technische universität dortmund

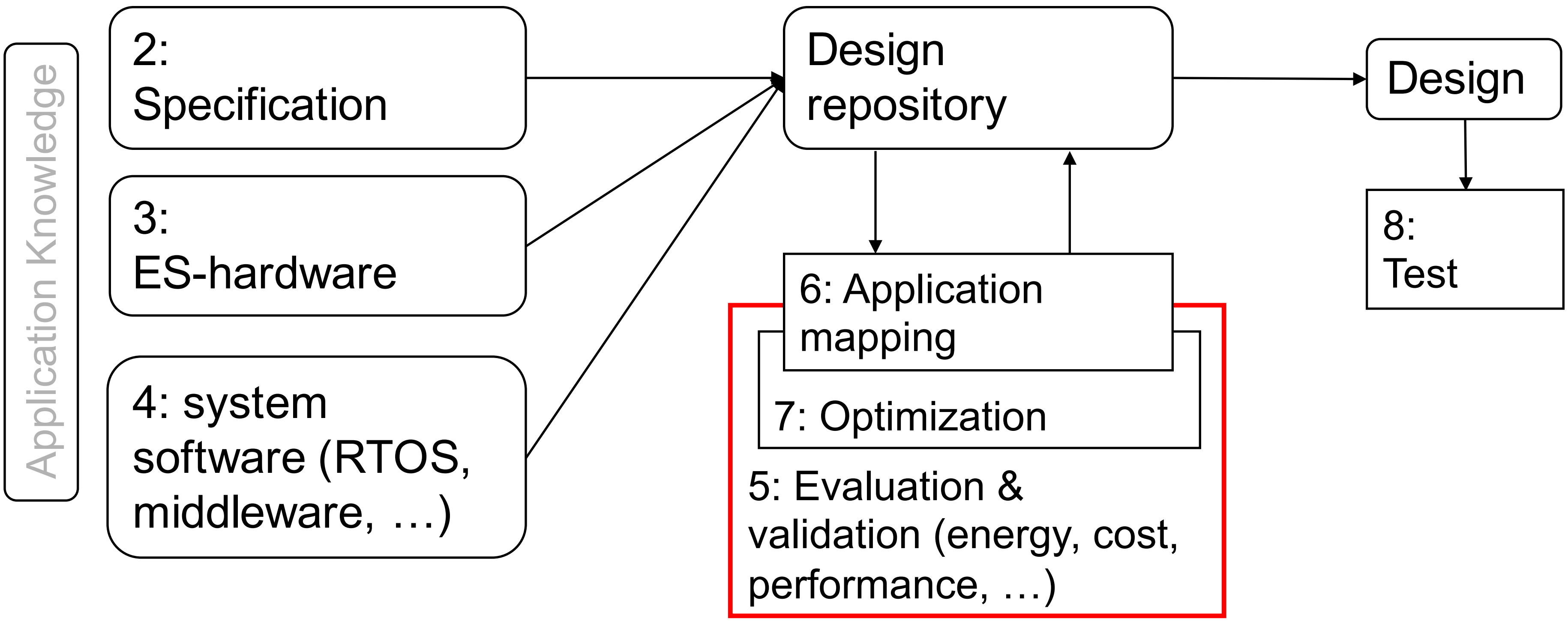
fakultät für informatik

Evaluation and Validation

Jian-Jia Chen
(Slides are based on
Peter Marwedel)
TU Dortmund, Informatik 12
Germany

2018年 11 月 27 日

Structure of this course



Numbers denote sequence of chapters

Validation and Evaluation

Definition: *Validation* is the process of checking whether or not a certain (possibly partial) design is appropriate for its purpose, meets all constraints and will perform as expected (yes/no decision).

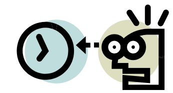
Definition: Validation with mathematical rigor is called *(formal) verification*.

Definition: *Evaluation* is the process of computing quantitative information of some key characteristics of a certain (possibly partial) design.

How to evaluate designs according to multiple criteria?

Many different criteria are relevant for evaluating designs:

- Average & worst case delay



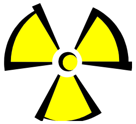
- power/energy consumption

- thermal behavior

- reliability, safety, security

- cost, size

- weight
- EMC characteristics

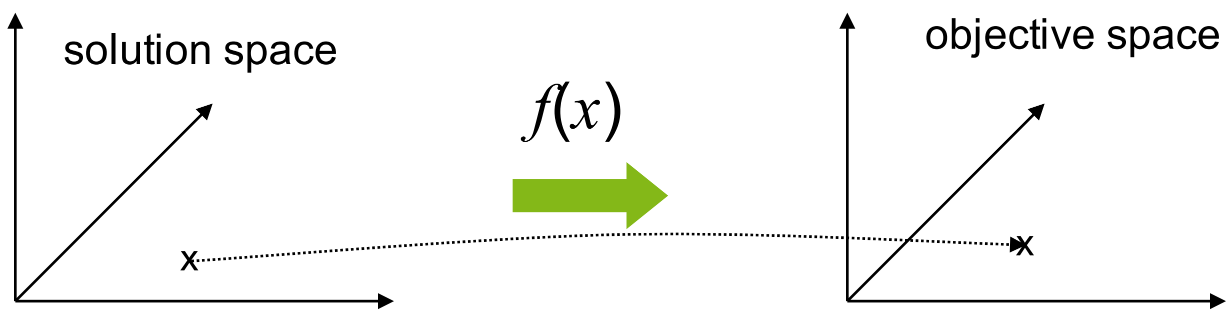


- radiation hardness, environmental friendliness, ..

How to compare different designs?
(Some designs are “better” than others)

Definitions

- Let X: m-dimensional **solution space** for the design problem. Example: dimensions correspond to # of processors, size of memories, type and width of busses etc.
- Let F: n-dimensional **objective space** for the design problem. Example: dimensions correspond to average and worst case delay, power/energy consumption, size, weight, reliability, …
- Let $f(x)=(f_1(x),\ldots,f_n(x))$ where $x \in X$ be an **objective function.** We assume that we are using $f(x)$ for evaluating designs.



