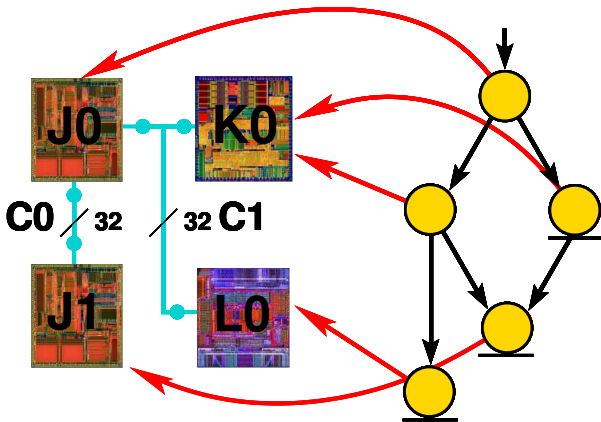




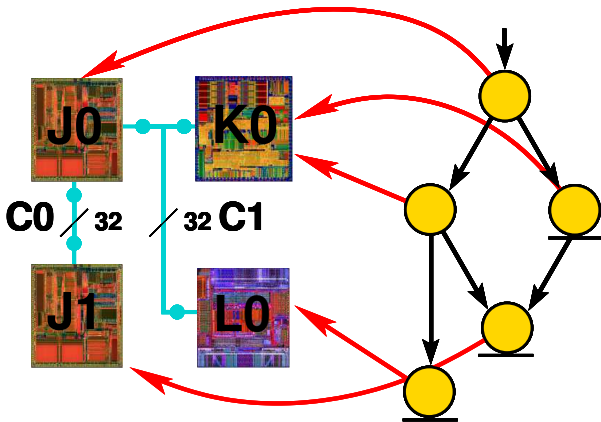
## Synthesis motivation

- Embedded systems are found everywhere: cars, houses, games, phones, hospitals, etc.
- Designers need tools to deal with increasing complexity, increase product quality, and guarantee correct operation.
- Software or hardware errors are not acceptable. Anti-lock brake systems aren't allowed to crash.
- Embedded systems should not require bug fixes or upgrades.
- Price competition can be intense.
- Power consumption should be low.

