Hardware-Software Codesign

5. Multi-Criteria Optimization

Lothar Thiele
Lecture Synopsis

- Introduction
- Multiobjective Optimization
- Multiobjective Evolutionary Algorithms
- Implementation Aspects
Design Space Exploration

Application \rightarrow Mapping \rightarrow Estimation \rightarrow Architecture

(multi-objective) optimization
Design Space Exploration

Specification → Optimization → Evaluation → Implementation

\( G_V \)
problem graph
\( M \)
mapping set
\( G_A \)
arboriculture graph

power consumption
latency
cost
**Example: Packet Processing in Networks**

```
for i=1 to n do nothing
    call comm(a,dsf,"e")
end for
```
**Network Processors**

**Network processor** = high-performance, programmable device designed to efficiently execute communication workloads

incoming flows (packet streams)  
outgoing flows (processed packets)  

routing / forwarding  
transcoding  
encryption / decryption  

real-time flows  
e.g., voice  
e.g., sftp  
non-real-time flows  

network processor (NP)  

?  

5-7
Optimization Scenario: Overview

Given:
1. specification of the task structure (task model) = tasks to be executed for each flow
2. different usage scenarios (flow model) = sets of flows

Sought:
network processor implementation = architecture + task mapping + scheduling

Objectives:
1. maximize performance
2. minimize cost

Subject to:
1. memory constraint
2. delay constraints
Method: Black-Box Optimization

**Objective functions**

- Decision vector $x$
- Objective vector $f(x)$

(e.g. simulation model)

**Optimization Algorithm:**

only allowed to evaluate $f$
Lecture Synopsis

- Introduction
- *Multiobjective Optimization*
- Multiobjective Evolutionary Algorithms
- Implementation Aspects
Let us suppose, we would like to select a typewriting device. Criteria are

- mobility (related to weight)
- comfort (related to keyboard size and performance)

<table>
<thead>
<tr>
<th>Icon</th>
<th>Device</th>
<th>weight (kg)</th>
<th>comfort rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>🖥️</td>
<td>PC of 2009</td>
<td>20.00</td>
<td>10</td>
</tr>
<tr>
<td>🖥️</td>
<td>PC of 1984</td>
<td>7.50</td>
<td>7</td>
</tr>
<tr>
<td>🖥️</td>
<td>Laptop</td>
<td>3.00</td>
<td>9</td>
</tr>
<tr>
<td>🖥️</td>
<td>Typewriter</td>
<td>9.00</td>
<td>5</td>
</tr>
<tr>
<td>🖥️</td>
<td>Touchscreen Smartphone</td>
<td>0.11</td>
<td>2</td>
</tr>
<tr>
<td>🖥️</td>
<td>PDA with large keyboard</td>
<td>0.09</td>
<td>3</td>
</tr>
<tr>
<td>🖥️</td>
<td>PDA with small keyboard</td>
<td>0.11</td>
<td>4</td>
</tr>
<tr>
<td>🖥️</td>
<td>Organizer with tiny keyboard</td>
<td>0.08</td>
<td>1</td>
</tr>
</tbody>
</table>
Multiobjective Optimization

- Writing comfort (better)
- Mobilty (better)
- Pareto optimal dominated
- Pareto set:
  - dominated area (hypervolume)

Comparing devices:
- PC of 2009
- Laptop
- Typewriter
- PC of 1984
- PDA with small keyboard
- PDA with larger keyboard
- Smartphone
- Organizer with tiny keyboard

Better devices dominate less better devices.
Basic Definitions

- We intend to minimize a vector-valued **objective function**
  \[ f = (f_1; \ldots ; f_n) : X \rightarrow \mathbb{R}^n \]
- \(X\) denotes the **decision space**, i.e., the feasible set of alternatives for the optimization problem
- The image of the decision space \(X\) using the objective function \(f\) is denoted as the **objective space** \(Z \subset \mathbb{R}^n\) with
  \[ Z = \{ f(x) | x \in X \} \]
- A single alternative \(x \in X\) is sometimes named ‘solution’ and the corresponding objective value \(z = f(x) \in Z\) is named ‘objective vector’.

