a) shows an example of latency visualization. In this figure, processes B and C have high write latencies since the color of their top parts are red. By including mapping information in the latency visualization we can understand that the reason of these high latencies is because of mapping channels 3 and 4 on the same memory (both of them are shown by dashed red lines). Thus, processes B and C are competing with each other to write to the shared memory. For the other processes both the read and write latencies are relatively low.

**Showing Processing Time of Processor Operations**  Basically, processors perform three operations: read, write and execution [23]. There is an option in VMODEX to see the percentage of time that a processor was doing each operation separately. To show this, for each processor in the architecture platform, a stacked bar chart is drawn. Each operation is shown as a stack in the bar with a different color. The stack height represents the percentage of time that the processor was performing the corresponding operation. Figure 3.10 represents an example of visualizing the processing time of different processor operations for a platform architecture consisting of two ASIPs and one mP.

**Showing Generation Numbers**  In some cases, the designer wants to know what interesting design points are evaluated in which search generations. Therefore, we have developed a method in VMODEX that allows the designer to easily find out this kind of information. During the process of design space exploration using an MOEA, some design points that are near to the optimal solutions may be regenerated in different generations. There is an option in the VMODEX user interface to show the generation numbers. By selecting that option, for each design point node in the DSE tree, a hexagon will be drawn at the upper left corner of the node. The number of the last/first (chosen by the designer) generation in which the corresponding design point was generated is written inside the hexagon.

Moreover, the designer is also able to select a specific design point and see all the generation numbers in which the design point was evaluated. Figure 3.11 shows the

---

**Figure 3.10:** Visualization of processing time of processor operations

**Figure 3.11:** Visualization of the generation numbers
visualization approach for showing the generation numbers. For each generation, a hexagon is drawn. The size of the hexagons increases from the first to the last generations. To save space, these pentagons are nested together. If within a generation the selected design point is found, then the color of the pentagon representing that generation is red. Otherwise, a gray pentagon is drawn. Figure 3.11(a) shows the situation that the corresponding design point is close to the optimum. Since it is regenerated in many search generations during the entire search. But in Figure 3.11(b) the corresponding design point is generated in only two generations. This indicates that the design point is far from the optimal solutions and after a few generations it is not regenerated any more.

In the evolutionary algorithms, we expect better solutions at the later generations, as they gradually converge to the optimal solutions during successive generations. So, the later generations are more important. In our proposed visualization, we emphasize the importance of later generations by increasing the size of their corresponding hexagons.

### Showing Exact/Normalized Objective Values
Instead of showing objectives with visual variables (color and size), VMODEX can also show the exact values of processing time and energy consumption. It also represents the normalized value of the objectives. We normalize objective values to make them scale independent. At the end of normalization, all design points get values in the range [0, 1] for their objective values. Before normalization, it is not possible to compare e.g. processing time and energy consumption with each other since they have different magnitudes. However, after normalization, comparing them is possible. Furthermore, for calculating the Euclidean distance between any two solutions (in the objective space), the normalized objective values are used (see Chapter 2, Section 2.1).

### 3.2.3 Edge Visualization

Edge visualization helps designers to navigate through the DSE tree and easily find more important parts. One feature of the design points is chosen as an importance factor and then the tree edges are visualized according to that factor, as follows:

- A minimum and maximum edge width is defined, and this range is linearly mapped against the range of importance factor values. For example, the following linear mapping can be used:

\[
\frac{\Delta IF}{\Delta Width} = \frac{IF_{max} - IF_{min}}{Width_{max} - Width_{min}} = \frac{IF(e) - IF_{min}}{Width(e) - Width_{min}}
\]

\[
Width(e) = Width_{min} + \frac{\Delta Width}{\Delta IF} \times (IF(e) - IF_{min})
\]

Where \(IF_{max}\) and \(IF_{min}\) are respectively the maximum and minimum values of the importance factor, and \(Width_{max}\) and \(Width_{min}\) are respectively the
3.2. MULTI-OBJECTIVE DSE VISUALIZATION

maximum and minimum widths that are defined. \( IF(e) \) denotes the importance factor of the edge \( e \) and \( Width(e) \) indicates the appropriate width for representing the edge \( e \). Wider edges lead toward more important subtrees. The effect is a bit like a network of roads, in which the more important roads are wider.

- A specific color with various saturations is chosen. Similar to the line width, a linear mapping is done between the maximum and minimum saturation and the importance factor values range. Darker edges represent more important parts and lighter edges show less important subtrees.

In VMODEX, two importance factors are defined: minimum Euclidian distance and last generation number. In the following we explain these factors.

**Minimum Euclidian Distance**  For each design point, the Euclidian distance (in the objective space) between a solution and the nearest global Pareto optimal point is calculated. A smaller distance indicates that the solution is closer to the optimal solutions and therefore is better. Thus, in the DSE tree, the edges in the path from the root to the global Pareto optimal points are the thickest and darkest since the distance is zero (see Figure 3.2). As the distance increases the edges become thinner and lighter. In this manner, just by looking at the DSE tree, the designer can easily determines which parts of the design space contain optimal and near optimal solutions and which parts contain solutions that are far away from the optimal solutions.

**Last Generation Number**  The number of the last generation in which a design point is evaluated can be considered as an importance factor. As the MOEA gradually converges to a set of Pareto optimal points, we expect better design points in the later generations. The edge visualization can show the progress of the searching algorithm in exploring and covering the design space during its generations. The edges with a higher generation number in their subtrees (i.e. the design points in their subtrees are evaluated in the later generations) are thicker and darker. As a result, the paths from the root to the last generated data nodes are the darkest and thickest paths.

As the importance factors are not applicable for edges that have no data nodes in their sub tree, these edges are shown by gray dashed lines.

