EECS 507: Introduction to Embedded Systems Research Optimization

Robert Dick

University of Michigan

Context

Just completed specification and modeling.

Will provide written feedback on paper summaries over weekend, so it can influence your next summary.

I will present embedded system memory hierarchy paper on Tuesday.

Outline

Why learn synthesis and optimization?

Learn how to view design process as cost optimization under constraints.

This is useful even if you never work on automated design.

Learn how to find solutions to very hard problems.

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.

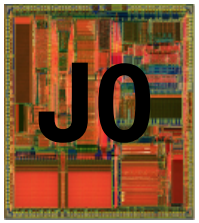
Embedded systems should not require bug fixes or upgrades.

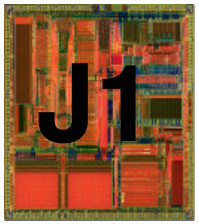
Price competition can be intense.

Power consumption should be low.

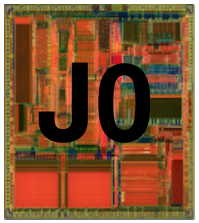
Section outline

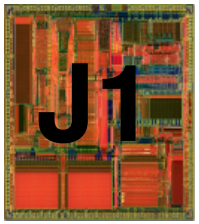
Allocation



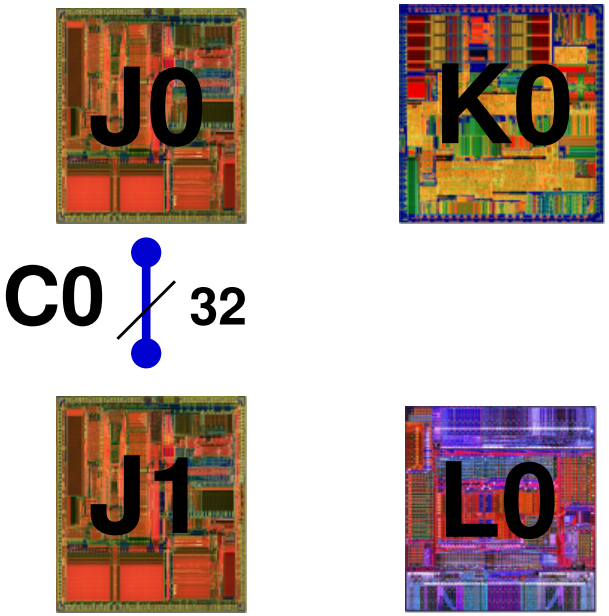


Allocation

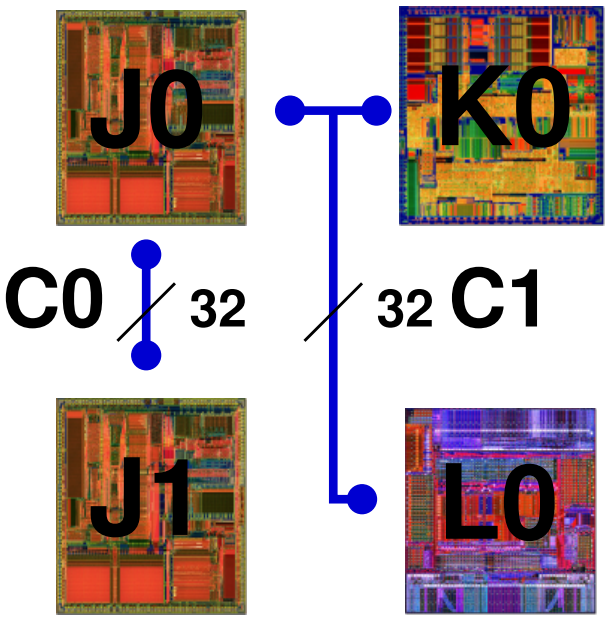