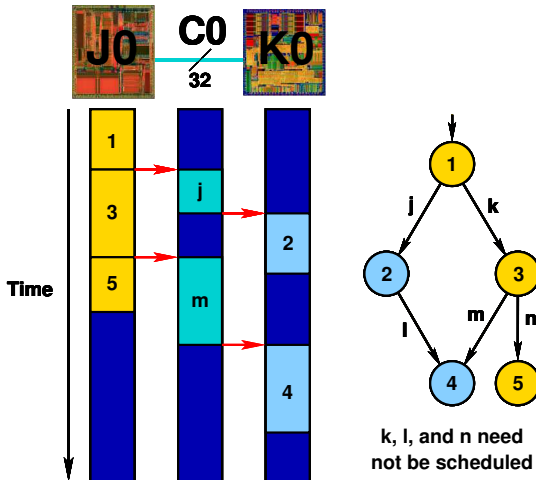
# Allocation



# Assignment

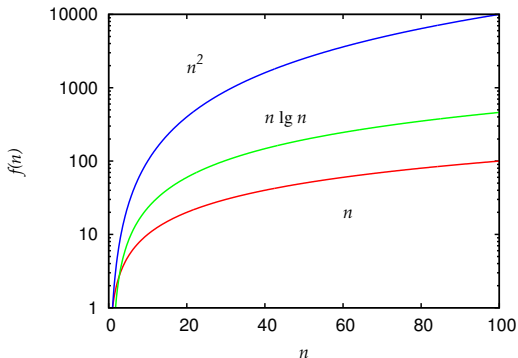


# Scheduling



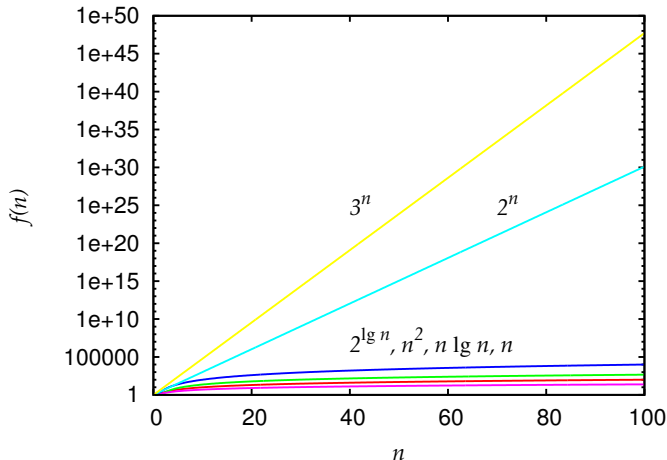
# Polynomial time complexities

- Recall that sorting may be done in  $\mathcal{O}(n \lg n)$  time
- DFS  $\in \mathcal{O}(|V| + |E|)$ , BFS  $\in \mathcal{O}(|V|)$
- Topological sort  $\in \mathcal{O}(|V| + |E|)$



# Exponential time complexities

There also exist exponential-time algorithms:  $\mathcal{O}(2^{\lg n})$ ,  $\mathcal{O}(2^n)$ ,  $\mathcal{O}(3^n)$



## Implications of exponential time complexity

For  $t(n) = 2^n$  seconds

$$t(1) = 2 \text{ seconds}$$

$$t(10) = 17 \text{ minutes}$$

$$t(20) = 12 \text{ days}$$

$$t(50) = 35,702,052 \text{ years}$$

$$t(100) = 40,196,936,841,331,500,000,000 \text{ years}$$



## $\mathcal{NP}$ -complete problems

- Digital design and synthesis is full of NP-complete problems
- Graph coloring
- Allocation/assignment
- Scheduling
- Graph partitioning
- Satisfiability (and 3SAT)
- *Covering*
- ... and many more

## Conjecture on hardness of problems

- There is a class of problems,  $\mathcal{NP}$ -complete, for which nobody has found polynomial time solutions
- It is possible to convert between these problems in polynomial time
- Thus, if it is possible to solve any problem in  $\mathcal{NP}$ -complete in polynomial time, all can be solved in polynomial time
- $\mathcal{P} \subseteq \mathcal{NP}$
- Unproven conjecture:  $\mathcal{P} \neq \mathcal{NP}$

$\mathcal{NP}$ 

- What is  $\mathcal{NP}$ ? Nondeterministic polynomial time.
- A computer that can simultaneously follow multiple paths in a solution space exploration tree is nondeterministic. Such a computer can solve  $\mathcal{NP}$  problems in polynomial time.
- Nobody has been able to prove either

$$\mathcal{P} \neq \mathcal{NP}$$

or

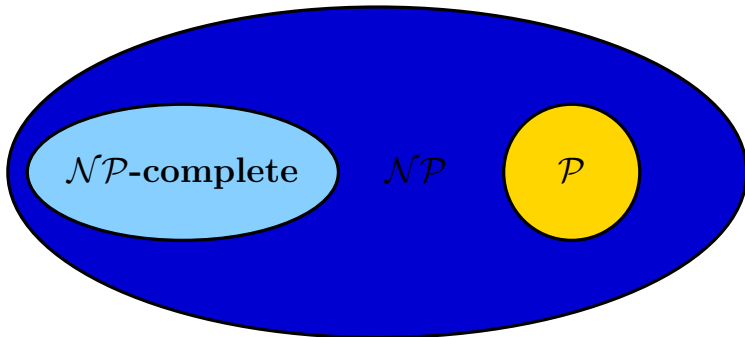
$$\mathcal{P} = \mathcal{NP}$$

# $\mathcal{NP}$ -completeness

If we define  $\mathcal{NP}$ -complete to be a set of problems in  $\mathcal{NP}$  for which any problem's instance may be converted to an instance of another problem in  $\mathcal{NP}$ -complete in polynomial time, then

$$\mathcal{P} \subsetneq \mathcal{NP} \Rightarrow \mathcal{NP}\text{-complete} \cap \mathcal{P} = \emptyset$$

## Basic complexity classes



- $\mathcal{P}$  solvable in polynomial time by a computer (Turing Machine).
- $\mathcal{NP}$  solvable in polynomial time by a nondeterministic computer.
- $\mathcal{NP}$ -complete converted to other  $\mathcal{NP}$ -complete problems in polynomial time.