3.2.4 Visualization of the Design Space Coverage

In VMODEX, we provide a technique to show how a heuristic searching algorithm walks through the design space and accesses new parts of the design space during its exploration process. To do this, a color-coding scheme is used to color the parameter nodes based on the generation number in which a design parameter is explored for the first time. Thus, the color of each parameter node represents the first generation that a design point, containing that parameter, is evaluated. On the other hands, for each parameter node, the number of the generation in which the first design point is
added to its subtree, is used for coloring the node. For visualizing the design space coverage, the green color with variations in lightness and saturation is used, such that the color of parameter nodes, which are searched in the earlier generations, is light green. The color of nodes becomes darker as they are investigated at the later generations. Therefore, the designer can easily see how a searching algorithm covers different parts of the design space during its successive iterations. In Section 3.3.1, an example of this visualization is shown for our case study results (Figure 3.15). In Figure 3.15 the color legend for visualizing the design space coverage is also shown.

### 3.2.5 Interactive Exploratory Techniques

User interaction plays an essential role in the effective visualization of data and information. It allows users to explore and work with the data actively to investigate the data from different perspectives. Interactive visualizations are very useful in analyzing data and can provide new insights that could not be obtained using static graphics or statistical methods. Interactivity means that users cannot only observe the visualization but moreover play with it. The visualization changes in response to the users actions and enable users to create customized views that are more interpretable and informative to be able to find hidden patterns in the data and complex relationships among variables. Colin Ware [75] notes,

"The best visualizations are not static images to be printed in books, but fluid, dynamic artifacts that respond to the need for a different view or more detailed information" (p. 385).

Shneiderman [76] identified two aspects of visualization technology that should be considered for visualizing information in the most comprehensive and efficient way. One focuses on mapping abstract information to a visual representation and the other provides interactions between the visualization system and the users for more effective data exploration and help users to deal with large volumes of information. A user cannot be expected to deal with a single image consisting of thousands of information elements. Therefore, we have provided a set of interaction techniques in VMODEX to help designers to handle large design spaces and to analyze the data and explore the search results from different perspectives and at multiple levels of abstraction in order to find out interesting and important features that may not be found just by looking at the DSE tree visualization.

### Zoom

Zooming and scrolling are traditional tools in visualization; they are quite indispensable when large trees are explored. In our visualization tool, besides normal zooming and scrolling, we provide two extra zooming features for augmenting the exploration process: *bird view* and *satellite view*. 
Bird view provides overview (context) and detail (focus) on the same screen by allowing users to view items at different scales. The aim of bird view is to enlarge items closer to the users point of interest while compress the objects that are farther away from the interested region. Bird view is a window moving with the mouse-pointer and works like a magnifier. That means, those parts of the main scene that are under the bird view window are shown in enlargement with a higher zoom factor. So, by simply hovering over the DSE tree with the mouse-pointer, it is possible to zoom in on an area of interest to see its details without losing the global context. Bird view is helpful when the tree is big and the user zooms out to see an overview of the entire tree in one scene and still wants to see the details of some specific nodes such as labels.

Satellite view, shown at the bottom of Figure 3.1, gives an overall, smaller scale view of the entire scene, which allows the user to navigate quickly across the view. It also enables the user to zoom in on certain parts of the scene to focus on certain nodes in the tree without losing track of the position in the entire scene.

**Hiding Nodes**

In VMODEX, two options are provided to reduce the number of visible nodes in the DSE tree in order to make its size smaller for better and more effective exploration. It should be mentioned that by hiding some nodes, we recalculate the location of visible nodes in the tree to optimize their fit on the screen. This means that, the empty spaces from the hidden nodes are occupied by the visible nodes. Note that the order of the visible nodes in the tree will not be changed. So, the structure of the tree remains the same.

**Hiding Sub Trees without Exploration Data** Since we use heuristic search techniques, some areas of the design space may not be visited by the searching algorithm (e.g., they are not interesting enough). So, we do not have any evaluated design points for those parts. In VMODEX it is possible to hide the sub trees of the nodes that have no evaluated data. This way, the designer can focus on the explored sub trees, which are more important. However, if the designer is interested to see the entire design space (both evaluated and not evaluated parts), there is another option in the tool that shows all parts of the design space. In this case, the color of parameter nodes that are not visited by the searching algorithm is white. So, the designer is able to recognize which parts of the design space are not searched during the exploration process.

**Hiding Uninteresting Sub Trees** If the designer is not interested in some parts of the tree, then he is able to hide them in order to make the tree smaller and pay more attention to other nodes. By double clicking on a node, its sub tree collapses (becomes invisible) and a blue triangle appears at the bottom of the node specifying that the children of the node are hidden. The size of the triangle represents the size of the sub tree. The bigger the triangle is, the more nodes in the sub tree exist. By double clicking again, the sub tree expands (becomes visible) and the blue triangle is removed. If the
designer is interested to know the exact number of nodes in a hidden sub tree, by clicking its corresponding triangle the amount of nodes is shown. In Figure 3.1, the blue triangle at the bottom of the cost node "190" indicates that its corresponding sub tree is hidden, which contains 600 nodes (the number written at the bottom of the triangle).

Filtering

The filtering option in VMODEX allows designers to filter out parts of the solutions, which are not required for further investigation, and view only preferred design points. Therefore, the designer can focus on the more interesting design points and find the similarities/dissimilarities between them. Various kinds of filtering are available based on different designer preferences: 1) primary objective values, 2) design parameters, 3) distance from the Pareto optimal set and 4) secondary objective values. The combinations of different filtering approaches are also possible. The filtering results can be shown in three ways: view all design points that fall within the filtering conditions or only see local Pareto optimal points or only global Pareto optimal points.

Filtering Based on the Primary Objective Values  Primary objectives are those objectives that participate in the optimization process and based on their values, the searching algorithm explores the design space. In some cases, the designer wants to consider only design points with some specific objective values (e.g. design points with the best processing time). The value of each objective is controlled by a range slider bar, in which the designer can set upper and lower limits on that objective. Design points with objective values inside the selected ranges are visible and the others become invisible. Therefore, the designer has the ability to view only design points with preferred objective values and find out the relationships between them.

Filtering Based on the Design Parameters  VMODEX allows designers to hide the parts of the design space that are not being considered for further analysis to pay more attention to the more interesting parts. For instance, the designer may not be interested in design points with more than four processors. Then it is possible to hide those subtrees in the DSE tree that contain more than four processors and focus only on those parts of the design space that contain design points with the preferred parameters.