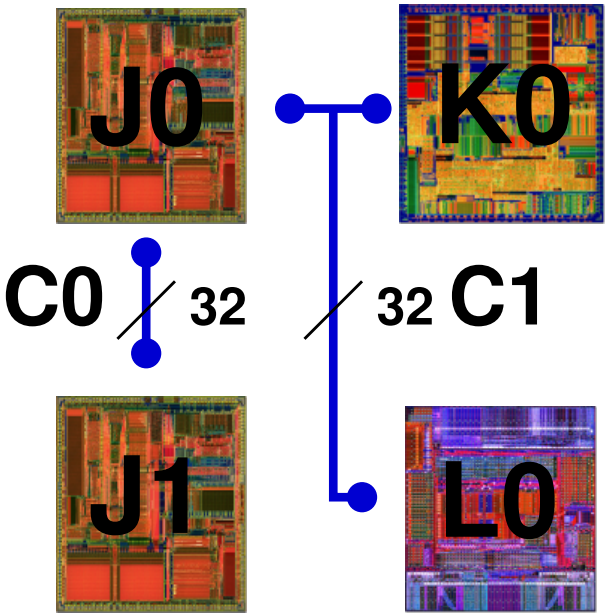
Allocation

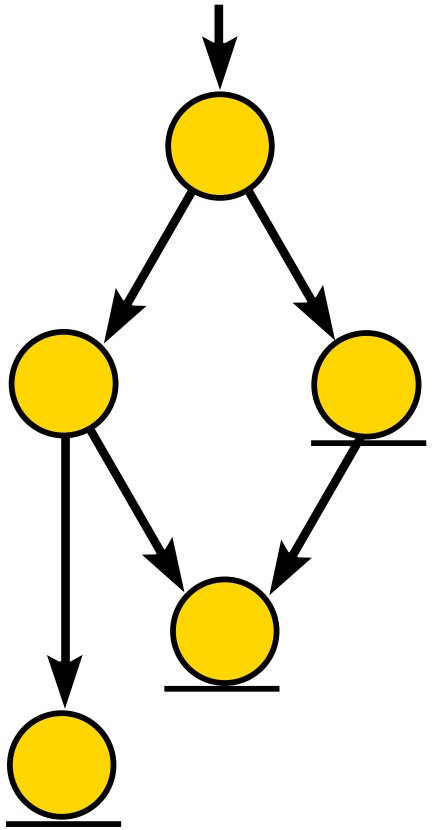


Allocation

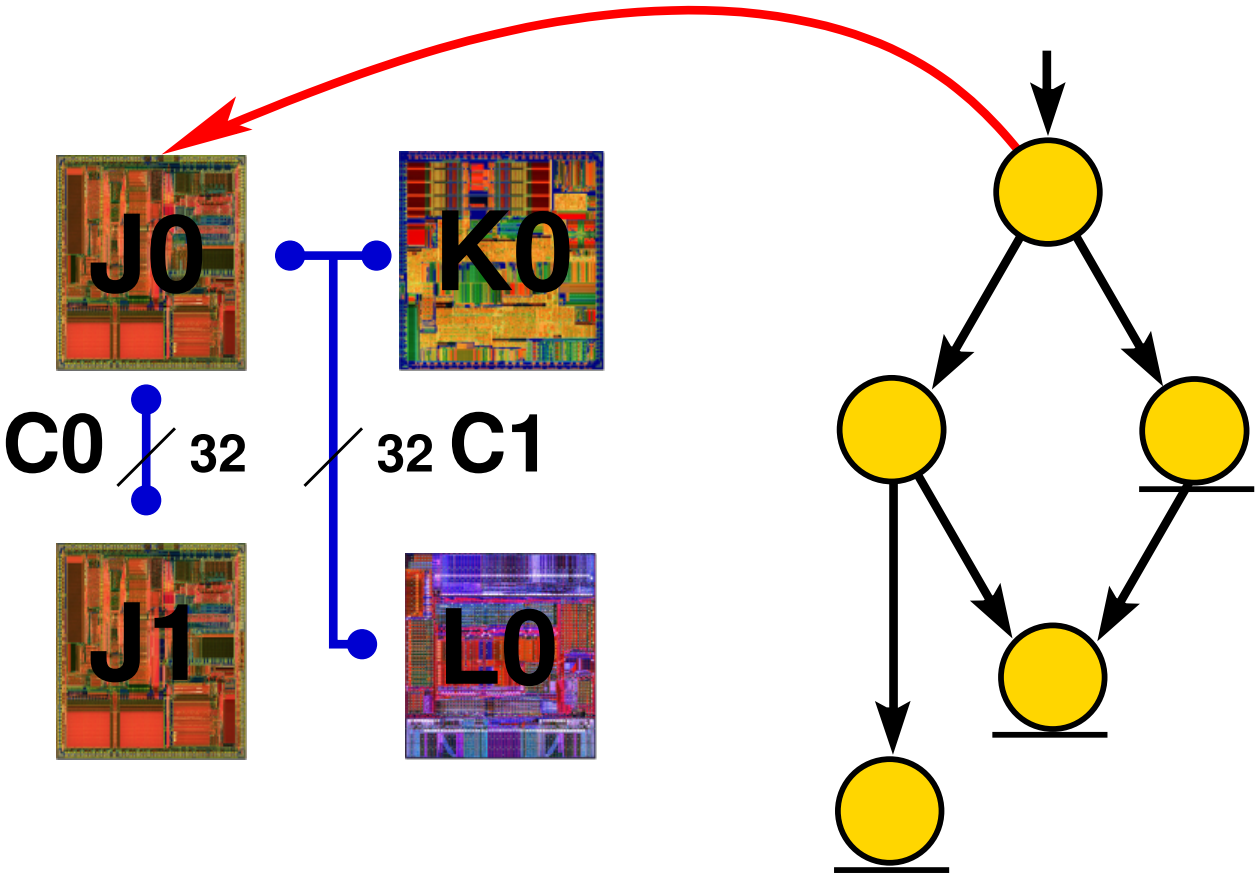


Allocation

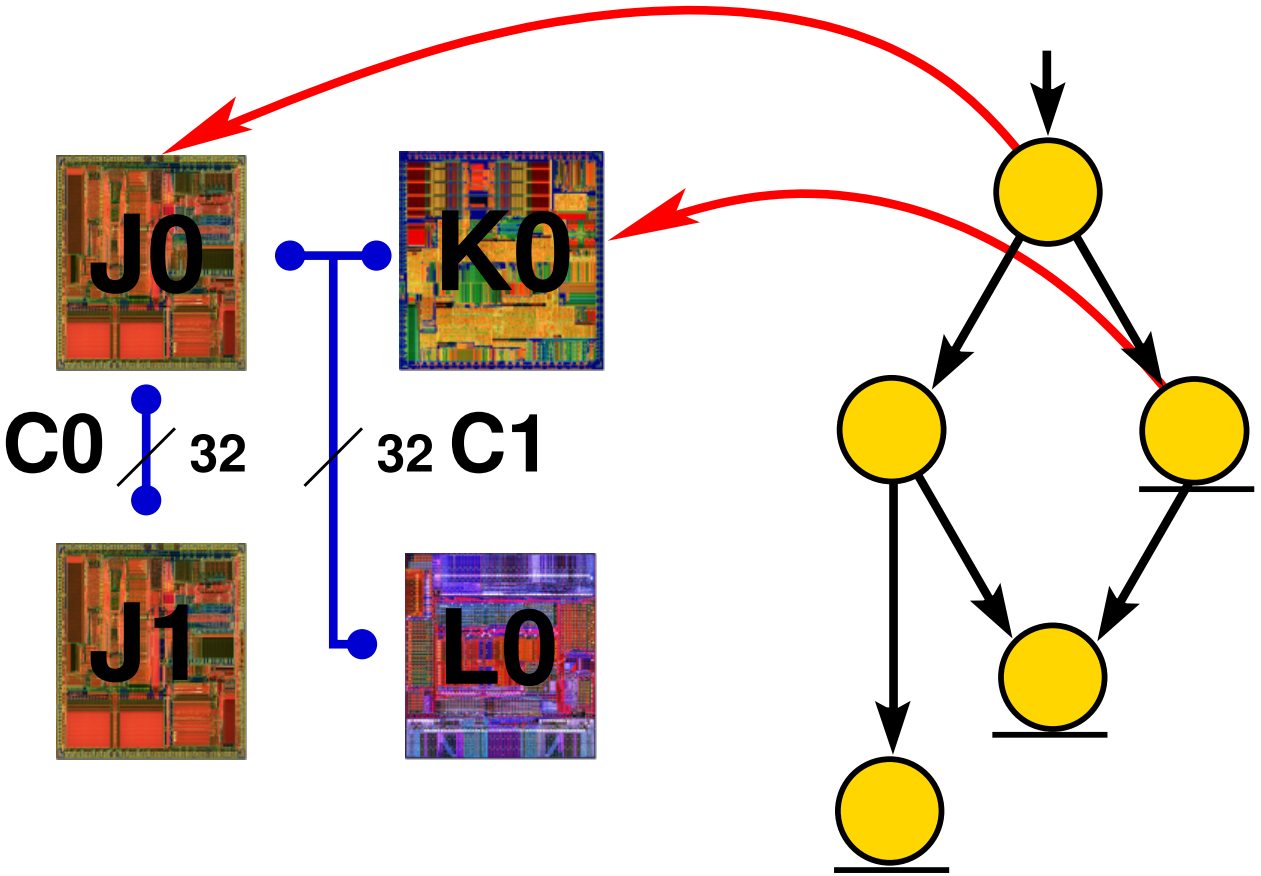




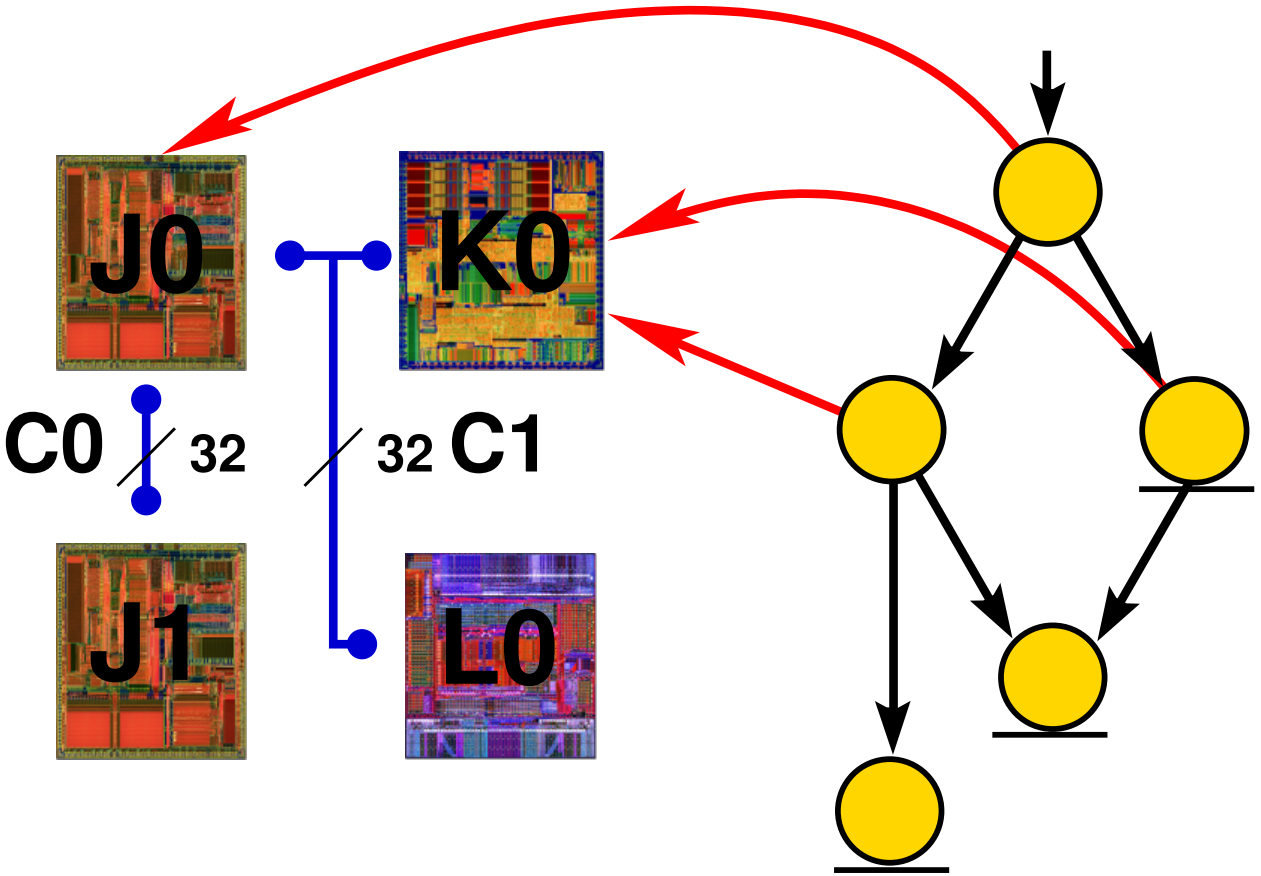
Assignment



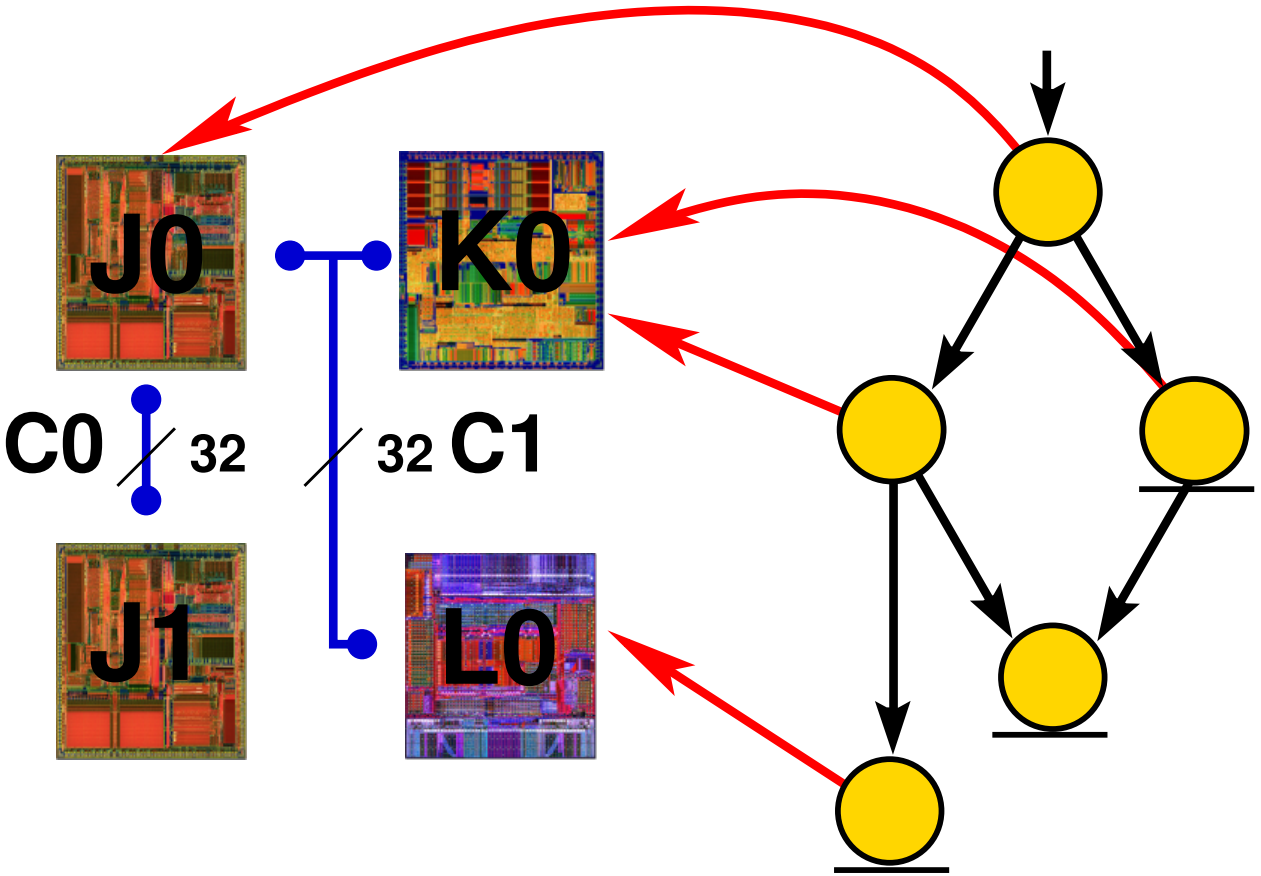
Assignment



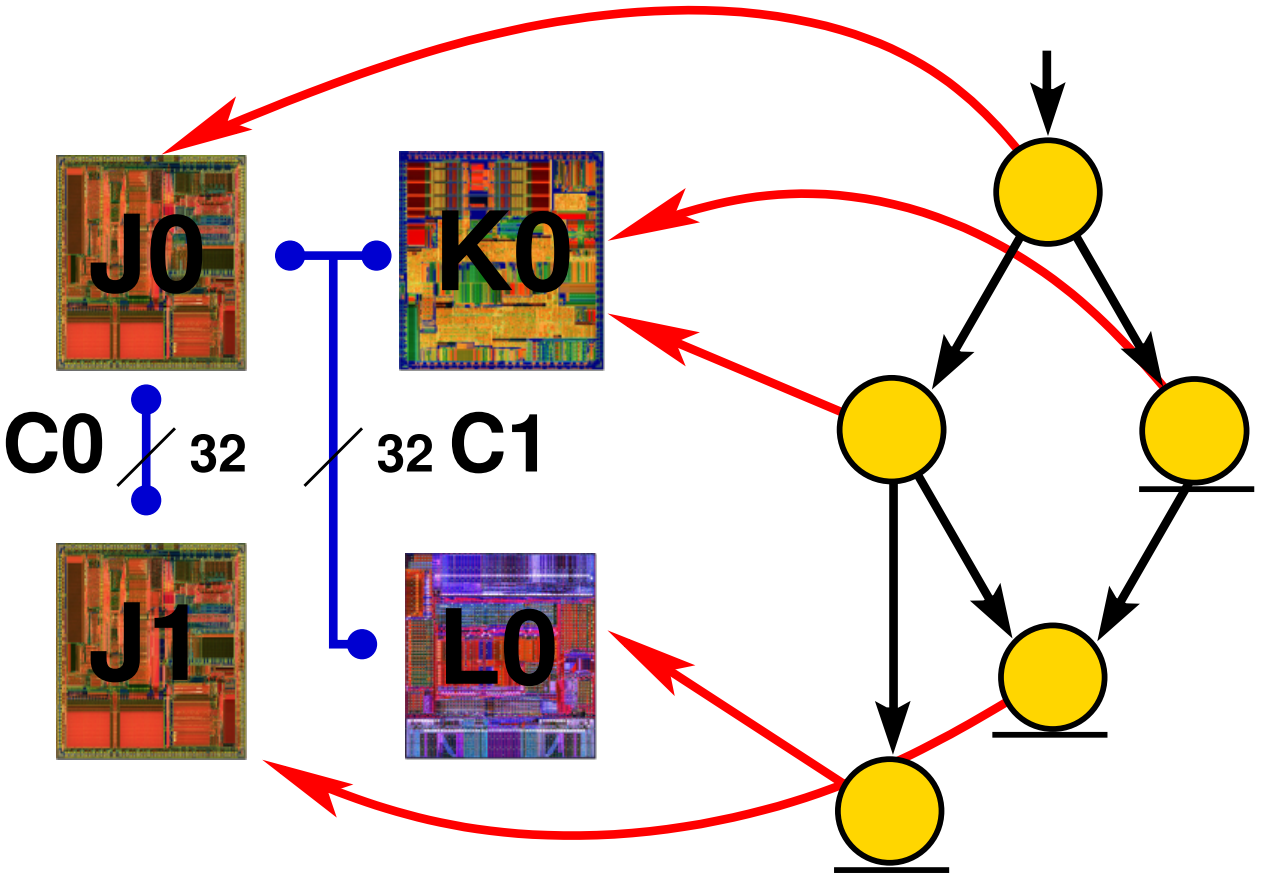