Pareto points

- We assume that, for each objective, an order < and the corresponding order ≤ are defined.
- **Definition**:
  Vector $u=(u_1,...,u_n) \in F$ **dominates** vector $v=(v_1,...,v_n) \in F$
  $\Leftrightarrow$
  u is "better" than v with respect to *at least* one objective and not worse than v with respect to all other objectives:

  $$\forall i \in \{1,...,n\} : u_i \leq v_i \; \wedge$$
  $$\exists i \in \{1,...,n\} : u_i < v_i$$

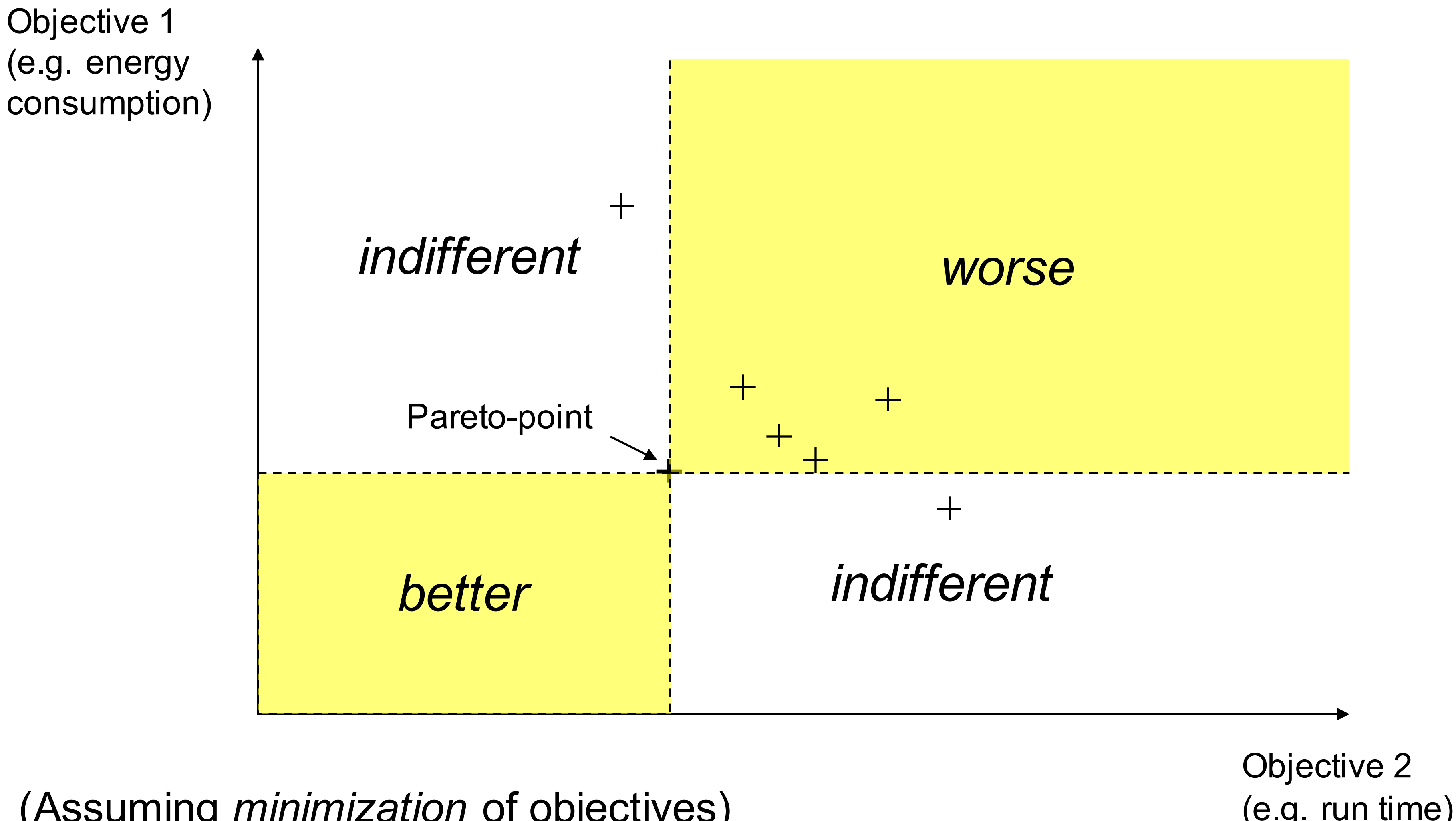
- **Definition:**
  Vector $u \in F$ is **indifferent** with respect to vector $v \in F$
  $\Leftrightarrow$ neither u dominates v nor v dominates u

Pareto points

- A solution $x \in X$ is called **Pareto-optimal** with respect to X $\Leftrightarrow$ there is no solution $y \in X$ such that $u=f(x)$ is dominated by $v=f(y)$. x is a **Pareto point**.
- **Definition**: Let $S \subseteq F$ be a subset of solutions. $v \in F$ is called a **non-dominated solution** with respect to S $\Leftrightarrow$ v is not dominated by any element $\in S$.
- v is called **Pareto-optimal** $\Leftrightarrow$ v is non-dominated with respect to all solutions F.
- A **Pareto-set** is the set of all Pareto-optimal solutions