Filtering Based on the Distance from the Global Pareto Optimal Set  In the multi-objective context, the goodness of a design point can be evaluated by its distance from the Pareto optimal set. The smaller the distance the better the solution. This distance measure is the Euclidean distance (in the objective space) between the design point and the nearest member of Pareto optimal set. In VMODEX, we provide a filtering option based on this distance measure. The designer is able to define a distance threshold and then filters the design points to see solutions that have lower/higher distance
values than the threshold. Furthermore, as in VMODEX both the design parameters and objectives are shown in a single view, the designer can easily understand which parts of the design space contains solutions that are close to/far away from the Pareto optimal set.

Filtering Based on the Mapping Decision For better analyzing the effect of different mappings on the objective values, we provide the following filtering option. The designer can specify some constraints on the mapping and then filters the design points to see only those solutions that not violate the specified mapping constraints. After that, it is possible to investigate the values of different objectives for the design points, which satisfy the mapping restrictions and find out the influences of mapping decisions on the objective values. There are three ways for defining a mapping constraint:

1. Some specific tasks (determined by the designer) are mapped on the same processor, without specifying the type of the processor. In this case, we can study the effectiveness of the internal communications between them as well as comparing the results of using different processor types for executing those tasks.

2. Some specific tasks are not mapped on the same processor. So, we can investigate the effect of using (shared) memories for their communications.

3. For one or more tasks a specific processor type is determined to be mapped to. In this case, we can investigate the effect of other tasks mappings on the objective values.

For instance, the designer may want to know what objective values can be obtained with the following mapping constraint: tasks 1 and 2 are mapped on the same processor, task 4 is executed by processor ASIP-1 and task 5 and 6 are mapped on different processors.

Filtering Based on the Secondary Objective Values Secondary objectives are not considered as the optimization goals, but they provide more detailed information about the evaluated design points. Thus, we also provide filtering options based on the secondary objectives. For instance, it is possible to filter design points based on their average utilizations or specify a minimum desired utilization for each architectural component.

Step by Step Animation

Since exploring the design space is an iterative process, it is important to know how a searching algorithm walks through the design space and trace its progression in finding new design points and covering the design space during successive iterations. In VMODEX, we provide an interactive user interface that allows designers to follow
the evolutionary exploration process during numerous generations. This interface is called step-by-step animation. The designer chooses an arbitrary generation number as a start point. At this step, the DSE tree shows only those design points that are found in the specified generation number or in earlier generations. Then, the designer can move forward or backward in generations to see the dynamic exploration.

When moving through the generations (i.e., replaying parts of or even the entire search process), it is important to know in each generation which new design points are found (added to the DSE tree). To show this, the design points generated in the current generation are blinking. If a parameter node, which had no data node in the previous step, receives its first data node in the current generation, it starts blinking as well. Therefore, the designer can easily notice that some new parts of the design space are visited for the first time by the algorithm in the current generation. For the hidden sub trees, in case of any data node addition, the blue triangle starts blinking and if the user is interested in viewing that sub tree, it can be expanded as well.

**Details on Demand**

Because of limited screen space as well as high data complexity, showing every detail for all design points is impractical. Therefore, generally, in proposing a visualization method, only the most important data features are shown directly in the visualization and detailed information for interesting parts is shown just after the user requests them. In VMODEX, a variety of detailed information are provided, enabling designers to interpret the evaluated design points from different aspects and gaining additional insight into the underlying information. This detailed information has been described earlier in this section and are called the secondary objectives. By right clicking on an interesting design point, a pop-up menu appears that shows the available detail options. The user can simply select the desired detailed information from the list and see it.

**Adjusting DSE Tree Appearance**

In VMODEX, there are some options for modifying the way that the DSE tree looks, based on alternatives for showing/hiding sub trees, coloring the parameter nodes and tree edges.

**Showing/Hiding Sub trees** In VMODEX, the designer can choose to see the entire design space or only those parts of the design space that are visited by the searching algorithm. Furthermore, it is possible to hide the sub tree of a (uninteresting) node by double clicking on it.

**Coloring Parameter Nodes** As we described in Section 3.2.2, in the DSE tree, the color of design point nodes represent the processing time and the color of the
third dimension shows the energy consumption. Parameter nodes, however, do not represent single design points and therefore do not have the direct notion of processing time or energy consumption. For this reason, we provide some options to color the parameter nodes. The designer is able to choose the average, minimum, or maximum of either processing time or energy consumption of the design points in their sub trees as a method for coloring the parameter nodes. The color of parameter nodes that have no data node (i.e., do not have any DSE data) is white.

Another option for coloring parameter nodes is based on the generation number in which the searching algorithm could get access to that part of the design space for the first time. (See Section 3.2.4)

Coloring and Thickness Regulation of Tree Edges As we explained in Section 3.2.3, tree edges can be visualized according to two importance factors: minimum Euclidian distance and last generation number. In the VMODEX user interface, there is an option for selecting one of these factors for visualizing the edges.

3.3 A Case Study

In this section, we present a real application case study to illustrate the benefits of using visualization in the design space exploration process. In this case study, we map a Motion-JPEG (M-JPEG) encoder to a heterogeneous multi-processor system-on-chip platform architecture consisting of a general-purpose microProcessor (mP), a micro-Controller (mC), an Application Specific Instruction Processor (ASIP) and two Application Specific Integrated Circuits (ASIC-DCT and ASIC-VLE). For communication, the platform architecture contains two dedicated point-to-point FIFOs (between mP and ASIP) and two shared memories; one Static RAM (SRAM) and one Dynamic RAM (DRAM), each one is accessible through a common bus. Note that the evaluated design instances only use a subset of the platform resources, based on the mapping of application tasks and communication channels onto the platform resources. Figure 3.12 represents our MP-SoC architecture model and Figure 3.13 shows the M-JPEG encoder application.

In our case study, an mC or an mP processor can execute all the different processes in the M-JPEG application while an ASIP can execute only three processes, namely: DCT, Q and RGB2YUV. The ASIC-DCT is designed for executing the DCT process and ASIC-VLE is dedicated for executing the VLE process.