Assignment



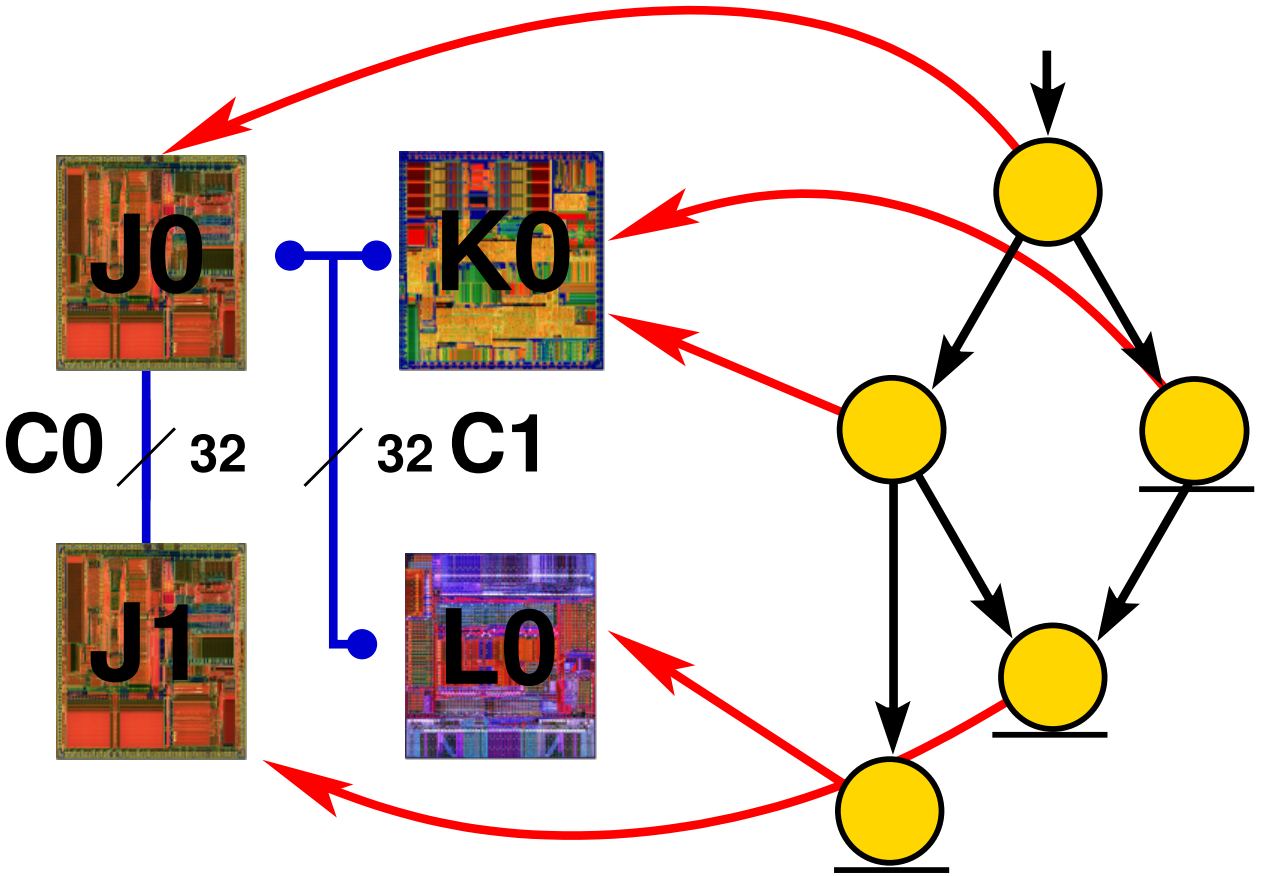
Assignment



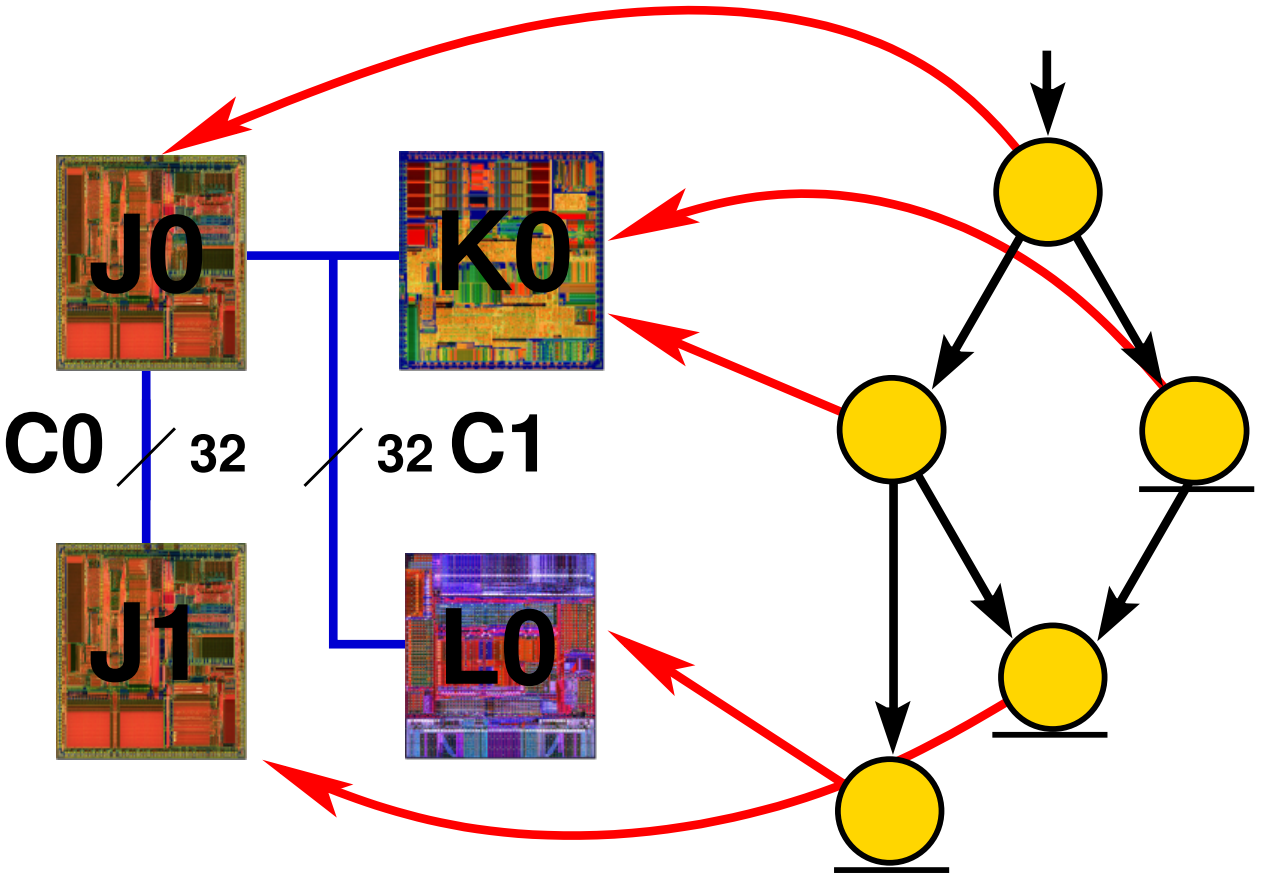
Assignment



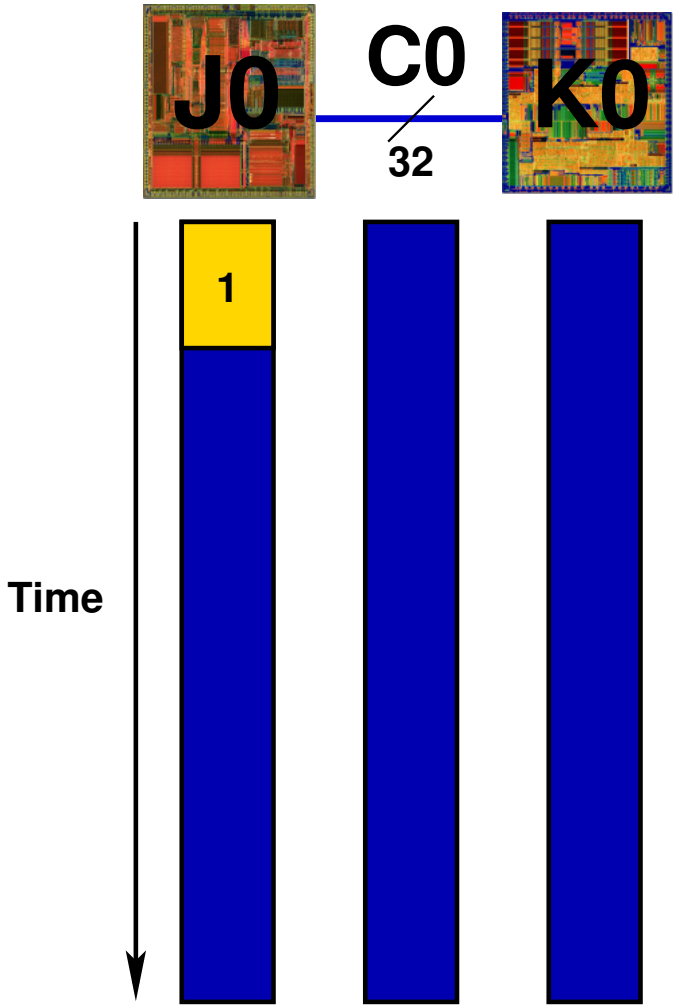
Assignment

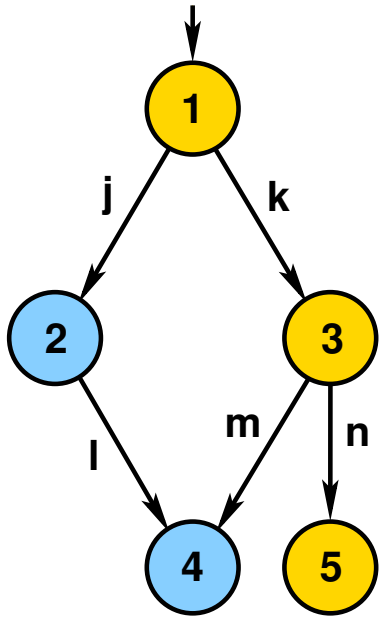


Assignment

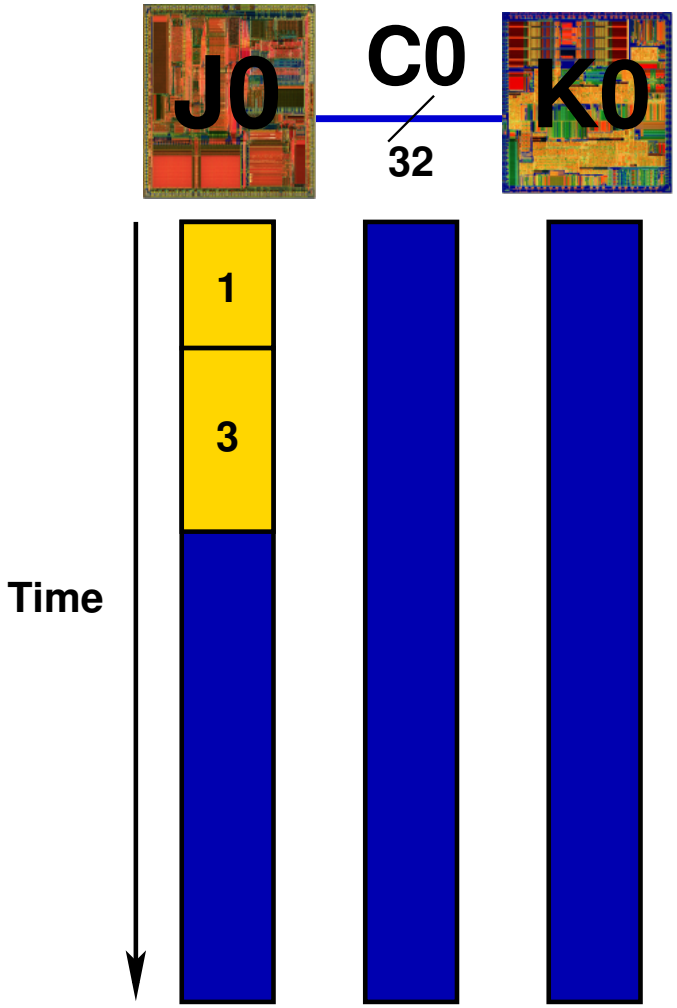


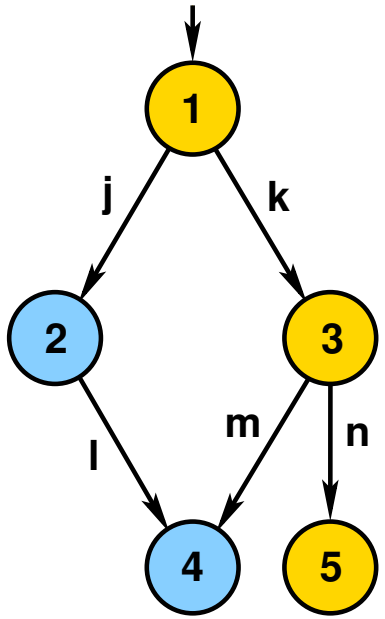
Scheduling