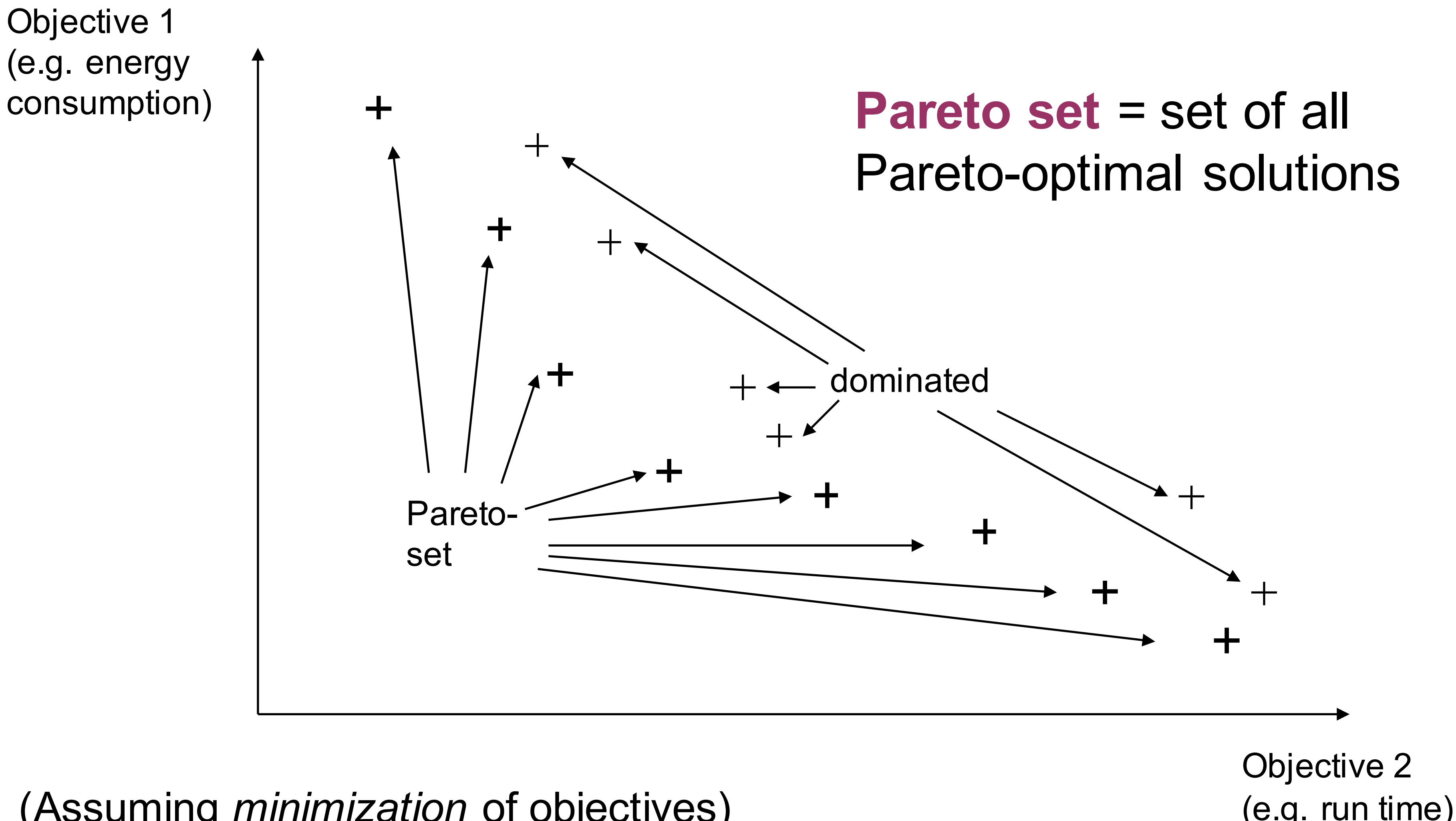
Pareto-sets define a **Pareto-front**
(boundary of dominated subspace)

Pareto Point



(Assuming *minimization* of objectives)

Pareto Set

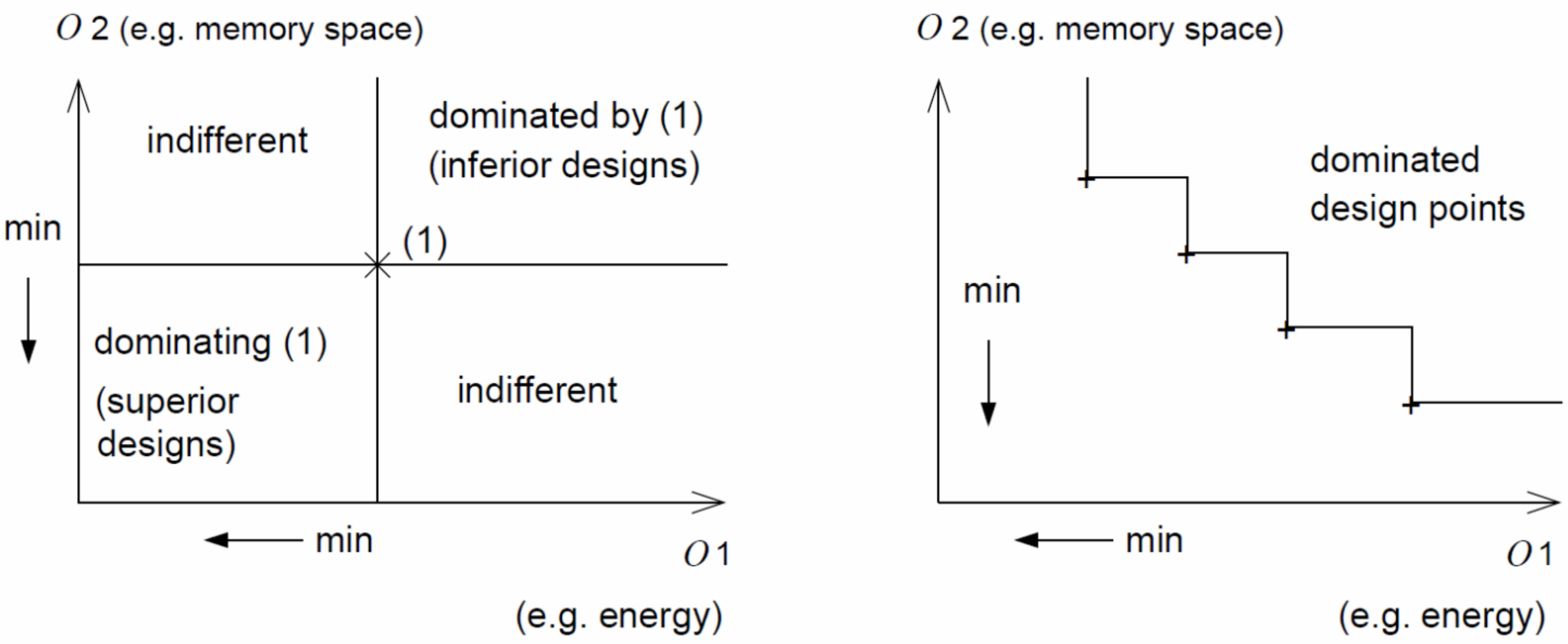


(Assuming *minimization* of objectives)

One more time …

Pareto point

Pareto front



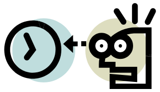
Design space evaluation

Design space evaluation (DSE) based on Pareto-points is the process of finding and returning a set of Pareto-optimal designs to the user, enabling the user to select the most appropriate design.