The design space that we consider in this study has four parameters: number of processors, processor type, number of memories and memory type. Each architecture platform instance is indicated by a unique combination of these parameters. Those combinations that lead to the platform instances that are capable of executing the application are denoted as feasible combinations. Using the NSGA-II multi-objective evolutionary optimizer [19], we intend to find a set of optimal design points (in terms of alternative architectural solutions and mappings) under three criteria: processing
time, energy consumption and architecture cost. For this study, we run the MOEA for 100 generations with 100 individuals per population. Therefore, 10000 design points are searched by the MOEA. In this experiment, we have used the uniform crossover with probability 0.9. The mixing ratio between two parents is 0.5 (i.e. the offspring has approximately half of the bits from first parent and the other half from second parent). The mutation probability² is 0.23 and the bit mutation probability³ is 0.01. For evaluating design points in terms of aforementioned criteria, we used the Sesame system-level simulator. In our case study, the processing time varied by factor of 6.4, energy consumption by factor of 4.45 and architecture cost by factor of 10.25. These large variations indicate the need for automated exploration and optimization.

VMODEX allows designers to look at the evaluated design points and explored design space from different perspectives and analyze the search results at multiple levels of abstraction. In the following subsections, we analyze the M-JPEG case study with respect to the following issues: 1) design space coverage, 2) the characteristics of the global Pareto optimal points, 3) investigating the absence of ASIC-VLE in the Pareto optimal set, and 4) studying the effects of executing the DCT process by different processor types on design criteria.

²I.e., the likelihood of mutating a particular individual.
³I.e., the likelihood of mutating each bit of an individual in mutation.
3.3.1 Design Space Coverage

Since we use an evolutionary algorithm (instead of exhaustive search) for exploring the design space, the searching algorithm may not visit some parts of the design space. Therefore, there is no evaluated data for those parts. However, it is essential that a searching algorithm achieves a broad coverage of the design space. This means that the algorithm is able to find solutions in as many as possible different regions of the decision space, even if those solutions are not Pareto-optimal. In this case, the exposed solutions represent the variety of possible designs in the decision space and enable the designer to do a more comprehensive study on the relationship between design parameters and their effects on the design criteria. Furthermore, a higher diversity and coverage in the searching algorithm also means that there is a higher chance of finding the global optimal solutions rather than the local optimum ones (the algorithm does not get trapped in local optimum points in the design space). The spread of design points in the design space can help the algorithm to escape from such local optima.

Using VMODEX, the designers can easily see the parts of the design space that are covered/not covered by the MOEA. This tool clearly shows which parts of the design space contain evaluated solutions and which parts are not searched at all (no design point is evaluated there). As we have described before, parameter nodes with a white color and dashed line have no data. In our case study there are 99 possible combinations of design parameters. By visualizing the exploration results, we could find out that more than 90% of possible platform instances are searched (91 out of 99). For platforms with two and three processors, all the possible combinations are searched. Further, only a few combinations of platforms with four and five processors have not been visited by the MOEA. Those parts of the design space that are not investigated during the exploration process are shown in Figure 3.14. In this figure, minimum processing time is used for coloring the parameter nodes. The white nodes represent the combinations that are not searched by the MOEA. For instance, there is no evaluated design point that consists of four processors of the types ASIC-DCT, ASIC-VLE, ASIP and mP and two memories of which one is DFIFO-1 and another one is DRAM (this platform instance is indicated by subtree ”A” in Figure 3.14). However, the color of the node labeled with ”2” in this subtree (at the number of memories level) indicates that the other combinations of two memories such as one DRAM and one SRAM are searched.

Next, we are going to study how our MOEA walked through the design space and found new architecture platform instances during its search iterations. We used the ”visualization of design space coverage” option in VMODEX to understand the searching behavior of the MOEA in the design space. By selecting this option, parameter nodes are colored based on the generation number in which the searching algorithm gained access to that part of the design space for the first time. Therefore, the color of the parameter nodes, which are searched in the earlier generations, is lighter and those parameters that are visited in the later generations are darker. In our case study, we found that most of the architecture platform instances (possible combinations of design parameters) are searched in the 15 first generations. Actually, in
3.3. A CASE STUDY

Figure 3.14: Platform instances that are not visited by the MOEA

our initial population, which is generated randomly, 58 unique instances are visited and then until the 15th generation totally 86 (out of 99) possible combinations are searched. In the next generations, the algorithm could find 4 more instances and after the 65th generation it could not find any new combination of design parameters. Figure 3.15 shows the visualization of design space coverage for our case study. Since the size of the DSE tree representing the entire design space is large and it is not possible to include it here, this figure shows only four processors architecture instances. Furthermore, the four last combinations that are found by the algorithm are in this part of the design space. So, we can easily see which architecture instances are searched in the later generations.

As can be seen in this figure the color of most parameter nodes is light green, which indicates that those parts of the design space are searched in the beginning generations. Only four nodes (denoted by A-D) are colored with darker green, which points out that those architecture instances are visited in the later generations. The number written at the bottom of these nodes shows the number of the first generation during which the algorithm has investigated them.

As we described before, the size of the blue triangle at the bottom of each node represents the number of design points in its subtree. From Figure 3.15 we can see that the platform instances containing four processors of the types ASIC-DCT, ASIP, mC and mP (subtree indicated by “S”) are searched more often by our MOEA since nodes in this subtree have the biggest triangles.