Basic question: How do we define the minimum of a vector-valued function?
**Pareto-Dominance**

**Definition**: A solution $a \in X$ weakly Pareto-dominates a solution $b \in X$, denoted as $a \preceq b$, if it is as least as good in all objectives, i.e., $f_i(a) \leq f_i(b)$ for all $1 \leq i \leq n$. Solution $a$ is better than $b$, denoted as $a < b$, iff $(a \preceq b) \land (b \not\preceq a)$.
Pareto-optimal Set

- A solution is named **Pareto-optimal**, if it is not Pareto-dominated by any other solution in \( X \).
- The set of all Pareto-optimal solutions is denoted as the Pareto-optimal set and its image in objective space as the **Pareto-optimal front**.

![Diagram](image)
The domination relation imposes a preorder on all design points: We are faced with a set of optimal solutions.

- **Reflexivity**: $a \leq a$
- **Transitivity**: if $a \leq b$ and $b \leq c$, then $a \leq c$
Optimization Alternatives

- Use of *classical single objective optimization* methods
  - simulated annealing, tabu search
  - integer linear program
  - other constructive or iterative heuristic methods
- Decision making is done before the optimization.

or

- *Population based optimization methods*
  - evolutionary algorithms / genetic algorithms
- Decision making is done after the optimization.
There are many possibilities to reduce a multi-objective problem to a single-objective problem.

**Example 1:** Aggregation (weighted sum)

\[
\text{minimize } f = w_1 f_1 + w_2 f_2 + \ldots + w_n f_n
\]

- Weights need to be fixed before optimization.
- Not all Pareto points can be found.
Single Objective Problem

**Example 2**: Single objective with constraints

\[
\text{minimize } f = f_1 \text{ with } f_2 < k_2 \ldots \ f_n < k_n
\]

bounds need to be fixed before optimization

all Pareto points can be found by ‘scanning’ through all bounds
Population-based Optimization

- In population-based methods, a whole *set of solutions* is investigated simultaneously.

- *Evolutionary algorithms* are black-box optimization methods which are randomized and population-based.

- Like other related methods, e.g. simulated annealing, they are based on the assumption that better solutions are preferably found in the *neighborhood* of good solutions.

- *Local minima* are avoided using
  - randomization, for example in the neighborhood operator or in the selection of better solutions
  - local and global neighborhood operators
Evolutionary Algorithm Cycle

Many variants of the following scheme exist.

1. A set of initial solutions (initial population) is chosen, usually at random. This set is named ‘parent’ set.

2. Solutions from the ‘parent’ set are selected (mating selection) as a basis for step 3.

3. Solutions from step 2 are changed using neighborhood operators, e.g. cross-over operators or mutation operators. The resulting set is named ‘children’ set.


5. Solutions of the set from 4. are selected based on their merit to construct the new ‘parent’ set (environmental selection).
Evolutionary Algorithm Cycle

- **representation**
- **mating selection**
- **crossover operator**
- **initial/parent set**
- **children set**
- **environmental selection**
- **mutation operator**
Evolutionary Algorithm Cycle

Some funny examples:

- **crossover**: crossover operator
- **mutation**: mutation operator
- **environmental selection**: environmental selection

Example from Johannes Bader
Lecture Synopsis

- Introduction
- Multiobjective Optimization
  - Multiobjective Evolutionary Algorithm
    - Environmental Selection
    - Neighborhood Operator
- Implementation Aspects
Evolutionary Algorithm Cycle

representation  mating selection  crossover operator

initial/parent set

environmental selection

children set

mutation operator
Environmental Selection

- How do we choose solutions that should be removed from the population?

**Informal criteria:**
- solutions should be ‘close’ to the (unknown) Pareto-optimal front (optimality)
- solutions should cover large parts of the objective space (diversity)

**Principle idea:**
- We are optimizing sets. Therefore, we need to define an indicator that characterizes the optimality of the whole set.
- We chose the optimal subset of solutions with respect to this set-indicator.
Optimality and Diversity

2-objective knapsack problem

- SPEA2
- VEGA
- extended VEGA

Trade-off between optimality and diversity.

Better

Three different evolutionary algorithms
The Hypervolume Indicator

- **Environmental selection:** Select subset of solutions that maximizes hypervolume indicator.

- Example: select optimal subset of 4 solutions from the 8 solutions in the set.

**hypervolume indicator:** hypervolume of the dominated subspace

**better**

mobility (better →)

writing comfort
The Hypervolume Indicator

(For the following, we again minimize \( f \).)