How to evaluate designs according to multiple criteria?

Many different criteria are relevant for evaluating designs:

- Average & worst case delay
- power/energy consumption
- thermal behavior
- reliability, safety, security
- cost, size
- weight
- EMC characteristics
- radiation hardness, environmental friendliness, ..

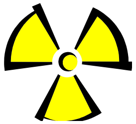









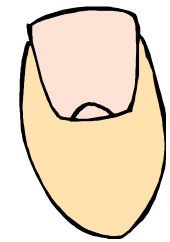





How to compare different designs?
(Some designs are “better” than others)

Average delays (execution times)

- **Estimated** average execution times :
  Difficult to generate sufficiently precise estimates;
  Balance between run-time and precision

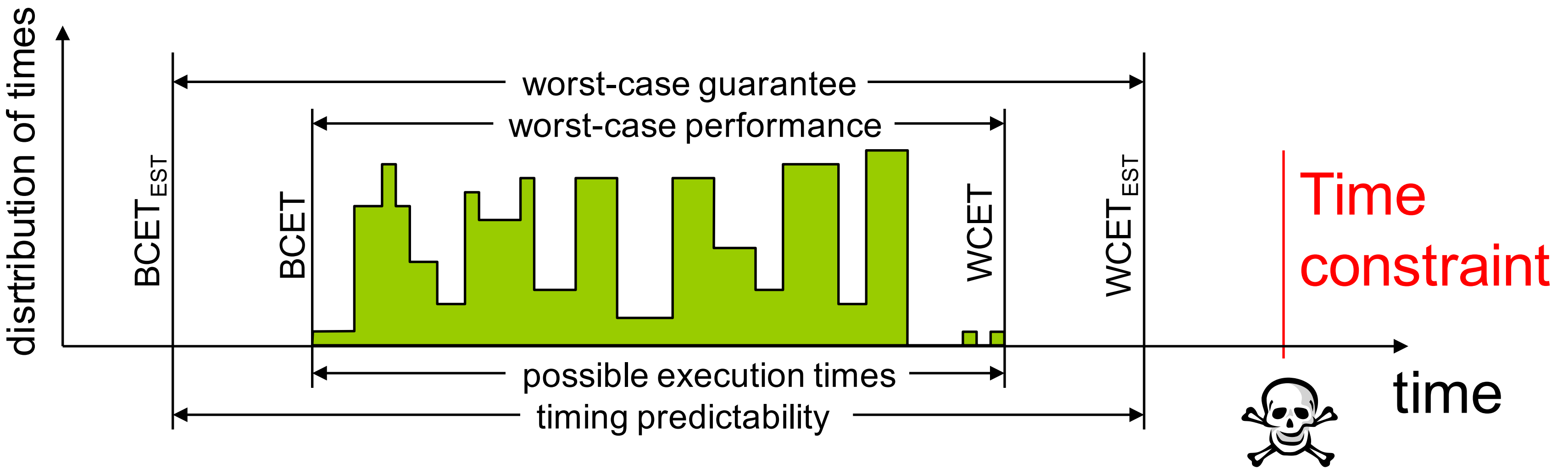
π x 

- **Accurate** average execution times:
  As precise as the input data is.

We need to compute **average** and **worst case** execution times

Worst case execution time (1)

Definition of worst case execution time:



$WCET_{EST}$ must be

1. safe (i.e. ≥ WCET) and
2. tight ($WCET_{EST}$-WCET≪$WCET_{EST}$)

Worst case execution times (2)

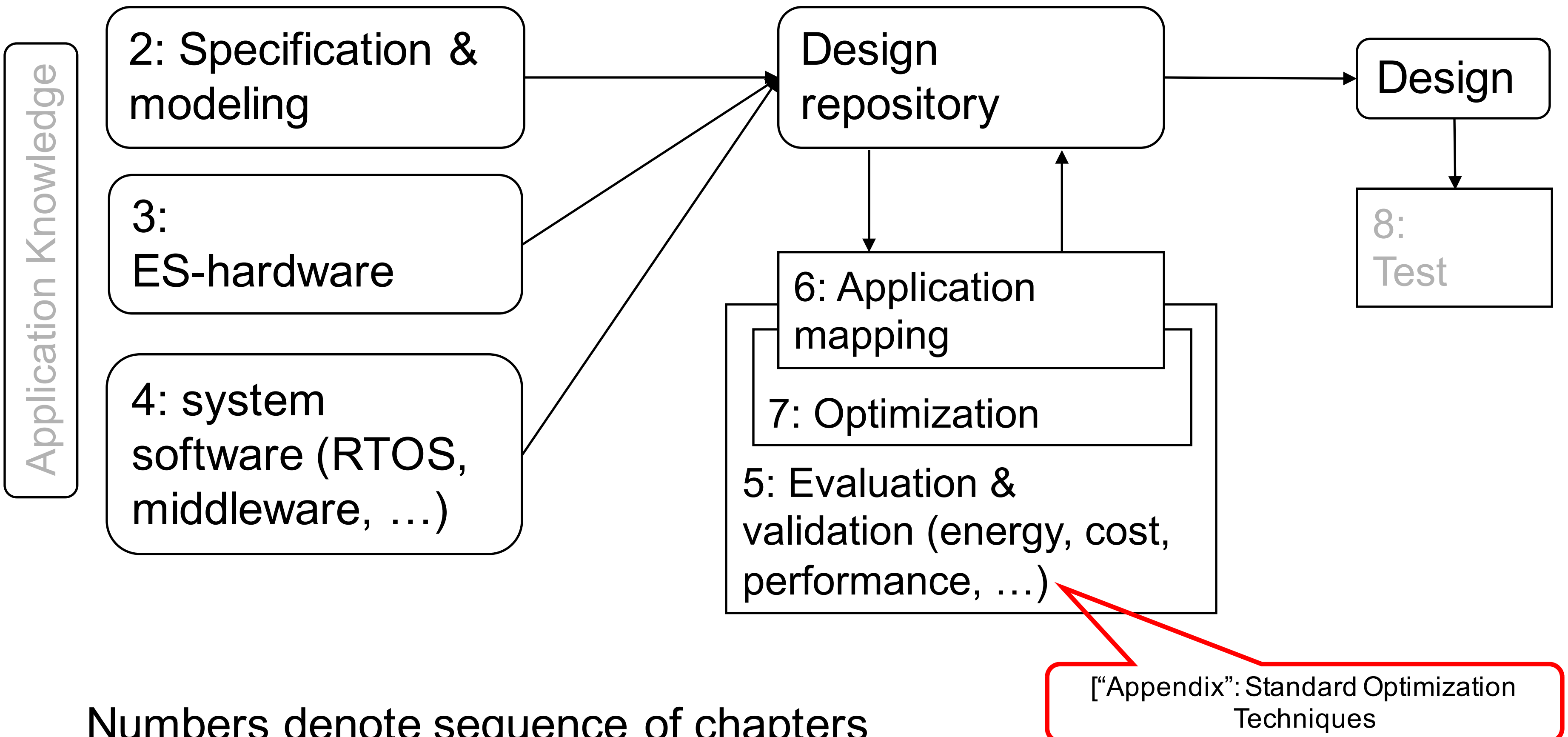
Complexity:

- in the general case: undecidable if a bound exists.
- for restricted programs: simple for “old“ architectures, very complex for new architectures with pipelines, caches, interrupts, virtual memory, etc.

Approaches:

- for hardware: requires detailed timing behavior
- for software: requires availability of machine programs; complex analysis (see, e.g., www.absint.de)

Structure of this course



Numbers denote sequence of chapters

Integer linear programming models

Ingredients:

- Cost function
- Constraints

Involving linear expressions of integer variables from a set X

Cost function

$$C = \sum_{x_i \in X} a_i x_i \text{ with } a_i \in \mathbb{R}, x_i \in \mathbb{N} \quad (1)$$

Constraints: $\forall j \in J: \sum_{x_i \in X} b_{i,j}\, x_i \geq c_j \text{ with } b_{i,j}, c_j \in \mathbb{R} \quad (2)$

Def.: The problem of minimizing (1) subject to the constraints (2) is called an **integer linear programming (ILP) problem**.

If all x_i are constrained to be either 0 or 1, the ILP problem is said to be a **0/1 integer linear programming problem**.

Example

$$C = 5x_1 + 6x_2 + 4x_3$$

$$x_1 + x_2 + x_3 \geq 2$$

$$x_1, x_2, x_3 \in \{0,1\}$$

x_1	x_2	x_3	C	
0	1	1	10	
1	0	1	9	← Optimal
1	1	0	11	
1	1	1	15	

Remarks on integer programming

- Maximizing the cost function: just set $C'=-C$
- Integer programming is NP-complete.
- Running times depend exponentially on problem size, but problems of >1000 vars solvable with good solver (depending on the size and structure of the problem)
- The case of $x_i \in \mathbb{R}$ is called *linear programming* (LP). Polynomial complexity, but most algorithms are exponential, in practice still faster than for ILP problems.
- The case of some $x_i \in \mathbb{R}$ and some $x_i \in \mathbb{N}$ is called *mixed integer-linear programming.*
- ILP/LP models good starting point for modeling, even if heuristics are used in the end.
- Solvers: lp_solve (public), CPLEX (commercial), …

An Example: Knapsack Problem

Example IP formulation:
The Knapsack problem:

I wish to select items to put in my backpack.

There are *m* items available.
Item *i* weights w_i kg,
Item *i* has value v_i.
I can carry Q kg.

$$\text{Let } x_i = \begin{cases} 1 & \text{if I select item } i \\ 0 & \text{otherwise} \end{cases}$$

$$\max \quad \sum_i x_i v_i$$

$$\text{s.t.} \quad \sum_i x_i w_i \le Q$$

$$x_i \in \{0,1\}, \ \forall \text{ i}$$

Variance of Knapsack Problem

- ***Given*** a set of periodic tasks with implicit deadlines
  - Task τ_i: period T_i,
    - Options: Execution without/with scratchpad memory (SPM)
    - Without SPM: Worst-case execution time $C_{i,1}$
    - With SPM: required m_i scratchpad memory size and Worst-case execution time $C_{i,2}$
    - Utilization without SPM $U_{i,1} = C_{i,1}/Ti$
    - Utilization with SPM is $U_{i,2} = C_{i,2}/Ti$
- ***Objective***
  - Select the tasks to be put into the SPM
  - Minimize the required SPM size
  - The utilization of the task set should be no more than 100%

$$\min \quad \sum_{\mathrm{i}} x_i m_i$$

$$\text{s.t.} \quad \sum_{\mathrm{i}} x_i U_{i,2} + \sum_i (1 - x_i) U_{i,1} \leq 1$$

$$x_i \in \{0,1\}, \ \forall \ \mathrm{i}$$

Summary

Integer (linear) programming

- Integer programming is NP-complete
- Linear programming is faster
- Good starting point even if solutions are generated with different techniques

Simulated annealing

- Modeled after cooling of liquids
- Overcomes local minima

Evolutionary algorithms

- Maintain set of solutions
- Include selection, mutation and recombination

Evolutionary Algorithms (1)

- ***Evolutionary Algorithms*** *are based on the collective learning process within a population of individuals, each of which represents a search point in the space of potential solutions to a given problem.*
- *The population is arbitrarily initialized, and it evolves towards better and better regions of the search space by means of randomized processes of*
  - ***selection*** *(which is deterministic in some algorithms),*
  - ***mutation****, and*
  - ***recombination*** *(which is completely omitted in some algorithmic realizations).*

[Bäck, Schwefel, 1993]

Evolutionary Algorithms (2)

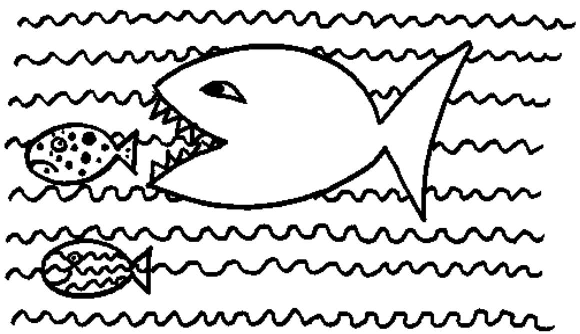
- *The environment (given aim of the search) delivers a quality information (**fitness value**) of the search points, and the selection process favours those individuals of higher fitness to reproduce more often than worse individuals.*
- *The recombination mechanism allows the mixing of parental information while passing it to their descendants, and mutation introduces innovation into the population*