## How to deal with hard problems

- What should you do when you encounter an apparently hard problem?
- Is it in  $\mathcal{NP}$ -complete?
- If not, solve it
- If so, then what?

Despair. Solve it! Resort to a suboptimal heuristic.

Bad, but sometimes the only choice. Develop an approximation algorithm.

Better. Determine whether all encountered problem instances are constrained.

Wonderful when it works.

## One example

O. Coudert. [Exact coloring of real-life graphs is easy.](#)  
*Design Automation*, pages 121–126, June 1997.

## Properties of complete optimization techniques

- If a solution exists, will be found
- Very slow for some problems
- Good formal understanding of complexity



## Example complete algorithms

- Enumeration
- Branch and bound
- Dynamic programming
- Integer-linear programming
- Backtracking iterative improvement

# Enumeration

- Considers all possible solutions
- Extremely slow for large  $n$
- Potentially has low constant factor, may be O.K. for small  $n$

## Example problem

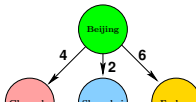
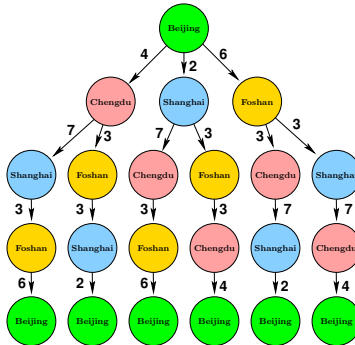
### Traveling salesman problem

Find shortest path visiting all cities.

# Traveling salesman problem



# Enumeration



# Branch and bound

- Keep track of minimal encountered cost
- When a path has a higher cost, terminate

# Branch and bound



**path=0**

# Branch and bound

- Better average-case complexity
- Still worst-case exponential



# Linear programming

- In  $\mathcal{P}$ - Ellipsoid Algorithm / internal point methods
- However, in practice WC exponential Simplex Algorithm better
- Goal: Maximize a linear weighted sum under constraints on variables

# Linear programming

Maximize

$$c_1 \cdot x_1 + c_2 \cdot x_2 + \cdots + c_n \cdot x_n$$

where

$$\forall c_i \in c, c_i \in R$$

subject to the following constraints:

$$a_{11} \cdot x_1 + a_{12} \cdot x_2 + \cdots + a_{1n} \cdot x_n \leq, =, \geq b_1$$

$$a_{21} \cdot x_1 + a_{22} \cdot x_2 + \cdots + a_{2n} \cdot x_n \leq, =, \geq b_2$$

.....

$$a_{n1} \cdot x_1 + a_{n2} \cdot x_2 + \cdots + a_{nn} \cdot x_n \leq, =, \geq b_n$$

$$\forall x_i \in x, x_i \geq 0 \qquad \forall a_{jk} \in A, a_{jk} \in R$$

# Linear programming

- Can be formulated as a linear algebra problem
  - Vector  $x$  of variables
  - Vector  $c$  of cost
  - Matrix  $A$  of constraints
  - Vector  $b$  of constraints
- Maximize or minimize  $c^T x$
- Satisfy  $Ax \leq b$
- Satisfy  $x \geq 0$

# Integer-linear programming (ILP)

- ILP is  $\mathcal{NP}$ -complete
- LP with some variables restricted to integer values
- Formulate problem as ILP problem
  - Excellent understanding of problem
  - Good solvers exist
- Variants – both NP-complete
  - Mixed ILP has some continuous variables
  - Zero-one ILP

# Example – ILP formulation for the traveling salesman problem

Let  $T$  be a tentative solution, or tour

$\forall e \in E$  let there be a variable

$$t_e = \begin{cases} 1 & \text{if } e \in T \\ 0 & \text{if } e \notin T \end{cases}$$