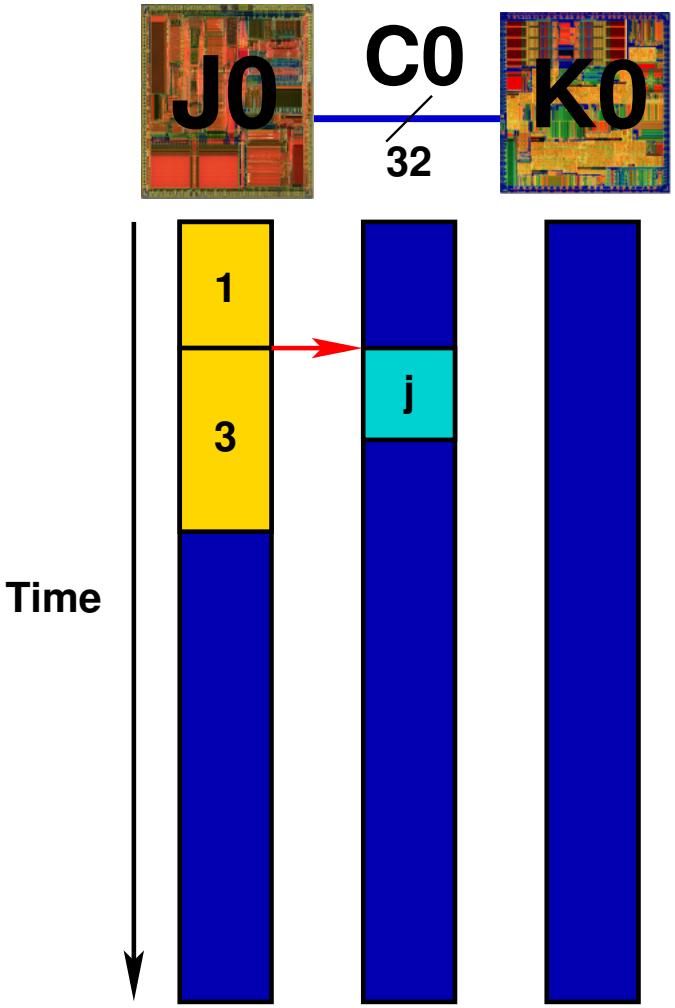


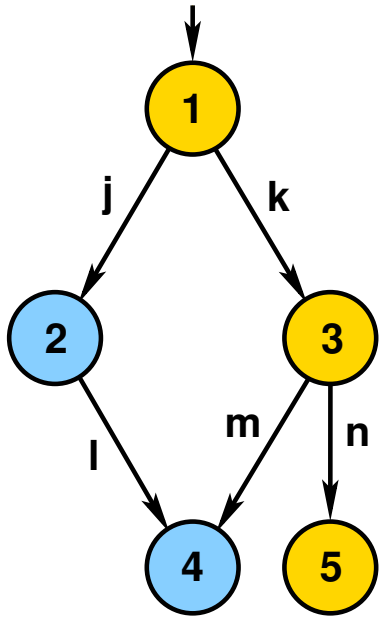
Scheduling



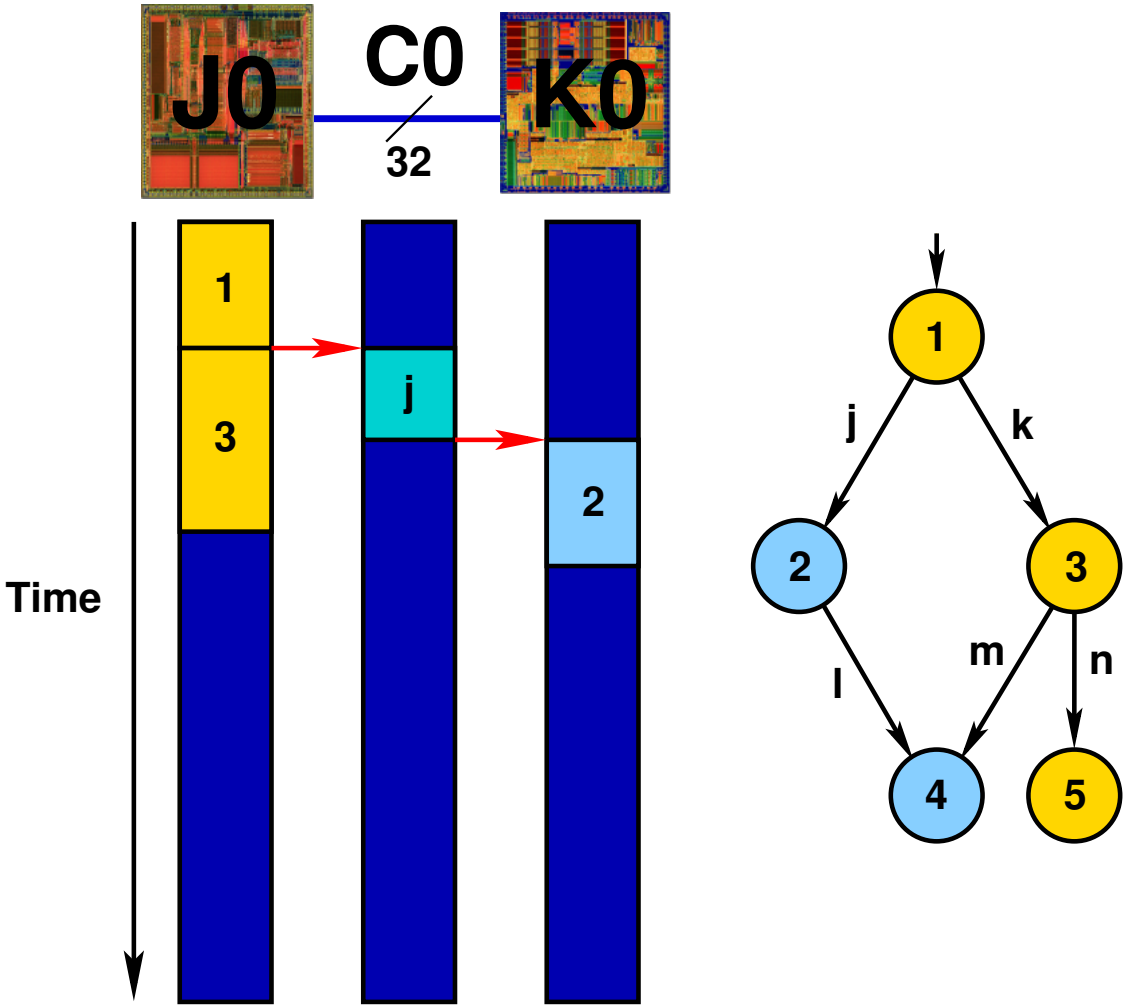


Scheduling

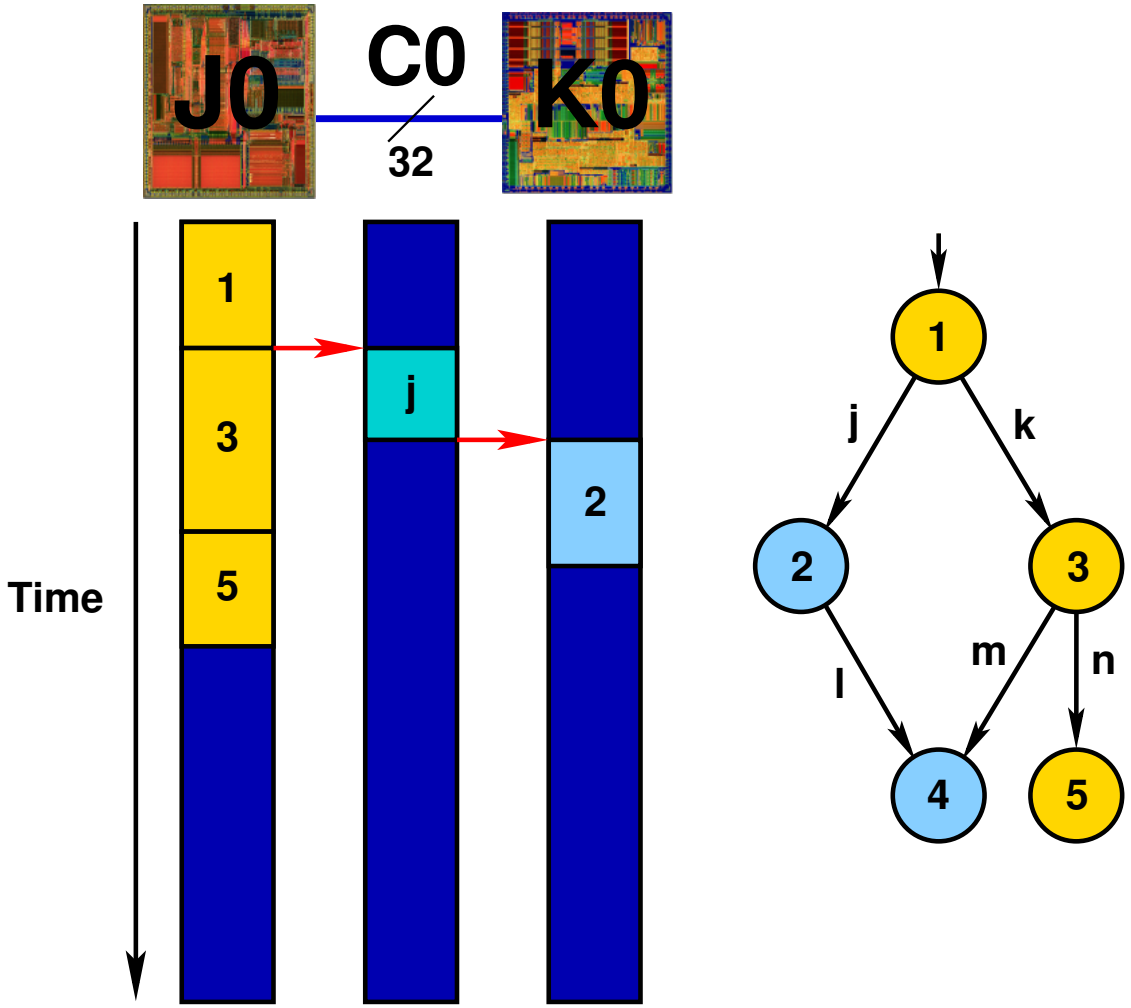




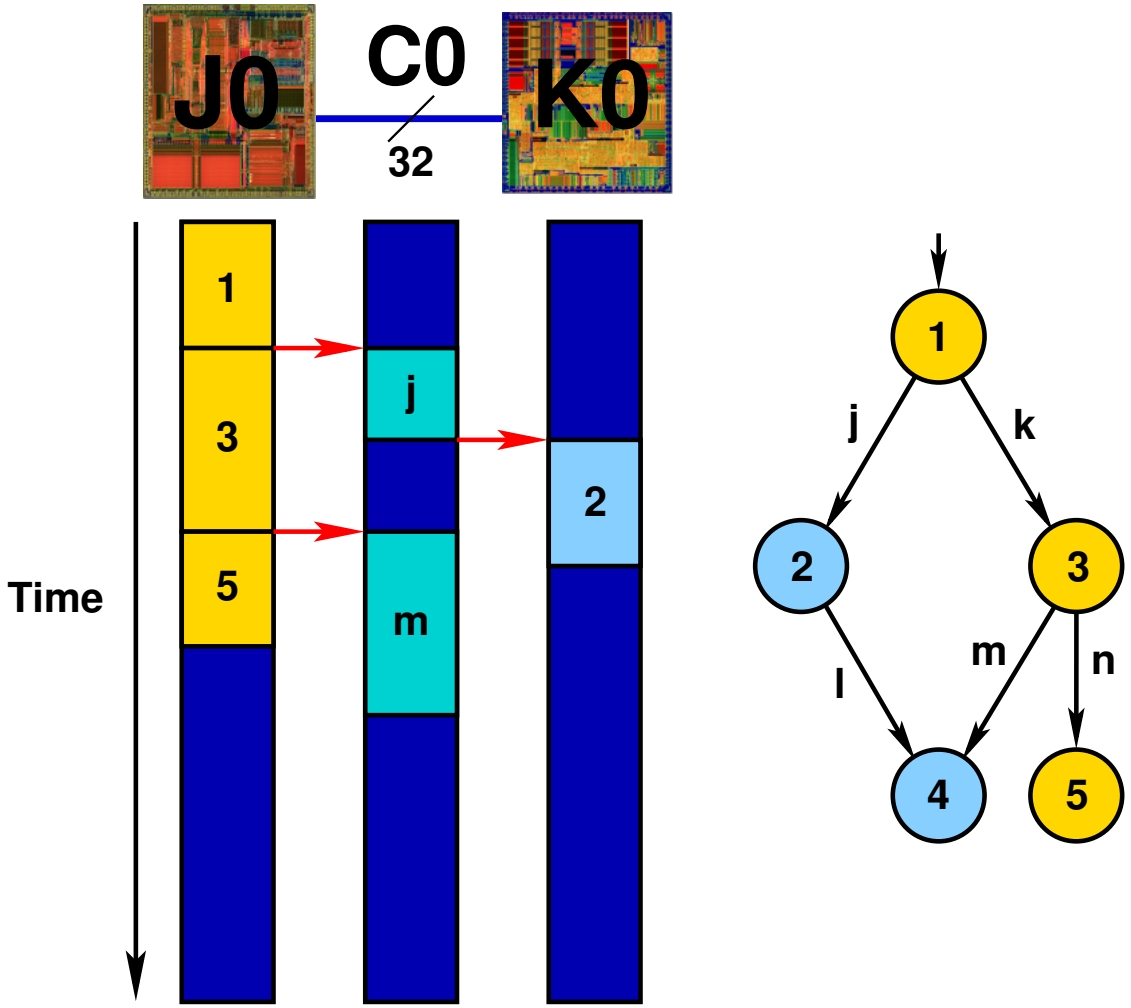
Scheduling



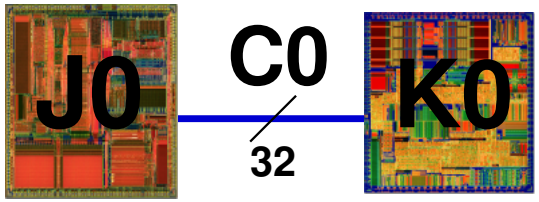
Scheduling

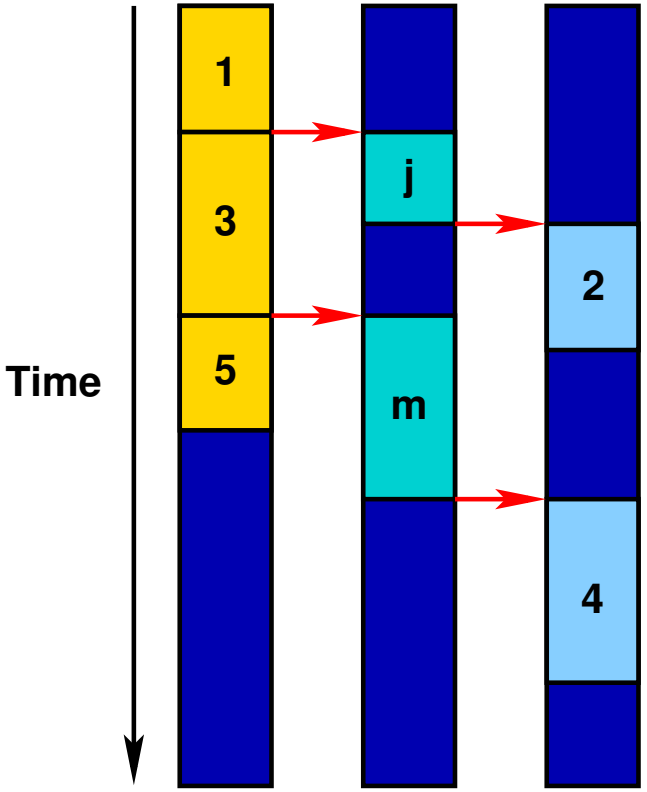