[Bäck, Schwefel, 1993]

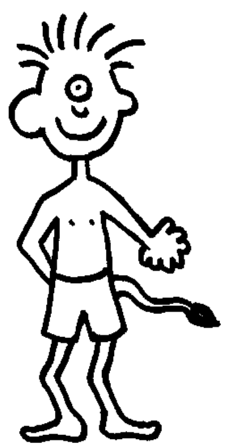
Evolutionary Algorithms

Principles of Evolution

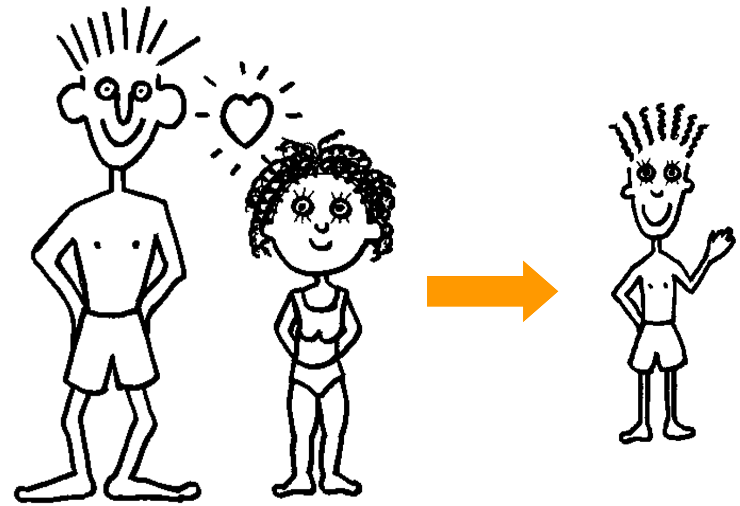
❶ Selection



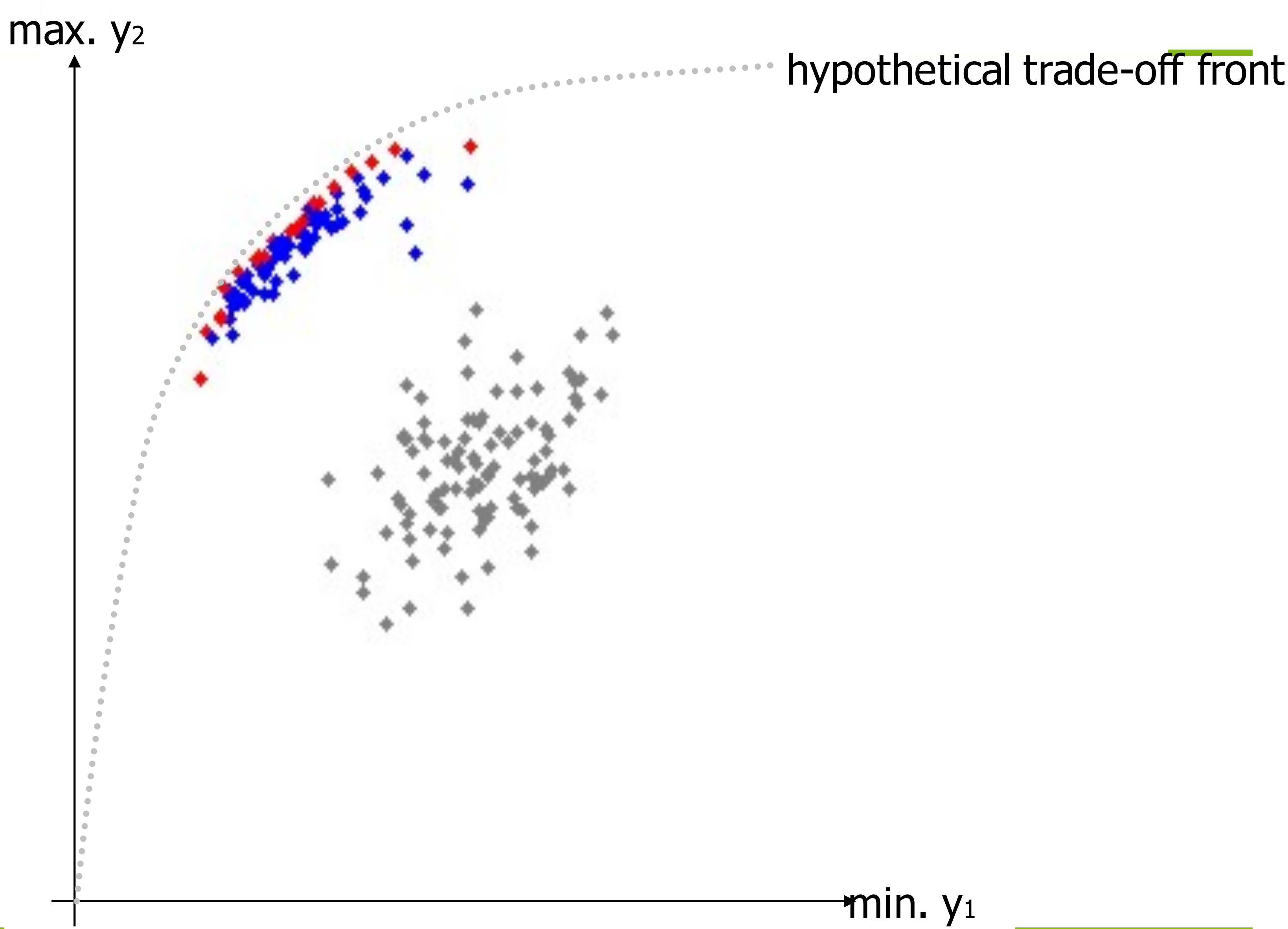
❷ Mutation



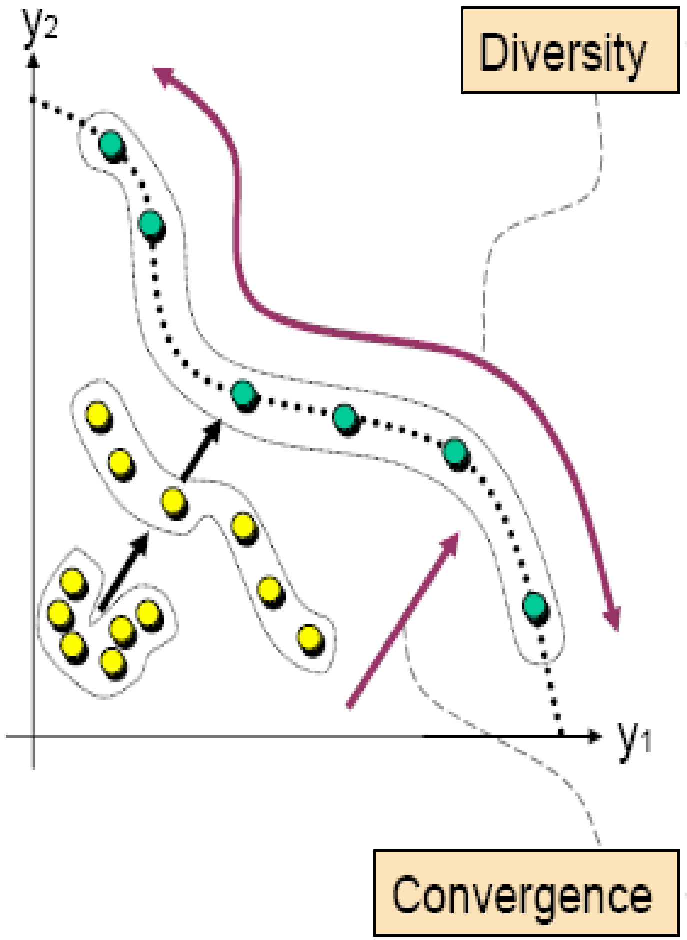
❸ Cross-over



An Evolutionary Algorithm in Action