In Figure 3.15 the minimum Euclidian distance is used as an importance factor for edge visualization. Therefore, the edges in the path from the root to the global Pareto optimal points are the thickest and darkest (black color) since the distance is zero. From Figure 3.15 we can see that among four processors platform instances only one of them leads to the global optimal design points, which is indicated by subtree S. This explains why the searching algorithm evaluated more design points in subtree S. In this part of the design space it has found design points that are closer to the opti-
3.3. A Case Study

Color legend for visualizing the design space coverage

Figure 3.15: Visualization of design space coverage for our case study
### 3.3. A Case Study

#### 3.3.2 The Characteristics of the Global Pareto Optimal Points

Figure 3.16 shows the global Pareto optimal points found by our MOEA. By looking at the picture, the designer can immediately recognize the characteristics of the Pareto optimal points, which are the best design points with respect to the design criteria. As in the DSE tree both design parameters and objective values are shown in a single view, the designer can easily find out which combinations of architectural components yield optimum design points and what the trade off is between objective values. As can be seen in Figure 3.16, in our case study, fifteen unique combinations of architectural components lead to the seventeen Pareto optimal points. The variety of architecture cost in the Pareto optimal set is quite large. The cost is varying from the cheapest one to almost the most expensive one that have been encountered during the whole search (approximately $10 \times (195/20)$). For the processing time, except the design point with a single processor architecture (labeled by “P1” in Figure 3.16) all the other points have relatively good processing time. The normalized value of the processing time (excluding P1) is in the interval $[0, 0.21]$. The energy consumption of all the found Pareto optimal points is relatively low. Its normalized value is in the range $[0, 0.16]$.

Now we are going to analyze the discovered Pareto optimal points in terms of design parameters. From Figure 3.16 we can see that, in our case study, there is no Pareto
3.3. A Case Study

optimal point with five processors. This means that with less processors (which is cheaper) we can get the same or better processing time and energy consumption. Therefore, using five processors is not appropriate for this application. Another interesting feature is that most of the obtained Pareto optimal points (10 out of 17) contain two processors in their underlying architecture. Thus, maybe two processors architectures are more suitable for executing our application. It can also be seen that there is no Pareto optimal point with four memories. So, the communications between processors can be done efficiently with less memories. Furthermore, we can see that the processor ASIC-VLE, which is dedicated for executing the VLE process, is never used in the Pareto optimal set. In the next subsection we are going to find out the reason of this remarkable property.

3.3.3 Investigating the Absence of the ASIC-VLE in the Pareto Optimal Set

As we can see in Figure 3.16 there is no Pareto optimal point that contains ASIC-VLE in its underlying architecture. By looking at the DSE tree representing the entire design space, we can find that there are several evaluated design points that use ASIC-VLE. Thus, our MOEA could access the parts of the design space that contain ASIC-VLE. However, the quality of design points in these parts with respect to the design criteria is not optimal and they are dominated by the solutions in the other parts. Figure 3.17 shows all the possible combinations of processor types. To make the picture fit here, different alternatives with respect to the number and types of memories are not shown. Thus, only the first two levels of the parameter segment are shown (i.e. the number of processors and processor types). In Figure 3.17(a) the minimum processing time and in Figure 3.17(b) the minimum energy consumption is used for coloring the parameter nodes. As can be seen in these two figures, in all of the platform instances containing ASIC-VLE, both the processing time and energy consumption is quite high.

We use filtering options provided in VMODEX to realize why both the processing time and energy consumption are significantly increased when using an ASIC-VLE. First, we filter design points based on their primary objective values. We were interested to see all design points which are relatively good in both processing time and energy consumption (their normalized value is in the range [0, 0.2] for both objectives). There was no restriction in the architecture cost objective. We found out that in all of them three processes V-Out, VLE and Control are mapped to the same processor. We did similar filtering to see those design points that are extremely poor in both (their normalized value is in the range [0.8, 1.0] for both objectives). Then, we could recognize that in all of them, the three aforementioned processes are mapped to different processors. Thus, we can conclude that the communications between these processes are much more intensive than the communications between other processes. The reason for this is that over these channels tables for Huffman encoding are being transmitted (see [77]). However, for the other processes, the communications are all the same as they communicate pixel blocks, which are small units of data. Therefore, using
3.3. A CASE STUDY

(a) Minimum processing time is used for coloring parameter nodes. For all platform instances containing ASIC-VLE (indicated by $A_1$-$A_{12}$), the processing time is quite high (the color of node is red).

(b) Minimum energy consumption is used for coloring parameter nodes. For all platform instances containing ASIC-VLE (indicated by $A_1$-$A_{12}$), the energy consumption is quite high (the color of node is dark violet).

*Figure 3.17: Possible combinations of processor types*
shared memories for these communications causes large overheads that can actually make the system slower. However, if these communications are done internally (mapping in the same processor) the mapping is much more efficient with respect to both processing time and energy consumption. As ASIC-VLE is dedicated for executing the VLE process, by using it, the processes V-Out and Control have to be mapped on the other processors and therefore their communications will be done externally using shared memories, which is not efficient. Thus, even though using an ASIC implementation for the VLE process is computationally efficient, because of its intensive communication, it is not an effective solution. In Figure 3.18 we illustrate this conclusion with an example. This figure represents the mapping decisions of two design points in which they have exactly the same mapping for all the processes except for VLE. In Figure 3.18(a) the VLE process is mapped to the mP (the same processor as V-Out and Control) and in Figure 3.18(b) it is mapped to the ASIC-VLE. From the visualization of their objective values we can see that they are significantly different in both processing time and energy consumption. The legend for shape and color of processor and memory types are shown in Figure 3.18(c).

**Figure 3.18:** Mapping decisions (with and without ASIC-VLE)
3.3.4 Studying the Effects of Executing the DCT Process by Different Processor Types on Design Criteria

In the M-JPEG application, the DCT is the most computationally intensive process and therefore it is essential to be mapped on the processor that is optimal for executing it. We are going to investigate how the values of processing time and energy consumption change by running the DCT on different types of processors. To do that, we use mapping filtering. From the previous discussion we understand that if three processes V-Out, VLE and Control are not mapped on the same processor, regardless of the mapping of the other processes, the processing time and energy consumption are quite high. So, in this study, we assume that these three processes are mapped on the same processor and then we examine the influences of different mappings for DCT on the objective values. Using mapping filtering, we could find out that if DCT is mapped to ASIC-DCT or ASIP both the processing time and energy consumption is relatively good. But, by executing the DCT on the mC or mP, we cannot get any satisfactory solution. To illustrate this conclusion, an example is shown in Figure 3.19. This figure represents the mapping decisions of four design points in which the mappings of all processes excluding DCT are the same. The variation on mapping the DCT are shown in Figure 3.19(a) to Figure 3.19(d). From the visualization of the objective values we can see the effect of each mapping on processing time and energy consumption.