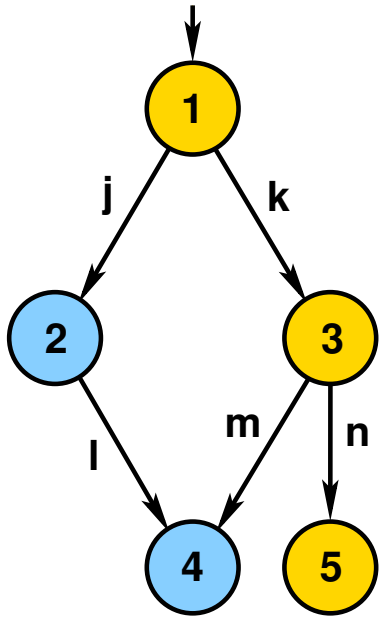
Scheduling



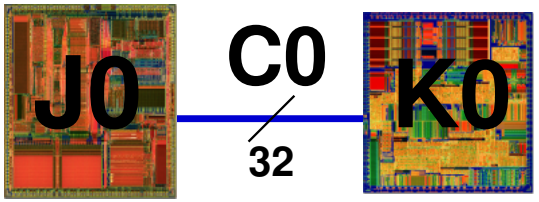
Scheduling

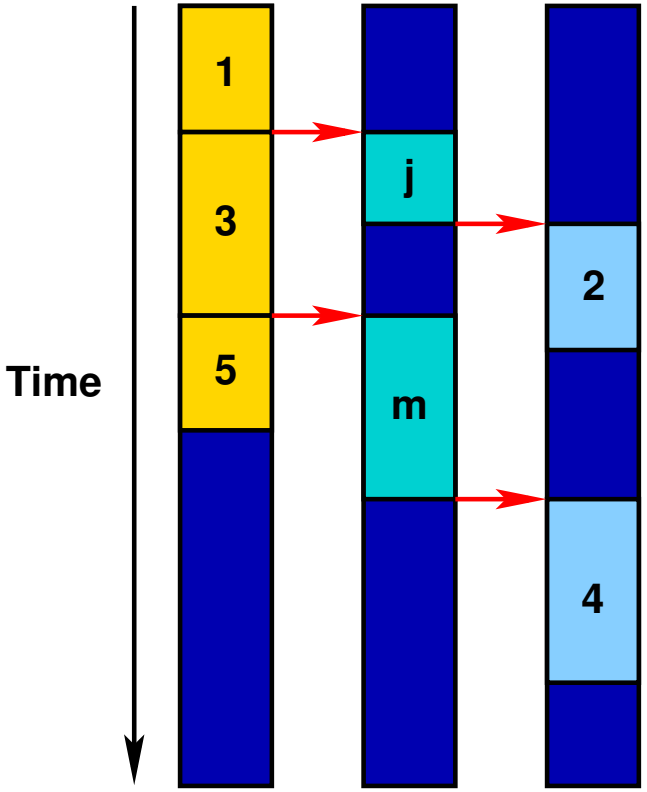


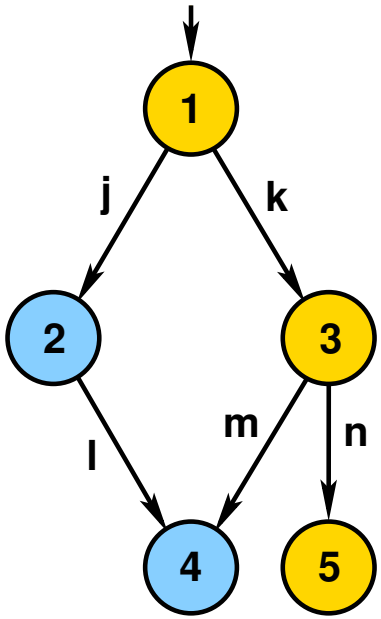




Scheduling







k, l, and n need not be scheduled

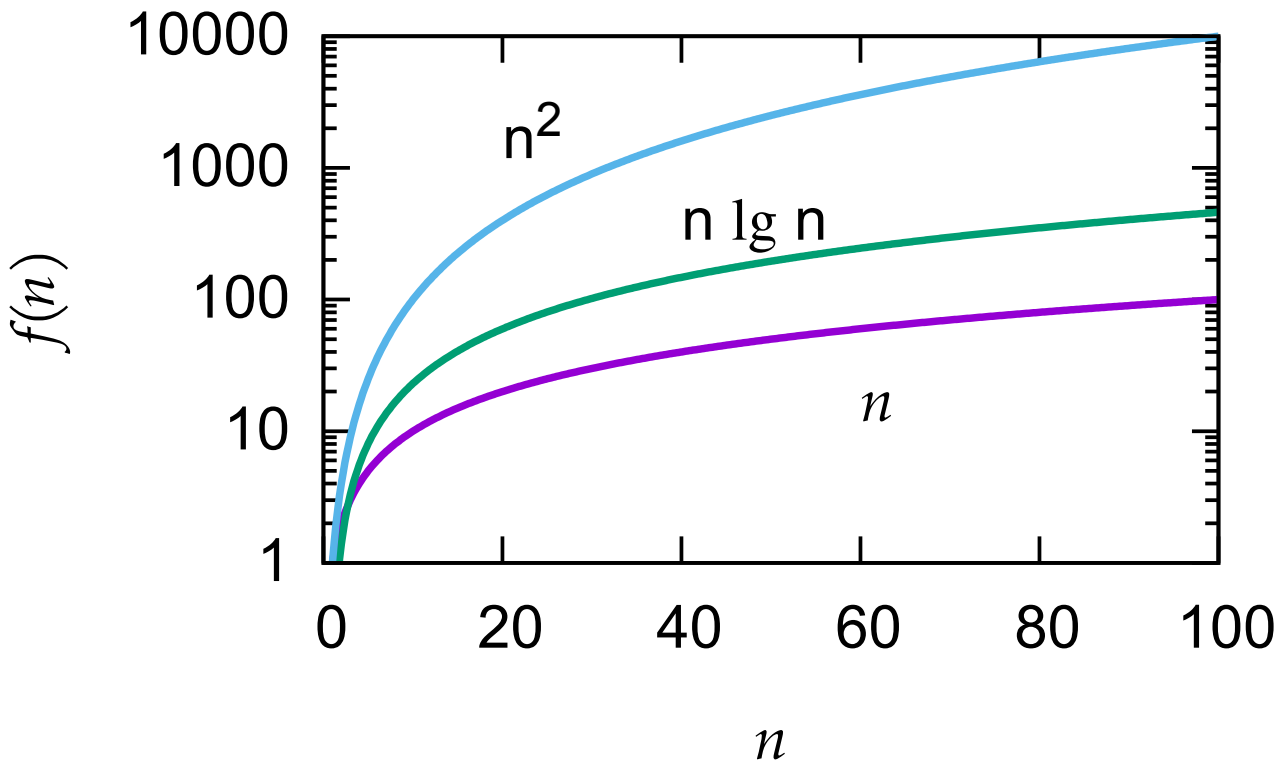
Section outline

Polynomial time complexities

