Hardware-Software Codesign

5. Multi-Criteria Optimization

Lothar Thiele
System Design

- Specification
- System Synthesis
- Estimation
- SW-Compilation
- Instruction Set
- HW-Synthesis
- Intellectual Prop. Code
- Machine Code
- Net lists
- Intellectual Prop. Block
Lecture Synopsis

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

Application → Mapping → Estimation → Architecture

(multi-objective) optimization
Design Space Exploration

1. Specification
2. Optimization
3. Evaluation
4. Implementation

- Specification
- Optimization
- Evaluation
- Implementation

- Problem graph
- Mapping set
- Architecture graph

- Power consumption
- Latency
- Cost
Example: Packet Processing in Networks

method (\texttt{esd})
for \texttt{i=1} to \texttt{n}
do nothing
end for

call comm(a, dsf, *e);
end for

Embedded Internet Devices

Wearable Computing

©UCB Rabaey

Mobile Internet

Access

Core

Wireless node

Offices

Entrance

Example: Packet Processing in Networks

Embedded Internet Devices

Wearable Computing

©UCB Rabaey

Mobile Internet

Access

Core

Wireless node

Offices

Entrance

Example: Packet Processing in Networks

Embedded Internet Devices

Wearable Computing

©UCB Rabaey

Mobile Internet

Access

Core

Wireless node

Offices

Entrance

Example: Packet Processing in Networks

Embedded Internet Devices

Wearable Computing

©UCB Rabaey

Mobile Internet

Access

Core

Wireless node

Offices

Entrance

Example: Packet Processing in Networks

Embedded Internet Devices

Wearable Computing

©UCB Rabaey

Mobile Internet

Access

Core

Wireless node

Offices

Entrance

Example: Packet Processing in Networks

Embedded Internet Devices

Wearable Computing

©UCB Rabaey

Mobile Internet

Access

Core

Wireless node

Offices

Entrance
Network Processors

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

- **Incoming flows** (packet streams)
  - real-time flows
    - e.g., voice
  - non-real-time flows
    - e.g., sftp
- **Routing / Forwarding**
  - transcoding
  - encryption / decryption
- **Outgoing flows** (processed packets)

**network processor** (NP)
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

Stretch-Module

Decision-Module

Handling-Module

Vehicle-Module

Decision-Module Handling-Module

Optimization Algorithm:
only allowed to evaluate f
Lecture Synopsis

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

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

<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

The diagram illustrates the trade-offs between writing comfort and mobility for various computing devices over time, from a typewriter (1984) to a PC of 2009, emphasizing the concept of Pareto optimization. The diagram shows how devices dominate each other in terms of writing comfort and mobility, with the Pareto optimal set identified by solid dots and dominated devices by open circles. The hypervolume is represented by the shaded area, highlighting the efficient frontier of solutions.

Better devices (dominates) are indicated by arrows pointing towards the right, indicating improved writing comfort. Devices that are incomparable are shown with a dashed line.
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**.

$$\text{f}_2$$

objective space Z:

$$\text{f}_1$$

Pareto-optimal = not dominated

dominated

better
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 \( a \leq c \)

**Preorder**

```
+--- OBJECTIVE SPACE: ---+
|  f2                        |
| h ------------------------ |
| e ------------------------ |
| d ------------------------ |
| f ------------------------ |
| g ------------------------ |
| c ------------------------ |
| h ------------------------ |
| i ------------------------ |
| j ------------------------ |
| k ------------------------ |
| l ------------------------ |
| m ------------------------ |
|  f1                        |
```

**Preordered Set**

```
+--- PREORDERED SET: ---+
| a ------------ b --- c |
| d ------------ e --- f |
| g ------------ h --- i |
| j ------------ k --- l |
| m ------------ n --- o |
```

The dominance relation imposes a preorder on all design points: We are faced with a set of optimal solutions.
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

minimize \( f = f_1 \) 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

environmental selection   mutation operator

initial/parent set

children set

Evolutionary Algorithm Cycle
Evolutionary Algorithm Cycle

Some funny examples:

1. **Crossover Operator**
   - Input: Two parent solutions
   - Output: Two offspring solutions

2. **Mutation Operator**
   - Input: A parent solution
   - Output: A modified offspring solution

3. **Environmental Selection**
   - Input: A population of solutions
   - Output: A new population of solutions
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
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**

Three different evolutionary algorithms

Trade-off between optimality and diversity.
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
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

$$I_H(A, R) = \int_{z \in H(A, R)} d\mathbb{z}$$

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

$$H(A, R) = \{z \in \mathbb{R}^n | \exists a \in A : \exists r \in R : (f(a) \leq z \leq r)\}$$
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 we do not have \((B \preceq A)\).
  
  - 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: \( P' \leftarrow \text{heuristicSetMutation}(P) \)

4: if \( I(P') \geq I(P) \) then

5: \( P \leftarrow P' \)

6: return \( P \)

- hypervolume indicator
- combines mating selection, crossover, mutation and environmental selection

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
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.

algorithm 2 Heuristic Set Mutation

```
procedure heuristicSetMutation(P)
generate k solutions r₁, ..., rₖ ∈ X based on P
P' ← P ∪ \{r₁, ..., rₖ\}
while |P'| > m do
    for all a ∈ P' do
        δₐ ← I(P') - I(P' \ \{a\})
    end for
    choose p ∈ P' with δₚ = minₐ∈P' δₐ
    P' ← P' \ \{p\}
end while
return P'
```
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 \( m=3 \):

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

- **search space**
- **decoder**
- **decision space**
- **objectives**
- **objective space**

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

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

<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>colors</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>...</td>
</tr>
</tbody>
</table>
Example: Real Vector Encoding

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

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 $u = c(x, y, d, t)$

- dock
- steering angle $u$
- cab
- trailer
- position $(x, y)$
- constant speed

$u$ constant speed

d

$\text{DIFF}$
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 a/b, if b &lt;&gt; 0, else 1</td>
</tr>
<tr>
<td>ATG(a,b)</td>
<td>returns atan2(a,b), if a&lt;&gt; 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

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**
- **children set**
- **environmental selection**
- **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 4 3 2 5 6

1 2 3 4 5 6

Rearrange

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

MULT

MINUS

DIFF

Y

TANG

X

Y

MULT

MINUS

DIFF

Y

MULT

X

Y

X

X
Mutation Operators on Trees: Shrink

shrink
Mutation Operators on Trees: Switch

\[
\text{MULT} \quad \text{PLUS} \quad \text{TANG} \\
\text{MINUS} \quad \text{DIFF} \quad Y \\
X \quad Y
\]

\[
\text{MULT} \quad \text{PLUS} \quad \text{TANG} \\
\text{MINUS} \quad \text{DIFF} \quad Y \\
X \quad Y
\]
Mutation Operators on Trees: Replace

replace
Vector Crossover: Examples

**Bit vectors:**

```
1 1 0 0 0 1 0
1 0 1 0 0 1
```

**Permutations:**

Parents:
```
1 2 3 4 5 6
6 2 3 4 1 5
```

Child:
```
1 2 3 6 4 5
```
Recombination of Trees

MULT

MINUS

DIFF

MINUS

X

Y

PLUS

Y

TANG

PLUS

TANG

MINUS

DIFF

MULT

Y

TANG

exchange
Constraint Handling

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

Solution in decision space

**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 (cross-over and mutation) operators
- stores and manages individuals
### Download of Selectors and Variators

This page contains the currently available selectors and selector modules, see also principles of PISA. The selectors are mainly test and benchmark problems that can be used to assess the performance of different optimizers. EXPO is a complex application form 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.