The analysis we performed in this section would have been very cumbersome and time consuming to do by only looking at the raw data or by using traditional 2D/3D graphs. Several traditional graphs are needed in order to interpret the data like we did. However, using VMODEX, a single visualization view of the design space enables very powerful and rapid analysis of the DSE data.

3.4 Comparing Subspaces

As we described in Section 3.2.1 in VMODEX, the design space is modeled as a tree and this kind of modeling causes the design space to be divided in several subspaces. Each subspace represents a unique instance of the architecture platform. On the other hand, solutions inside a subspace have exactly the same architecture components (have the same parents at the parameter levels) but the way that the application is mapped onto those components is different. We have provided additional functionality in VMODEX to allow designers to evaluate and compare the properties of interesting subspaces from various perspectives.

Figure 3.20 represents the design space of our case study, which is visualized by VMODEX. This figure shows only those parts of the design space that are compared with each other in this section. To make the picture smaller to be able to put it here, only global and local Pareto points are shown and non-Pareto points are not displayed. In the DSE tree shown in Figure 3.20 there are four subspaces, which are indicated by dashed contour lines and are labeled from $S_1$ to $S_4$. For instance, the subspace $S_1$ consists of one ASIP, one mC, one DRAM and one SRAM. Thus, all design points
3.4. Comparing Subspaces

Figure 3.19: Illustration of the effects of executing DCT process by different processor types on design criteria
in this subspace utilize these architecture components (same resource allocation) but their task and communication bindings are different. In each subspace, the Pareto optimal solutions found by a MOEA are called local Pareto optimal solutions, which are optimal with respect to a specific architectural instance. However, in the entire design space, a design point might exist which dominates the local Pareto point. In the following subsections, we explain the techniques VMODEX provides for analyzing and comparing the properties of discovered design points in different subspaces. We use these techniques for comparing the properties of the four local Pareto optimal sets, which are labeled by $LPS_1$ to $LPS_4$ in Figure 3.20.

The concept of local Pareto solutions and proposing some methodologies for evaluating and comparing different local Pareto optimal sets (like we explain here) is a new point of view in the multi-objective DSE process and has not been considered before. Such comparison is, however, essential to the designer as it provides insight into the landscape of the design space and can help him to comprehensively understand the properties of the discovered design points in different subspaces of the explored design space. Using VMODEX the designer is able to select the interesting subspaces and compare them with respect to the different aspects, which are explained in the following subsections.

Figure 3.20: Subspaces of the design space
3.4. Comparing Subspaces

3.4.1 Distance from the Global Pareto Optimal Solutions

A subspace containing local Pareto optimal solutions, which are closer to the global Pareto optimal set, is more preferable. The distance can be measured in two ways: 1) the number of solutions in a local Pareto optimal set which are also in the global Pareto optimal set, and 2) the average of Euclidian distances (in the objective space) between the solutions in a local Pareto optimal set and the nearest member of the global Pareto optimal set.

In the DSE tree, both distance measures can simply be evaluated. Just by looking at the tree, one can easily recognize which solutions of a local Pareto optimal set are in the global Pareto optimal set as well. For those solutions, their parents (in the tree) are Pareto optimal nodes; otherwise they become children of a relation node. For example, in Figure 3.20, two solutions (out of three) in the set \( LPS_2 \) are globally Pareto optimal. However, in the other sets, none of the solutions are in the global Pareto optimal set (their parents are relation nodes).

The second distance measure can be seen directly in the DSE tree as well. The color and thickness of edges show the distance from the nearest global Pareto optimal solutions. The edges in the path from the root to the main Pareto optimal solutions are the thickest and darkest since the distance is zero. As the distance increases the edges become thinner and lighter. The value of the average distance between solutions in a local Pareto optimal set and global Pareto optimal set is shown at the bottom of the local Pareto set (denoted by \( \bar{d} \)). If the designer is interested to know the exact value of the distance measure for a particular solution, then the distance value is shown by clicking the corresponding edge. For example, in Figure 3.20, the average distance for \( LPS_2 \) is 0.01 which means that solutions in this set are quite close to the Pareto optimal set. However, the solutions in set \( LPS_5 \) are relatively far from the global Pareto optimal set since their edges are thin and light and the average distance is 0.38.

3.4.2 Coverage of Local Pareto Sets

Zitzler and Thiele [78] introduced the Coverage (C) metric, which directly compares two Pareto optimal sets with each other. The metric \( C(P_1, P_2) \) calculates the proportion of solutions in Pareto optimal set \( P_2 \) which are weakly dominated at least by one solution in \( P_1 \):

\[
C(P_1, P_2) = \frac{|\{p' \in P_2 | \exists p \in P_1 : p \leq p' \}|}{|P_2|}
\]

(3.11)

Where \( \leq \) is the weakly dominance relationship. For two solutions \( p \) and \( p' \) it is said that \( p \) weakly dominates \( p' \), if \( p \) is not worse than \( p' \) in all objectives. In fact, the function \( C \) maps the ordered pair \( (P_1, P_2) \) to the interval \([0, 1]\). The value \( C(P_1, P_2) = 1 \) means that all members of \( P_2 \) are weakly dominated by \( P_1 \) and \( C(P_1, P_2) = 0 \) represents the situation where none of the solutions in \( P_2 \) are weakly dominated by \( P_1 \). Note that for fully understanding the dominance relations between two Pareto optimal sets, both directions \( C(P_1, P_2) \) and \( C(P_2, P_1) \) have to be considered, since
3.4. Comparing Subspaces

$C(P_1, P_2)$ is not necessarily equal to $1 - C(P_2, P_1)$. The $C$ metric compares only two sets with each other. Thus, for comparing more than two sets, we propose a new metric called Total Coverage ($TC$), as follows:

$$TC(P_i) = \sum_{j=1, j \neq i}^{n} C(P_i, P_j) - C(P_j, P_i)$$  \hspace{1cm} (3.12)