Given a set of solutions \( A \subseteq X \) and a set of reference points \( R \subseteq \mathbb{R}^n \). Then the hypervolume indicator \( I_H(A, R) \) of \( A \) with respect to \( R \) is defined as

\[
I_H(A, R) = \int_{z \in H(A, R)} d\mathbf{z}
\]

where \( H(A, R) \) is the dominated space of \( A \) regarding \( R \):

\[
H(A, R) = \{ z \in \mathbb{R}^n \mid \exists a \in A : \forall r \in R : (f(a) \leq z \leq r) \}
\]
Hypervolume Indicator

- **Why does the hypervolume indicator lead to diversity and optimality?**
- **Diversity:** It appears that the indicator well covers the intuitive notion of diversity in objective space.
Hypervolume Indicator

- **Optimality**: One can show the following relation between the hypervolume indicator and Pareto-dominance:
  - Suppose that a set of solutions \( A \) is better than a set \( B \) (\( A \prec B \)), i.e. every solution of \( B \) is weakly Pareto-dominated by at least one solution in \( A \) (\( A \preceq B \)) but not (\( B \preceq A \)).
  - Then, the **hypervolume of \( A \) is larger than that of \( B \)**.

In other words, if I have a set of solutions with the largest possible hypervolume, then there is no other set that dominates it completely.

Therefore, it makes sense to **determine a set of solutions that maximizes the hypervolume indicator**.
Template of a Practical Algorithm

Algorithm 1: Main Loop

1: generate initial set of solutions $P$ of size $m$, i.e., randomly choose $m$ solutions
2: while termination criterion not fulfilled do
3: \hspace{1cm} $P' \leftarrow \text{heuristicSetMutation}(P)$
4: \hspace{1cm} if $I(P') \geq I(P)$ then
5: \hspace{2cm} $P \leftarrow P'$
6: \hspace{1cm} return $P$

The algorithm can be seen as a simple Greedy strategy: if the new population is not worse than the old one, use it in the next iteration.

hypervolume indicator combines mating selection, crossover, mutation and environmental selection.
Template of a Practical Algorithm

Algorithm 2 Heuristic Set Mutation

1: procedure heuristicSetMutation(P)
2: generate $k$ solutions $r_1, \ldots, r_k \in X$ based on $P$
3: $P' \leftarrow P \cup \{r_1, \ldots, r_k\}$
4: while $|P'| > m$ do
5:   for all $a \in P'$ do
6:     $\delta_a \leftarrow I(P') - I(P' \setminus \{a\})$
7:   choose $p \in P'$ with $\delta_p = \min_{a \in P'} \delta_a$
8:   $P' \leftarrow P' \setminus \{p\}$
9: return $P'$

This is a **heuristic to select the best $m$ solutions** in $P'$:
Solutions are removed from $P'$ one-by-one; in each iteration, the solution with the smallest loss in hypervolume is removed.

Combines mating selection, crossover and mutation.
Union of parent and children population.
Lecture Synopsis

- Introduction
- Multiobjective Optimization
- *Multiobjective Evolutionary Algorithm*
  - Environmental Selection
  - *Neighborhood Operator*
- Implementation Aspects
Evolutionary Algorithm Cycle

**representation**

mating selection

crossover operator

initial/parent set

children set

environmental selection

mutation operator
Representation and Neighborhood

- Usually, neighborhood operators such as crossover and mutation are based on a representation of solutions.
- A representation corresponds to an abstract data structure that encodes a solution.
- Neighborhood operators work on representations.

Simple example:
- Decision space: All partitionings of n objects into m blocks.
- Representation of single solution using a vector of length n with elements in [1, m]. Example for n=6 and m=3:

```
1 2 3 4 5 6
2 1 1 1 3 2
```
Representation

solutions encoded by vectors, matrices, trees, lists, ...