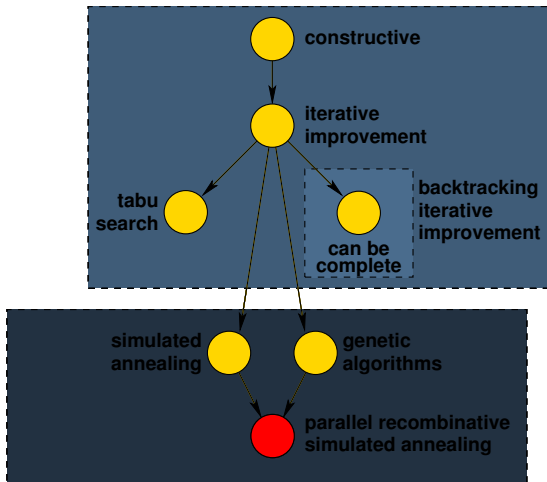
Constraint: Given that  $S$  is a set of vertices,  $\mathbf{con}(S)$  is the set of edges connecting  $v \in S$  to  $v \notin S$ , and  $\{v_i\}$  is the vertex set containing only  $v_i$ , every vertex,  $v_i$  must be connected to two edges of the tour

$$\forall v_i \in V, \quad \sum_{e \in \mathbf{con}(\{v_i\})} = 2$$

## Backtracking iterative improvement

- Allows  $B$  steps of backtracking
- Can be incomplete
- Complete if  $B =$  the problem decision depth
- Allows use of problem-specific heuristics for ordering
- Incomplete if  $B <$  decision depth
- More on this later

# Optimization techniques

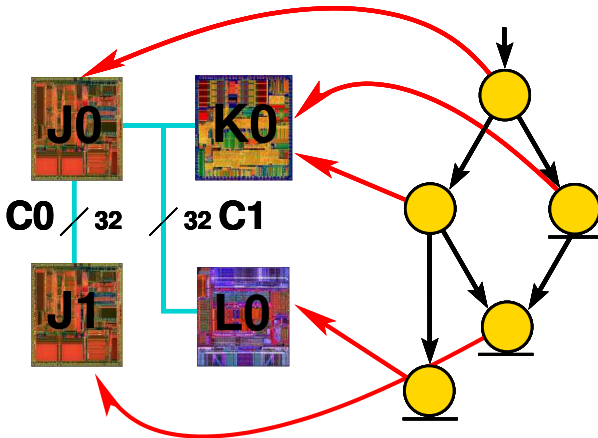


# Constructive algorithms

- Build solution piece by piece
- Once complete solution is generated, don't change
- Typically fast
- Easy to use problem-specific information
- Easy to implement
- Prone to becoming trapped in poor search space



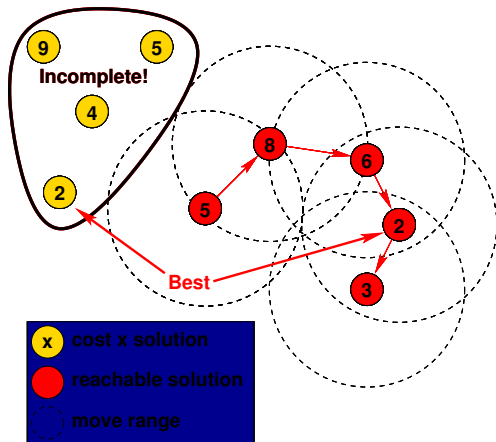
# Constructive algorithms example



# Iterative improvement

- Starts with complete but poor solution
  - therefore contains constructive algorithm
  - superset of constructive
- Makes changes to solution to improve it
- Typically fast
- Easy to use problem-specific information
- Easy to implement
- Prone to becoming trapped in **local minima**

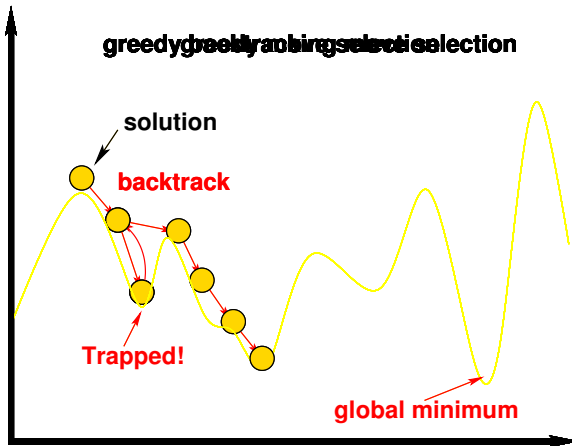
## Local minima move size



## Local minima

- Even if all solutions reachable, may not get best solution
- Depends on move selection