Where $n$ is the number of comparing sets. $TC > 0$ means that the dominating rate is higher than the dominated rate and the $TC < 0$ implies that the solutions are more dominated by the other sets than they dominate solutions in the other sets. Therefore, a set with a bigger $TC$ value is better. In VMODEX, the designer is able to select the interesting local Pareto sets and compare them using the $TC$ metric. For better understanding the dominance relations between sets, we visualize this metric. A directed weighted graph is used for visualizing the $TC$ metric. Each comparing set is shown as a node in the $TC$ graph. For each two sets $P_1$ and $P_2$, if $C(P_1, P_2) \neq 0$ then an edge is drawn from $P_1$ to $P_2$, of which the weight is equal to the $C$ value. In the case of $C(P_1, P_2) = 0$, there is no edge from $P_1$ to $P_2$. Due to the summation property, we can break up a summation across a sum or difference. Therefore, we can rewrite the $TC$ formula as follows:

$$TC(P_i) = \sum_{j=1, j \neq i}^{n} C(P_i, P_j) - \sum_{j=1, j \neq i}^{n} C(P_j, P_i)$$  \hspace{1cm} (3.13)

According to the $TC$ graph, the first sum in Equation 3.13 is equal to the sum of the weights of outgoing edges and the second sum is equivalent to the sum of the weights of incoming edges. Thus, for each node in the $TC$ graph, the value of $TC$ metric is calculated by the sum of the weights of outgoing edges minus the sum of the weights of incoming edges:

$$TC(P_i) = \sum (Outgoing Edges) - \sum (Incoming Edges)$$  \hspace{1cm} (3.14)

The size of the nodes in the graph indicates the $TC$ value. Therefore, nodes with higher $TC$ values are bigger. Because the $TC$ value greater than zero (means more dominating) or less than zero (means more dominated) have completely opposite meaning, we demonstrate this property in the nodes color. Nodes with $TC > 0$ are shown in blue while nodes with $TC < 0$ are shown in red. Figure 3.21 shows the visualization of the $TC$ metric for sets $LPS_1$ to $LPS_5$ shown in Figure 3.20. As can be seen in this figure, all solutions in $LPS_5$ are dominated by all the other comparing sets (there are four incoming edges from $LPS_1$ to $LPS_4$ with the weight $4/4 = 1$), while there is no solution in $LPS_2$ that dominates a solution in the other sets ($LPS_5$ does not have any outgoing edge). Thus the $TC$ value for $LPS_5$ is less than zero ($TC = -4$) and this node is shown by a red color. Furthermore, we can understand that solutions in $LPS_1$ dominate all the solutions in set $LPS_3$, while no solutions in $LPS_1$ are dominated by any other sets, since there is no incoming edge. Moreover, we can see that half of the solutions is $LPS_4$ are dominated by solutions in $LPS_2$. As
a result, \( LPS_1 \) has the highest \( TC \) value (biggest node in the graph) and thus is the best local Pareto optimal set among the comparing sets with respect to the \( TC \) metric. 

If the designer is interested to know more about the dominance relation between each two sets, such as which solutions in one set dominate which solutions in the other set, it is possible to select those sets to see more details. To this end, we visualize the dominance relation between two sets as follows. Solutions in both sets \( P_1 \) and \( P_2 \) are shown in two different rows. If a solution in \( P_1 \) dominates a solution in \( P_2 \), an arrow is drawn between them coming out from the solution in \( P_1 \) to the solution in \( P_2 \). Furthermore, a cross is displayed at the dominated solution in \( P_2 \) to show that this solution is dominated by another one. Figure 3.22 shows the visualization of the dominance relation between two local Pareto sets \( LPS_1 \) and \( LPS_3 \). From this figure, we can understand that all solutions in set \( LPS_1 \) dominate all the solutions in \( LPS_3 \) and therefore \( LPS_1 \) is absolutely better than \( LPS_3 \). The architectural components of each local Pareto set are shown on the left side of each set. As can be seen in Figure 3.22, both local Pareto sets have the same components except that in \( LPS_3 \) the ASIP is replaced by mP. Using an mP instead of an ASIP is not beneficial since it increases the cost but does not yield a better processing time nor a better energy consumption.

3.4.3 Size of the Dominated Region

In [79][80], the Hypervolume metric is proposed, which measures how much of the objective space is dominated by a given non-dominated set. A set with a larger hypervolume is desirable. We use this metric to compare the local Pareto optimal sets in different subspaces of the design space. Moreover, we visualize this metric to illustrate which area of the objective space is dominated by solutions in a local Pareto set. Figure 3.23 represents the visual from of the hypervolume metric for two local Pareto sets \( LPS_1 \) and \( LPS_5 \) shown in Figure 3.20. Each side of the cube shows one objective. The colored region indicates the dominated area in the objective space. Each point in this region is dominated at least by one member of Pareto optimal set. For better vision, each side of the cube is colored with the corresponding color scheme in the DSE tree. Black lines in the colored region denote the solutions in a local Pareto...
optimal set. The normalized values of processing time ($PT$) and energy consumption ($EC$) for each solution, is written at the bottom of the cube. As can be seen in Figure 3.23 in set $LPS_1$ almost half of the objective space is dominated while only a small portion of the objective space is covered by the solutions in the $LPS_5$. Using the Hypervolume visualization, it is possible to see the dominated parts of each objective separately. For instance, in Figure 3.23 the energy consumption and processing time surfaces are almost completely covered for $LPS_1$ while only half of the cost surface is dominated. We also propose another approach for visualizing the Hypervolume metric, which is extendable to be used in problems with more than three objectives. This visualization method is explained in Chapter 4.

### 3.4.4 Sensitivity of Subspaces to Different Mappings on Design Criteria

In studying the properties of a certain subspace (i.e., specific platform architecture instance), it is a worthwhile issue to investigate how the objective values are being changed with modifying the mapping decisions. In this paper, we propose a visualization method for exploring the sensitivity of each subspace to different mappings on the objective values. To do this, the frequency distribution of objective values in each subspace is visualized to see the range as well as the frequency of objective values that can be achieved with a specific architecture instance by using different mappings. Since all solutions in a subspace have the same architecture cost, the frequency distribution is not applicable for this objective and should only be considered for the other two objectives: processing time and energy consumption.