Issues in Multi-Objective Optimization



- How to maintain a diverse Pareto set approximation?

  ❷ **density estimation**

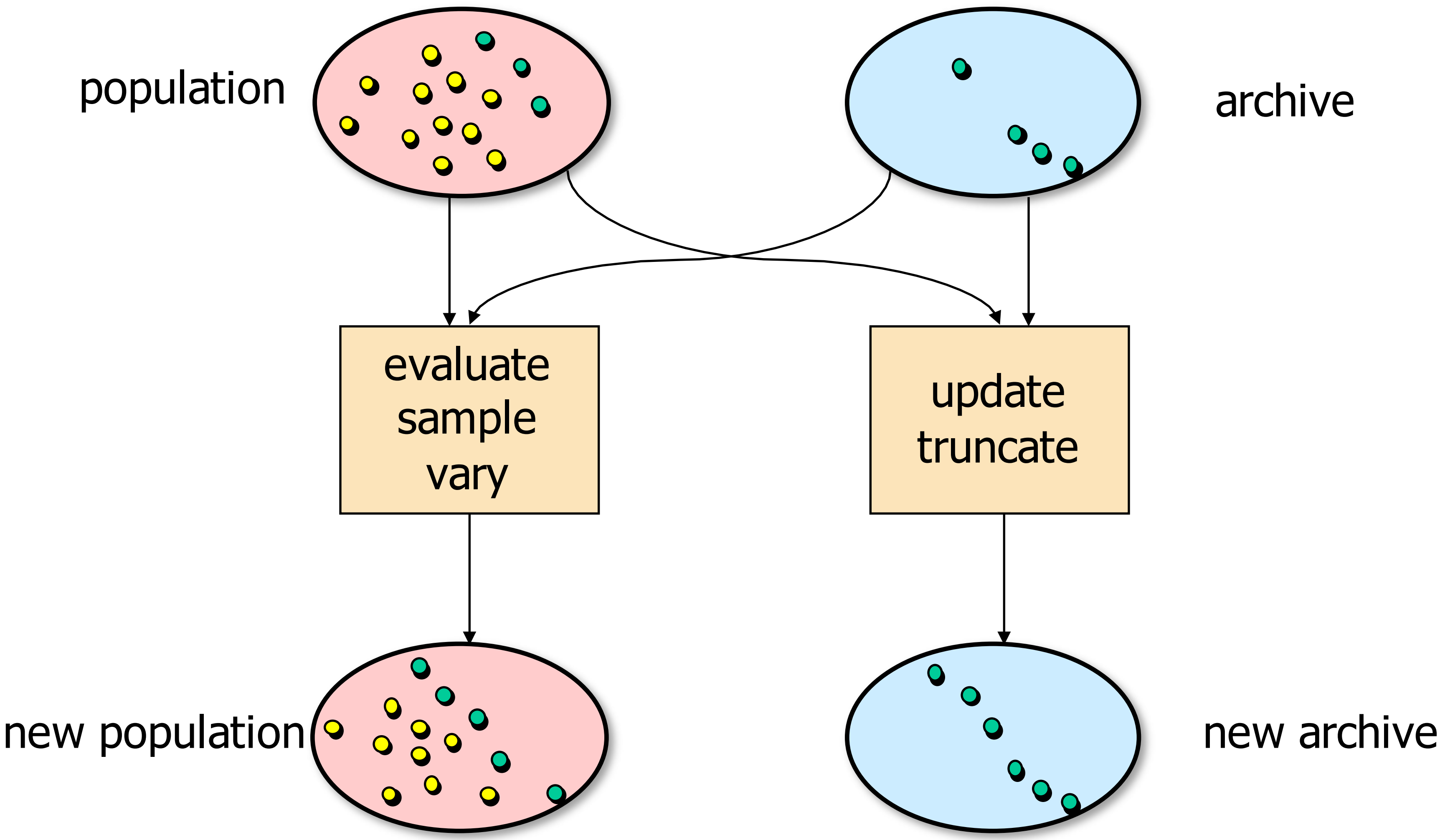
- How to prevent nondominated solutions from being lost?

  ❸ **environmental selection**

- How to guide the population towards the Pareto set?