# Trapped in local minima



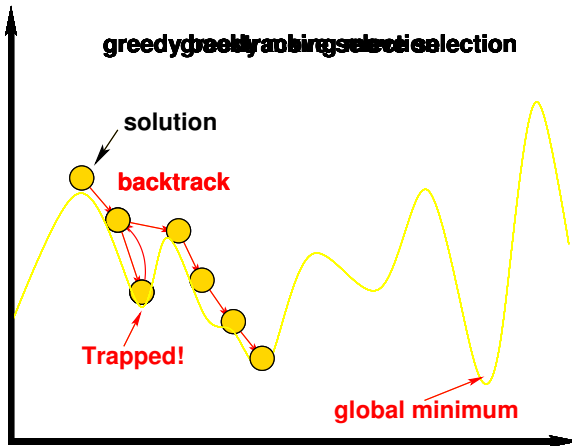
# Local minima

- Being trapped in local minima is a big problem
- Numerous probabilistic optimization techniques designed
  - avoid local minima
  - find global minima
  - do so efficiently

## Backtracking iterative improvement

- Backtracking iterative improvement is complete if
  - all solutions are reachable
  - the backtracking depth  $\geq$  search depth
  - ... however, this can be slow
- Even if incomplete, backtracking can improve quality
- Can trade optimization time for solution quality
- Greedy iterative improvement if backtracking depth is zero

# Backtracking



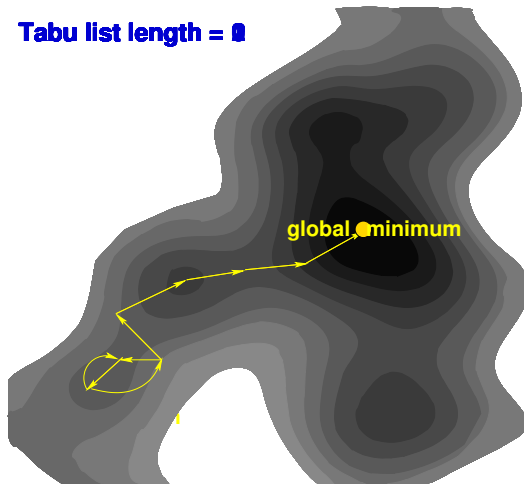


# Tabu search

- Similar to interative improvement
- Iterative improvement can cycle
  - chooses largest cost decrease move. . .
  - . . . then chooses smallest cost increase move
- Tabu search has a *tabu list*
  - solutions to avoid
  - solution characteristics to avoid
- Prevents iterative cycles

## Tabu search example

Tabu list length = 0



# Simulated annealing

- Inspired by annealing of metals
- Start from high temperature and gradually lower
- Avoids local minima traps
- Generate trial solutions
- Conduct Boltzmann trials between old and new solution

# Simulated annealing

- Easy to implement
- Can trade optimization time for solutions quality
- Greedy iterative improvement if temperature is zero
- Famous for solving difficult physical problems, e.g., placement

# Boltzmann trials

Solutions are selected for survival by conducting Boltzmann trials between parents and children.

Given a global temperature  $T$ , a solution with cost  $K$  beats a solution with cost  $J$  with probability:

