8.1 Design Space, Pareto Points

The task graph in Figure 1 shows a system specification with four tasks, T1 . . . T4. The tasks can be executed on different components. Table 1 displays the execution times for the tasks on the different components as well as the component cost. For example, the MIPS processor costs 200 units and can run tasks T1 in 5 ms and task T4 in 2 ms. The numbers in Table 1 show also the number of available components of each type (MIPS, DSP, FPGA and ASIC). All components execute tasks sequentially – at any given time a component executes at most one task. Task execution is non-preemptive – once a task is started, it runs to completion.

![Figure 1: Task graph](image)

(a) Construct the design space by listing all possible design points. A design point consists of an allocation (selection of components), a binding (assignment of tasks to selected components) and a schedule (execution order for the tasks). Determine the total cost and execution time for each design point.

### Solution:

See Figure 2 and Table 2. If T2 and T3 are bound to the same resource, then the execution of T2 takes place prior to T3. If T2 and T3 are bound to different resources, then T2 and T3 execute in parallel. It is valid to consider 4 additional design points where execution of T3 takes place before execution of T2.

<table>
<thead>
<tr>
<th>component</th>
<th>number</th>
<th>cost</th>
<th>execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>T1</td>
</tr>
<tr>
<td>MIPS</td>
<td>1</td>
<td>200,–</td>
<td>5 ms</td>
</tr>
<tr>
<td>DSP</td>
<td>1</td>
<td>120,–</td>
<td>—</td>
</tr>
<tr>
<td>FPGA</td>
<td>1</td>
<td>240,–</td>
<td>—</td>
</tr>
<tr>
<td>ASIC</td>
<td>1</td>
<td>400,–</td>
<td>—</td>
</tr>
</tbody>
</table>

Table 1: Available components with cost and execution times for tasks T1 . . . T4.
(b) Draw the design points in a cost-time diagram. Which design points are Pareto points?

**Solution:** Points 2, 8, 10, 12 are Pareto-optimal.

(c) Consider an allocation without resource constraints. That means we are given an arbitrarily high number of components of each type (MIPS, DSP, FPGA and ASIC). Are there new design points? Does the set of Pareto points change?

**Solution:** See Figure 3 and Table 3. Points 2, 8, 13, 15 are now Pareto-optimal.

(d) How many design points exist at least if a task graph comprises \( n \) tasks, each task can be executed on at least three component types, and there are no resource constraints?

**Solution:** \( 3^n \)

### 8.2 Design Space Exploration with Evolutionary Algorithms

In this exercise, we consider a design space exploration problem with an evolutionary algorithm as depicted in

<table>
<thead>
<tr>
<th></th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_1 )</td>
<td>MIPS</td>
<td>MIPS</td>
<td>MIPS</td>
<td>MIPS</td>
</tr>
<tr>
<td>( T_2 )</td>
<td>DSP1</td>
<td>DSP1</td>
<td>FPGA1</td>
<td>FPGA1</td>
</tr>
<tr>
<td>( T_3 )</td>
<td>DSP2</td>
<td>DSP2</td>
<td>FPGA2</td>
<td>FPGA2</td>
</tr>
<tr>
<td>( T_4 )</td>
<td>MIPS</td>
<td>DSP1/2</td>
<td>MIPS</td>
<td>DSP</td>
</tr>
<tr>
<td>Cost</td>
<td>440.-</td>
<td>440.-</td>
<td>680.-</td>
<td>800.-</td>
</tr>
<tr>
<td>Execution time</td>
<td>27 ms</td>
<td>30 ms</td>
<td>19 ms</td>
<td>22 ms</td>
</tr>
</tbody>
</table>

Table 3: Additional design points
Figure 3: Additional design points in cost-time graph

Fig. 4

**Evolutionary Algorithm**

1. selection  
2. recombination  
3. mutation

- encoded individual (chromosome)  
- decode chromosome  
- design point (implementation)  
- fitness evaluation

Figure 4: Evolutionary Algorithm for Design Space Exploration

An individual is defined by:

(a) **Allocation** (which resources are used?)
(b) **Binding** (which task runs on which resource?)
(c) **Scheduling** (how are the tasks scheduled on a single resource?)

*Note: In this exercise the scheduling is also part of the chromosome. Usually, only the allocation and the binding is encoded, whereas the scheduling is computed and optimized in an extra step after the decoding.*

A simple problem specification is given in Fig. 5 and Fig. 6. In this example, we do not consider the communication between resources.

Tasks:

8.2.a) Encoding of Solutions
• Find a suitable encoding for the given problem specification for: (i) the allocation, (ii) the binding, and (iii) the scheduling.
• Prove the completeness of your encoding.
• Evaluate your encoding regarding the uniformity, the redundancy, and the feasibility.

Solution:
Many solutions are possible with different trade-offs. We provide one solution here:

Allocation and Binding:

<table>
<thead>
<tr>
<th>Task</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource</td>
<td>1</td>
<td>2</td>
<td>1/2</td>
<td>1/2</td>
<td>2/3</td>
<td>2/3</td>
</tr>
</tbody>
</table>

Scheduling:

<table>
<thead>
<tr>
<th>Order</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
</table>

Completeness: For completeness, we need to prove that all feasible solutions can be represented. The allocation and binding representation is complete since we have the possibility of allocating a given task to any resource which can execute it. For the scheduling representation, we must first understand how the encoding is decoded to find a schedule. At each point in time, we look at the tasks eligible for execution. For a given resource, if there are multiple eligible tasks, then we look at the schedule encoding and execute the task with lower order. This encoding encapsulates all feasible orders of execution and is therefore complete.

This encoding is redundant since multiple schedule encodings may lead to the same schedule. The encoding is non-uniform and feasible.

8.2.b) Mutation and Recombination

• Specify a mutation and a recombination operator for the given problem.
• Do the operations lead to infeasible solutions? If yes, what do you suggest for the handling of such solutions?

Solution:
Again, many possibilities. For mutation, we can randomly select a task and change its order. For crossover, we select parts of two solutions and combine individual parts to generate a new solution. Depending on the encoding, this may lead to infeasible solutions. We have several options to deal with this. These include:

(a) Discard infeasible solutions

(b) Change randomization rule such that new solutions are always feasible. This can be rather difficult to implement.

(c) Implement a penalty function that adds a large value to all objectives in-case of infeasible solution.