Issues:
- completeness (each solution has an encoding)
- uniformity (all solutions are represented equally often)
- feasibility (each encoding maps to a feasible solution)
Example: Binary Vector Encoding

**Given:** graph

**Goal:** find minimum subset of nodes such that each edge is connected to at least one node of this subset (minimum vertex cover)

<table>
<thead>
<tr>
<th>nodes</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>selected?</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Example: Integer Vector Encoding

**Given:** graph, k colors

**Goal:** assign each node one of the k colors such that the number of connected nodes with the same color is minimized (graph coloring problem)
Example: Real Vector Encoding

\[ G_2(\bar{x}) = \left| \sum_{i=1}^{n} \cos^4(x_i) - 2 \prod_{i=1}^{n} \cos^2(x_i) \right| \]

\[ \sqrt{\sum_{i=1}^{n} i x_i^2} \]

parameters values

\[ x_1 \quad x_2 \quad x_3 \quad x_4 \quad \ldots \quad x_n \]

\[ 0.33 \quad 0.53 \quad 1.03 \quad 3.25 \quad \ldots \quad 9.83 \]

[Michalewicz, Fogel: How to Solve it. Springer 2000]
Tree Example: Parking a Truck

Goal: find function \( c \) with \( u = c(x, y, d, t) \)
## Search Space for the Truck Problem

### Operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLUS(a,b)</td>
<td>returns a+b</td>
</tr>
<tr>
<td>MINUS(a,b)</td>
<td>returns a-b</td>
</tr>
<tr>
<td>MUL(a,b)</td>
<td>returns a*b</td>
</tr>
<tr>
<td>DIV(a,b)</td>
<td>return ( \frac{a}{b} ), if ( b \neq 0 ), else 1</td>
</tr>
<tr>
<td>ATG(a,b)</td>
<td>returns ( \text{atan2}(a,b) ), if ( a \neq 0 ), else 0</td>
</tr>
<tr>
<td>IFLTZ(a,b,c)</td>
<td>returns b, if ( a &lt; 0 ), else returns c</td>
</tr>
</tbody>
</table>

### Arguments:

- X: position x
- Y: position y
- DIFF: cab angle d
- TANG: trailer angle t

### Search space:

set of symbolic expression using the above operators and arguments
Example Solution: Tree Representation

\[ u = (x - d) \times (y + t) \]

 encodes the function (symbolic expression): \( u = (x - d) \times (y + t) \)
A Solution Found by an EA

truck simulation

encoded tree
Evolutionary Algorithm Cycle

representation          mating selection          crossover operator

initial/parent set

environmental selection

children set

mutation operator
Vector Mutation: Examples

**Bit vectors:**

- 1 0 1 1 1 0
- Each bit is flipped with probability 1/6
- 1 0 0 1 1 0

**Permutations:**

- 1 2 3 4 5 6
  - swap
  - 1 3 4 2 5 6
  - rearrange
  - 1 3 4 2 5 6
Mutation Operators on Trees: Grow

grow

MULT

MINUS

DIFF

Y

TANG

X

Y

MINUS

PLUS

MULT

MINUS

DIFF

Y

MULT

X

Y

X

X
Mutation Operators on Trees: Shrink

\[
\text{shrink}
\]
Mutation Operators on Trees: Switch
Mutation Operators on Trees: Replace

```
MULT
   /\   \
MINUS PLUS
   /\   /\  
MINUS DIFF Y TANG
   /\   /\ 
 X   Y  
```

replace

```
MULT
   /\   \
MINUS DIFF Y TANG
   /\   /\ 
 X   Y  
```
Vector Crossover: Examples

**Bit vectors:**

\[
\begin{array}{cccccc}
1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccc}
1 & 1 & 0 & 0 & 0 & 1 \\
\end{array}
\]

**Permutations:**

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
6 & 2 & 3 & 4 & 1 & 5 \\
\end{array}
\]

\[
\begin{array}{cccccc}
1 & 2 & 3 & 6 & 4 & 5 \\
\end{array}
\]

parents

\[
\begin{array}{cccccc}
1 & 2 & 3 & 6 & 4 & 5 \\
\end{array}
\]

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
6 & 2 & 3 & 4 & 1 & 5 \\
\end{array}
\]

child
Recombination of Trees

MULT

MINUS

DIFF

X

Y

PLUS

Y

TANG

DIFF

MINUS

PLUS

TANG

exchange

MULT

Y

TANG
Constraint Handling

**Constraint:** \( g(x) \geq 0 \)

**Approaches:**
- representation is chosen such that decoding always yields a feasible solution
- construct initialization and neighborhood operators such that infeasible solutions are not generated
- add to children population only feasible solutions
- in environmental selection, preferably select feasible solutions
- calculate constraint violation \( g(x) \) and incorporate it into objective function using a penalty function: \( \text{penalty}(x) > 0 \) if \( g(x) < 0 \), \( \text{penalty}(x) = 0 \) if \( g(x) \geq 0 \). For example, add penalty function to every objective.
- include the constraints as new objectives
Lecture Synopsis