$$\frac{1}{1 + e^{(J-K)/T}}$$

# Boltzmann trials

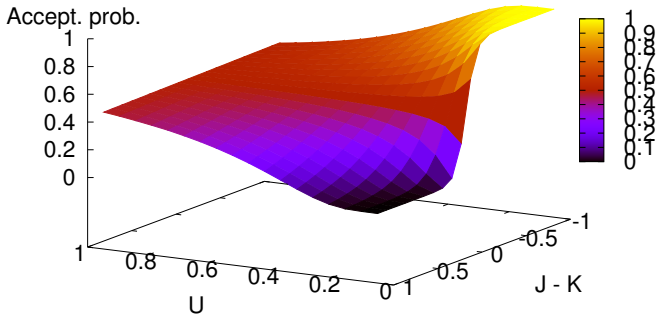
Introduce convenience variable  $U$

$$U(T) = 1 - \frac{1}{T + 1}$$

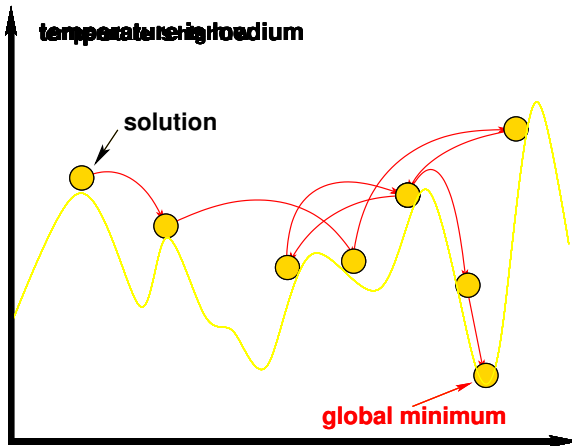
$$U(0) = 0$$

$$T \rightarrow 1 \Rightarrow U(T) \rightarrow \infty$$

# Boltzmann trials

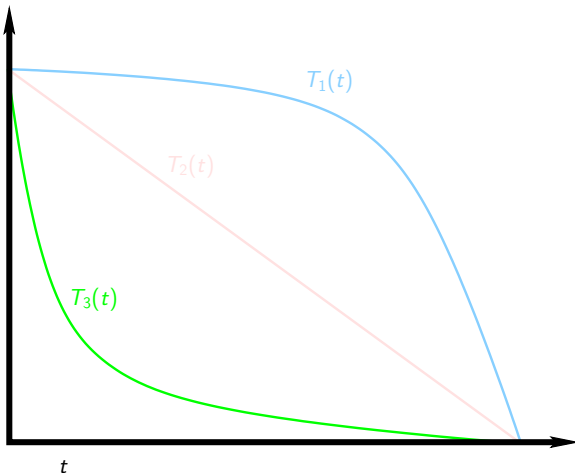


## Simulated annealing example





# Cooling schedule often not important



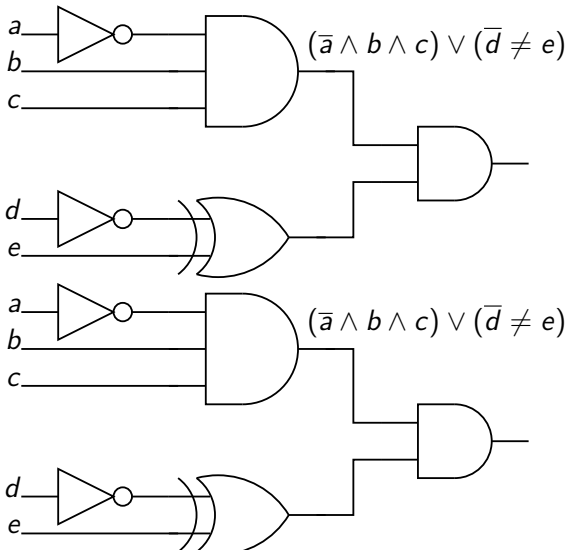
# Simulated annealing notes

- Time complexity extremely difficult to analyze
- Given a slow enough cooling schedule, will get optimum
  - This schedule sometimes makes simulated annealing slower than exhaustive search
  - Determining optimal schedule requires detailed knowledge of problem's Markov chains

# Genetic algorithms

- Multiple solutions
- Local randomized changes to solutions
- Solutions share information with each other
- Can trade optimization time for solution quality
- Good at escaping sub-optimal local minima
- Greedy iterative improvement if no information sharing
- Difficult to implement and analyze
- Researchers have applied in testing, system synthesis

## Solution representation



# Mutation

- Choose an element of the solution
- Change it to another value
- Local modification, similar to that in iterative improvement