<table>
<thead>
<tr>
<th>Optimization Problems (variant)</th>
<th>Optimization Algorithms (selector)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LOTZ - Demonstration Program</strong></td>
<td><strong>SIBEA - Simple Indicator Based Evolutionary Algorithm</strong></td>
</tr>
<tr>
<td>Source: in C</td>
<td>Source: in Java as .jar or .zip</td>
</tr>
<tr>
<td>Binaries: Solaris, Windows, Linux</td>
<td>Binaries: Solaris, Windows, Linux</td>
</tr>
<tr>
<td>more...</td>
<td>more...</td>
</tr>
<tr>
<td><strong>LOTZ2 - Leading Ones Trailing Zeros</strong></td>
<td><strong>HypE - Hypervolume Estimation Algorithm for Multiobjective Optimization</strong></td>
</tr>
<tr>
<td>Source: in C</td>
<td>Source: in C</td>
</tr>
<tr>
<td>Binaries: Solaris, Windows, Linux</td>
<td>Binaries: Windows, Linux 32bit, Linux 64bit</td>
</tr>
<tr>
<td>more...</td>
<td>more...</td>
</tr>
<tr>
<td><strong>LOTZ2 - Java Example Variator</strong></td>
<td><strong>SEMO - Demonstration Program</strong></td>
</tr>
<tr>
<td>Source: in Java</td>
<td>Source: in C</td>
</tr>
<tr>
<td>Binaries: Windows, Linux</td>
<td>Binaries: Solaris, Windows, Linux</td>
</tr>
<tr>
<td>more...</td>
<td>more...</td>
</tr>
<tr>
<td><strong>Knapsack Problem</strong></td>
<td><strong>SEMO2 - Simple Evolutionary Multiobjective Optimizer</strong></td>
</tr>
<tr>
<td>Source: in C</td>
<td>Source: in C</td>
</tr>
<tr>
<td>Binaries: Solaris, Windows, Linux</td>
<td>Binaries: Solaris, Windows, Linux</td>
</tr>
<tr>
<td>more...</td>
<td>more...</td>
</tr>
<tr>
<td><strong>EXPO - Network Processor Design Problem</strong></td>
<td><strong>FEMO - Fair Evolutionary Multiobjective Optimizer</strong></td>
</tr>
<tr>
<td>Source: as .tar, .zip</td>
<td>Source: in C</td>
</tr>
<tr>
<td>Binaries: (incl. .R file), Solaris, Windows, Linux</td>
<td>Binaries: Solaris, Windows, Linux</td>
</tr>
<tr>
<td>Binaries: (without .R file), Solaris, Windows, Linux</td>
<td>more...</td>
</tr>
<tr>
<td>more...</td>
<td>more...</td>
</tr>
<tr>
<td><strong>WFG - Walking Fishgroup testproblems</strong></td>
<td><strong>SPEA2 - Strength Pareto Evolutionary Algorithm</strong> 2</td>
</tr>
<tr>
<td>Source: in C</td>
<td>Source: in C</td>
</tr>
<tr>
<td>Binaries: Linux 32bit, Linux 64bit</td>
<td>Binaries: Solaris, Windows, Linux</td>
</tr>
<tr>
<td>more...</td>
<td>more...</td>
</tr>
<tr>
<td><strong>DTLZ - Continuous Test Functions (incl. ZDT)</strong></td>
<td><strong>NSGA2 - Nondominated Sorting Genetic Algorithm</strong> 2</td>
</tr>
<tr>
<td>Source: in C</td>
<td>Source: in C</td>
</tr>
<tr>
<td>Binaries: Solaris, Windows, Linux</td>
<td>Binaries: Solaris, Windows, Linux</td>
</tr>
<tr>
<td>more...</td>
<td>more...</td>
</tr>
<tr>
<td><strong>BBV - Biobjective Binary Value Problem</strong></td>
<td><strong>ECEA - Epsilon-Constraint Evolutionary Algorithm</strong></td>
</tr>
<tr>
<td>Source: in C</td>
<td>Source: in C</td>
</tr>
<tr>
<td>Binaries: Solaris, Windows, Linux</td>
<td>Binaries: Solaris, Windows, Linux</td>
</tr>
<tr>
<td>more...</td>
<td>more...</td>
</tr>
<tr>
<td><strong>NLOTZ - Generalization of the LOTZ Problem</strong></td>
<td></td>
</tr>
</tbody>
</table>