  ❶ **fitness assignment**

A Generic Multiobjective EA



Example: SPEA2 Algorithm

Step 1:	Generate initial population P0 and empty archive (external set) A_0. Set $t = 0$.
Step 2:	Calculate fitness values of individuals in P_t and A_t.
Step 3:	A_{t+1} = nondominated individuals in P_t and A_t. If size of $A_{t+1} > N$ then reduce A_{t+1}, else if size of $A_{t+1} < N$ then fill A_{t+1} with dominated individuals in P_t and A_t.
Step 4:	If $t > T$ then output the nondominated set of A_{t+1}. Stop.
Step 5:	Fill mating pool by binary tournament selection.
Step 6:	Apply recombination and mutation operators to the mating pool and set P_{t+1} to the resulting population. Set $t = t + 1$ and go to Step 2.

Simulated Annealing

- General method for solving combinatorial optimization problems.
- Based the model of slowly cooling crystal liquids.
- Some configuration is subject to changes.
- Special property of Simulated annealing: Changes leading to a poorer configuration (with respect to some cost function) are accepted with a certain probability.
- This probability is controlled by a temperature parameter: the probability is smaller for smaller temperatures.

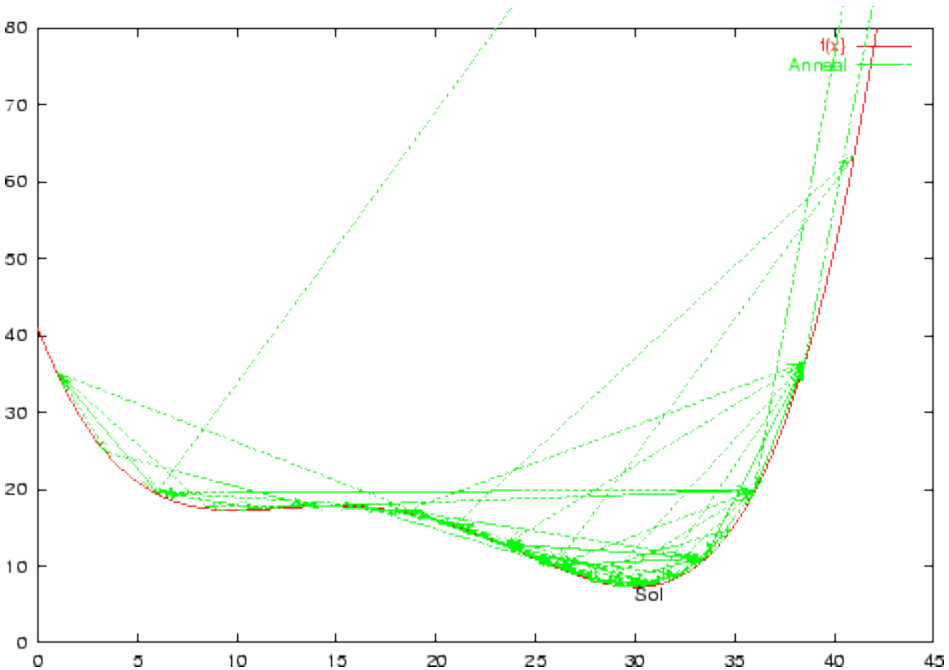
Simulated Annealing Algorithm

```
procedure SimulatedAnnealing;
var i, T: integer;
 begin
  i := 0; T := MaxT;
  configuration:= <some initial configuration>;
  while not terminate(i, T) do
   begin
    while InnerLoop do
      begin NewConfig := variation(configuration);
        delta := evaluation(NewConfig,configuration);
        if delta < 0
        then configuration := NewConfig;
        else if SmallEnough(delta, T, random(0,1))
          then configuration := Newconfiguration;
      end;
  T:= NewT(i,T); i:=i+1;
 end; end;
```

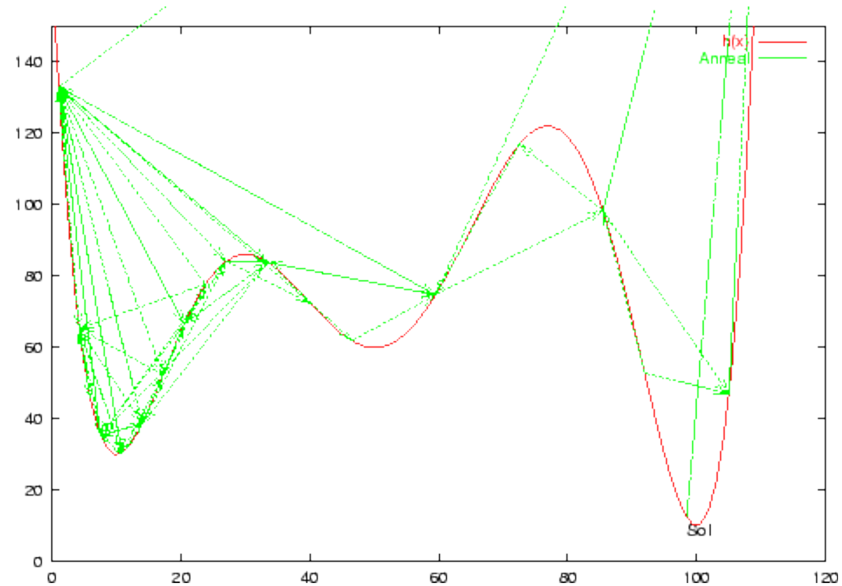
Explanation

- Initially, some random initial configuration is created.
- Current temperature is set to a large value.
- Outer loop:
  - Temperature is reduced for each iteration
  - Terminated if (temperature ≤ lower limit) or (number of iterations ≥ upper limit).
- Inner loop: For each iteration:
  - New configuration generated from current configuration
  - Accepted if (new cost ≤ cost of current configuration)
  - Accepted with temperature-dependent probability if (cost of new config. > cost of current configuration).

Behavior for actual functions



130 steps



200 steps

[people.equars.com/~marco/poli/phd/node57.html]

http://foghorn.cadlab.lafayette.edu/cadapplets/fp/fpIntro.html

Performance

- This class of algorithms has been shown to outperform others in certain cases [Wegener, 2005].
- Demonstrated its excellent results in the TimberWolf layout generation package [Sechen]
- Many other applications …

Summary

Integer (linear) programming

- Integer programming is NP-complete
- Linear programming is faster
- Good starting point even if solutions are generated with different techniques

Simulated annealing

- Modeled after cooling of liquids
- Overcomes local minima

Evolutionary algorithms

- Maintain set of solutions
- Include selection, mutation and recombination