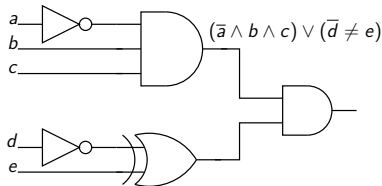
## Solution representation

<i>a</i>	<i>d</i>	<i>b</i>	<i>e</i>	<i>c</i>
0	0	1	0	1
<i>a</i>	<i>d</i>	<i>b</i>	<i>e</i>	<i>c</i>
0	0	1	0	1
<i>a</i>	<i>d</i>	<i>b</i>	<i>e</i>	<i>c</i>
0	0	1	1	1
<i>a</i>	<i>d</i>	<i>b</i>	<i>e</i>	<i>c</i>
0	0	1	1	1

# Crossover

<i>a</i>	<i>d</i>	<i>b</i>	<i>e</i>	<i>c</i>
<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>

# Locality not preserved



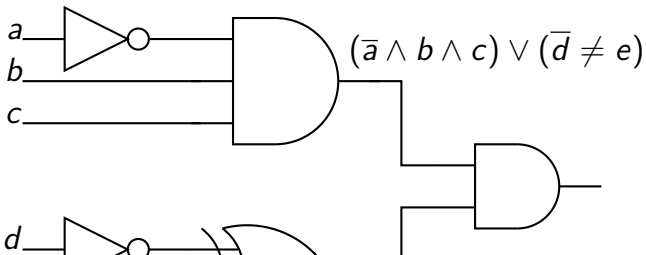
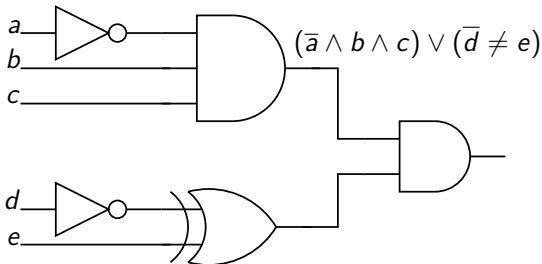
<i>a</i>	<i>d</i>	<i>b</i>	<i>e</i>	<i>c</i>
0	0	0	0	1

0	1	1	1	1
---	---	---	---	---





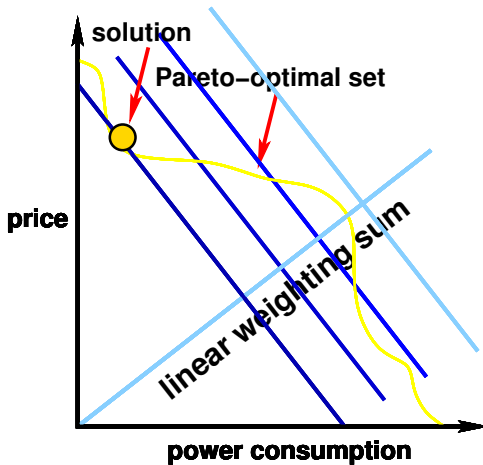
# Locality preserved



# Multidimensional optimization

- Real-world problems often have multiple costs
  - Price
  - Power consumption
  - Speed
  - Temperature
  - Reliability
  - etc.
- Necessary to simultaneously minimize all costs

## Linear weighting sum



# Pareto-ranking

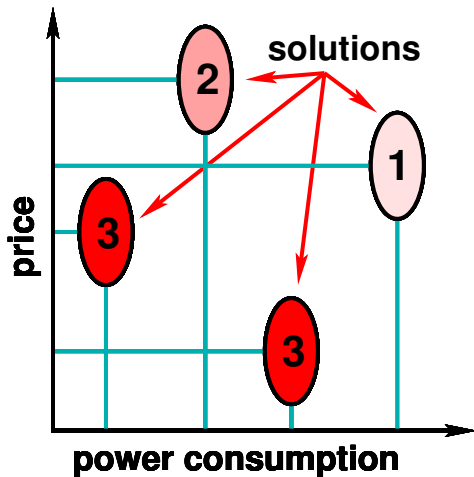
A solution dominates another if all its costs are lower, i.e.,

$$\mathbf{dom}_{a,b} = \forall_{i=1}^n \text{cost}_{a,i} < \text{cost}_{b,i} \wedge a \neq b$$