- Introduction
- Multiobjective Optimization
- Multiobjective Evolutionary Algorithms
- *Implementation Aspects*
Implementation Framework

We want a framework that

- provides ready-to-use modules (algorithms / applications)
- is simple to use
- is independent of programming language and OS
- comes with minimum overhead

Idea: separate problem-dependent from problem-independent part
PISA: Implementation

application independent:
- environmental selection
- individuals are described by IDs and objective vectors

handshake protocol:
- state / action
- individual IDs
- objective vectors
- parameters

application dependent:
- neighborhood (crossover and mutation) operators
- stores and manages individuals
Download of Selectors and Variators

This page contains the currently available variators and selector modules, see also Principles of PISA. The variators are mainly test and benchmark problems that can be used to assess the performance of different optimizers. EXPO is a complex application from the area of computer design that can be used as a benchmark problem too. The selectors are state-of-the-art evolutionary multi-objective optimization methods. If you want to write or submit a module, please look at Write and Submit a Module. Links to documentation on the PISA specification can be found at Documentation.

### Optimization Problems (variant)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Source</th>
<th>Binaries</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOTZ - Demonstration Program</td>
<td></td>
<td>in C</td>
<td>Solaris, Windows, Linux</td>
</tr>
<tr>
<td>LOTZ2 - Leading Ones Trailing Zeros</td>
<td></td>
<td>in C</td>
<td>Solaris, Windows, Linux</td>
</tr>
<tr>
<td>LOTZ2 - Java Example Variator</td>
<td></td>
<td>in Java</td>
<td>Windows, Linux</td>
</tr>
<tr>
<td>Knapsack Problem</td>
<td></td>
<td>in C</td>
<td>Solaris, Windows, Linux</td>
</tr>
<tr>
<td>EXPO - Network Processor Design Problem</td>
<td></td>
<td></td>
<td>Solaris, Windows, Linux</td>
</tr>
<tr>
<td>WFG - Walking Fishgroup testproblems</td>
<td></td>
<td>in C</td>
<td>Solaris, Linux 32bit, Linux 64bit</td>
</tr>
<tr>
<td>DT LZ - Continuous Test Functions (incl. ZDT)</td>
<td></td>
<td>in C</td>
<td>Solaris, Windows, Linux</td>
</tr>
<tr>
<td>BBV - Biobjective Binary Value Problem</td>
<td></td>
<td>in C</td>
<td>Solaris, Windows, Linux</td>
</tr>
</tbody>
</table>

### Optimization Algorithms (selector)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Source</th>
<th>Binaries</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIBEA - Simple Indicator-Based Evolutionary Algorithm</td>
<td></td>
<td>in Java</td>
<td>tar.gz, zip</td>
</tr>
<tr>
<td>HypE - HyperVolume Estimation Algorithm for Multiobjective Optimization</td>
<td></td>
<td>in C</td>
<td>Windows, Linux 32bit, Linux 64bit</td>
</tr>
<tr>
<td>SEMO - Demonstration Program</td>
<td></td>
<td>in C</td>
<td>Solaris, Windows, Linux</td>
</tr>
<tr>
<td>SEMO2 - Simple Evolutionary Multiobjective Optimizer</td>
<td></td>
<td>in C</td>
<td>Solaris, Windows, Linux</td>
</tr>
<tr>
<td>FEMO - Fair Evolutionary Multiobjective Optimizer</td>
<td></td>
<td>in C</td>
<td>Solaris, Windows, Linux</td>
</tr>
<tr>
<td>SPEA2 - Strength Pareto Evolutionary Algorithm</td>
<td></td>
<td>in C</td>
<td>Solaris, Windows, Linux</td>
</tr>
<tr>
<td>NSGA2 - Non-dominated Sorting Genetic Algorithm</td>
<td></td>
<td>in C</td>
<td>Solaris, Windows, Linux</td>
</tr>
<tr>
<td>ESEA - Epsilon-Constraint Evolutionary Algorithm</td>
<td></td>
<td>in C</td>
<td>Solaris, Windows, Linux</td>
</tr>
</tbody>
</table>