For each objective, a horizontal axis from 0 to 1 is drawn and colored like the color-coding technique used for showing that objective in the DSE tree. For example, in our case, colors from yellow to red are used for representing the processing time and therefore, this color scheme is used for coloring the corresponding axis in the frequency distribution visualization. The height of each color bar in the (color-coded) objective axis indicates the number of design points with the objective value inside that range. Figure 3.24 shows the visualization of the frequency distribution for three subspaces. Figure 3.24(a) shows the frequency distribution of solutions in a subspace that contains an ASIP, mP, DFIFO-2 and SRAM. This subspace is denoted as $S_2$ in Figure 3.20. As can be seen in this figure, for all design points, both the processing time and energy consumption is relatively good (less than 0.5). Therefore, for this particular architecture, with different mappings we can get approximately good design points. So if the designer is looking for a system that can flexibly deal with different mappings, this architecture is a good solution. The subspace in Figure 3.24(b) consists of an ASIP, mC, mP, DFIFO-2 and SRAM, which is denoted as $S_4$ in Figure 3.20. In this subspace, both the processing time and energy consumption are varying from the best to almost the worst. However, the processing time and energy consumption of most design points are approximately good. Therefore, if the designer is interested in this platform architecture instance, he should take care about the mapping because a wrong mapping decision can make the difference between the best or the worst design
3.4. **Comparing Subspaces**

LP₁: PT=0.11, EC=0.02
LP₂: PT=0.13, EC=0.01
LP₃: PT=0.14, EC=0.01
Cost=0.54

\[ HV(S₁) = 0.43 \]

LP₁: PT=0.41, EC=0.79
LP₂: PT=0.41, EC=0.76
LP₃: PT=0.42, EC=0.74
Cost=0.89

\[ HV(S₃) = 0.017 \]

**Figure 3.23:** Visualization of the Hypervolume metric

(a) Subspace consisting of: ASICP, mP, DFIFO-2, SRAM

(b) Subspace consisting of: ASICP, mC, mP, DFIFO-2, SRAM

(c) Subspace consisting of: ASIC-DCT, ASIC-VLE, ASICP, mP, DFIFO-2, SRAM, DRAM

**Figure 3.24:** Visualization of the frequency distribution of objective values for three subspaces
3.5 Conclusion

In this chapter, we introduced our proposed visualization techniques, which are specially designed for understanding the multi-objective DSE process of embedded systems that are based on heterogeneous multi-processor system-on-chip architectures. All the developed visualization techniques are incorporated into a tool, called VMODEX. Our visualization tool provides insight into the search process of heuristic searching algorithms that are typically used in the DSE process. It helps designers to understand how such algorithms explore the design space during their iterations and converge towards the optimal design points. It should be mentioned that although in this thesis we choose the Sesame simulator for evaluating design points and MOEAs for searching the design space, our visualization tool is not restricted to neither evaluating approach nor searching mechanism. It can easily be used for different types of evaluating methods as well as searching strategies.

The tree model that we propose for visualizing the design space enables us to visualize multivariate data. There is no limitation on the number of neither design parameters nor criteria. Furthermore, in our DSE tree model, the concepts of subspaces and local Pareto points are proposed, which are new points of view in the multi-objective DSE process and has not been considered before.

VMODEX also enables very powerful and rapid analysis of DSE results. Designers can look at the data from different perspectives and at multiple levels of abstraction. Several interactive capabilities are provided, which allow designers to play with data and find out some interesting and important features that may not be found just by looking at the static visualization.

To illustrate the benefits of using our visualization techniques in the DSE process, we performed an actual case study on Motion-HPEG encoder application. Then, we used VMODEX to interpret and analyze the results. In this chapter, We explained some new and unexpected insights that we could gain from the tool that is not possible to find out those insights from the raw data.

To summarize the different visualization approaches that we proposed in this chapter, Figure 3.25 shows all our proposed visualization methods together with the graphical
3.5. CONCLUSION

Figure 3.25: summarization of all our proposed visualization approaches together with their relations with each other.

tricks that we used for visualizing them. Furthermore, in this figure, we clarify how these visualizations are related to each other.

In the following, we summarize the benefits of using VMODEX in multi-objective design space exploration process:

- It models the design space as a tree in which both the design parameters and objectives are shown in a single view. Therefore, it is easy to understand where the optimum design points are located and what objectives they have.
- There is no limitation on the number of design variables since each parameter is located at one level of the tree.
- It can easily be extended to show more than three objectives. Since each ob-
3.5. Conclusion

Objective is assigned to one attribute of the tree nodes and nodes have various attributes such as shape, orientation, size, color, transparency, texture, border, etc.

- Several techniques are provided in VMODEX to be able to handle large design spaces.

- Various user-interactions are developed for more effective data exploration and enabling to find hidden patterns in the data and complex relationships among variables.

- Different properties of evaluated design points can be shown, allowing designers to interpret the data from different aspects and gaining additional insight into the underlying information.

- By coloring the parameter nodes, it is possible to do some statistical analysis. The designer can compare different architectures (in terms of number and type of the processors and memories) according to the minimum, maximum or average of each of the design criteria.

- Edge visualization helps designer to easily navigate through the design space and find more important parts.

- The concept of local Pareto point is proposed and in the DSE tree there is an explicit level called local Pareto level, which shows the optimal design points with respect to specific architecture instances. Therefore, the designer can easily compare the best design points of different architecture instances with each other.

- In VMODEX, each unique instance of the architecture platform is considered as a subspace of the design space. Various methods are provided for evaluating and comparing the properties of design points in different subspaces of design space.

- It helps designers to understand the search behavior in heuristic based design space exploration such as:
  - Which parts of the design space are not searched at all (no design point is evaluated there). As we mentioned before, nodes with a white color and dashed line have no data.
  - Which parts of the design space are searched more often by the searching algorithm (more design points are evaluated there). In these areas, the tree provides more nodes so the sub trees of the corresponding nodes are bigger.
  - The step by step animation option in VMODEX allows us to trace the progression of the searching algorithm in finding new design points and covering the design space during its consecutive iterations.