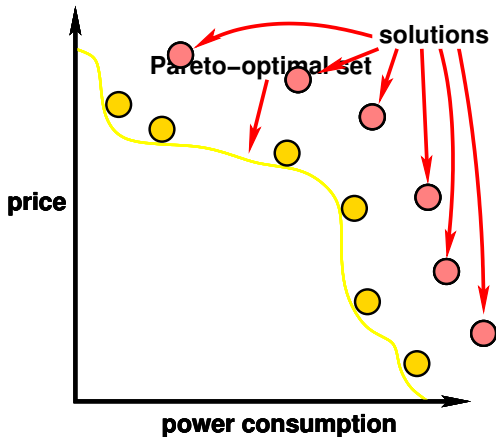
A solution's rank is the number of other solutions which do not dominate it, i.e.,

$$\mathbf{rank}_{s'} = \sum_{i=1}^n \mathbf{not\ dom}_{s_i, s'}$$

## Pareto-ranking



# Pareto-rank based multiobjective optimization



## Genetic algorithm selection

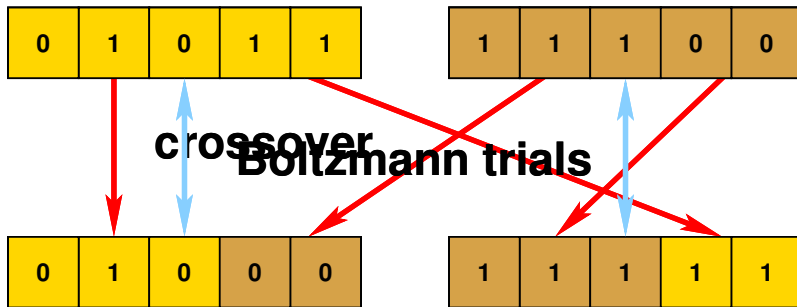
- Solutions are selected for survival by cost or rank
- Resistant to becoming trapped in local minima
  - mutation
  - crossover
- Possible to do better?

# PRSA

- Genetic algorithm where Boltzmann trials are used for solution selection
- Genetic algorithm if temperature is set to zero
- Simulated annealing if only one solution
- Easily parallizable
- Has strengths of genetic algorithms and simulated annealing
- Difficult to implement but not more difficult than genetic algorithms



## PRSA example



## Multiobjective GAs

Carlos M. Fonseca and Peter J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization.

In *Proc. Int. Conf. Genetic Algorithms*, pages 416–423, July 1993

- Explains importance of multiobjective optimization
- Shows simple way to use Pareto-rank in parallel optimization meta-heuristics

## Very high-level optimization reference

Robert P. Dick. *Multiobjective synthesis of low-power real-time distributed embedded systems*.

PhD thesis, Dept. of Electrical Engineering, Princeton University, July 2002

- Chapter 4 contains an overview of some of the popular probabilistic optimization techniques used in CAD
- Chapters 5 and 6 describe a PRSA for system synthesis.

# Evolutionary algorithms

D. Graham-Rowe. [Radio emerges from the electronic soup.](#)  
*New Scientist*, August 2002

- Interesting short article on a physical application on evolutionary algorithms
- Similar results for FPGA-based filter

## Genetic algorithms reference

David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*.

Addison-Wesley, MA, 1989

- The most basic and complete book on genetic algorithms
- Weak on multiobjective potential this meta-heuristic

## PRSA reference

Samir W. Mahfoud and David E. Goldberg. [Parallel recombinative simulated annealing: A genetic algorithm.](#)  
*Parallel Computing*, 21:1–28, January 1995

## What to do by Wednesday evening

Send the following things to me by email

- An itemized list of 1-3 value propositions, i.e., the values you think your embedded system or research idea can provide to your customers.
- A text file containing a paragraph-long description of the embedded system you are currently planning to prototype.
- A text file listing the 2-3 most important hypotheses you are attempting to validate or invalidate via interviews.
- A text file (or files) containing notes from all your interviews. You should have around 10 by Wednesday. Each should contain the following.
  - Date and time.
  - Name of interviewee.
  - Why the interviewee is a potential customer.
  - A chronologically organized series of questions and answers. These can be terse.