Recall that sorting may be done in $\mathcal{O}(n \lg n)$ time.

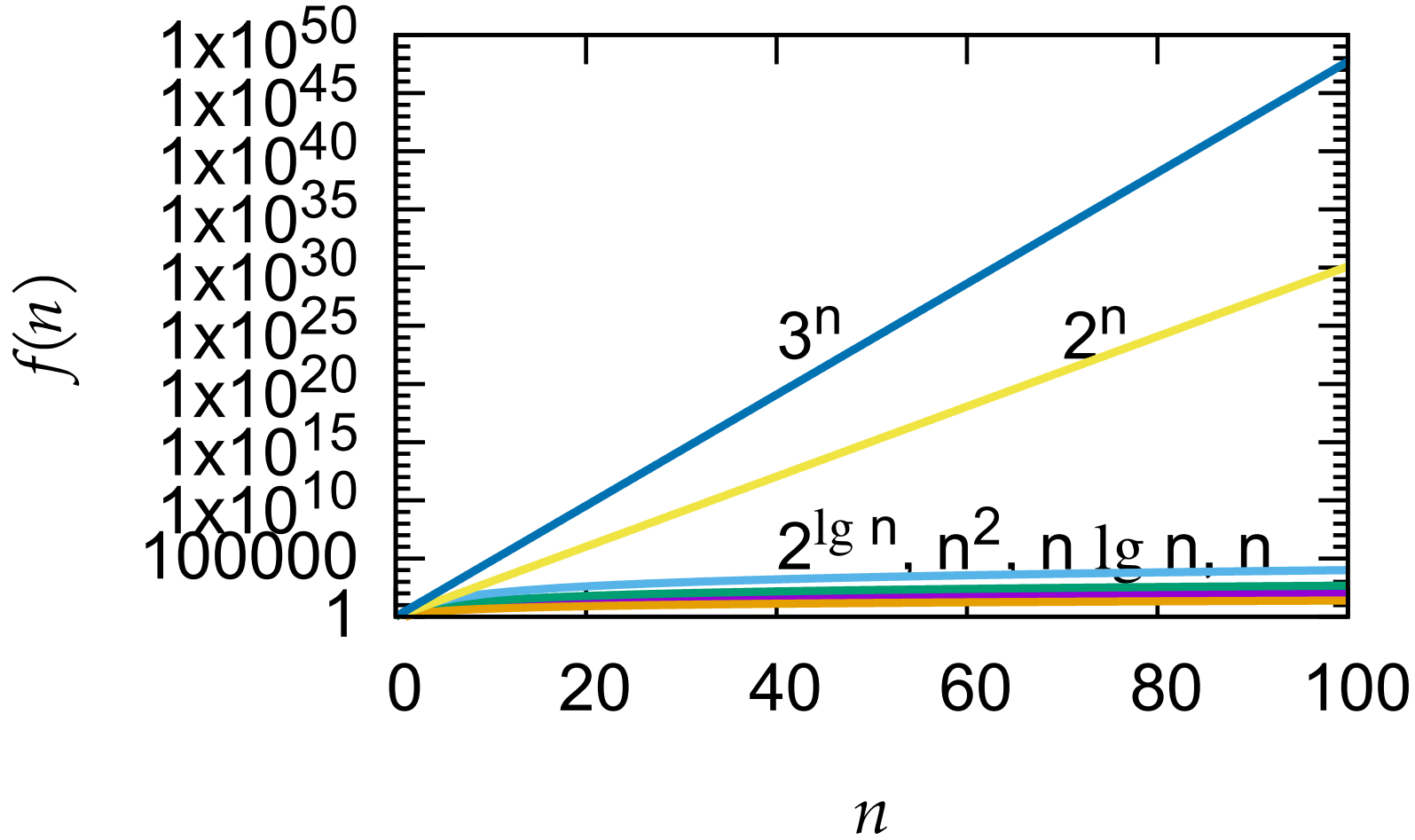
DFS $\in \mathcal{O}(|V| + |E|)$, BFS $\in \mathcal{O}(|V|)$.

Topological sort $\in \mathcal{O}(|V| + |E|)$.



Exponential time complexities

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



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

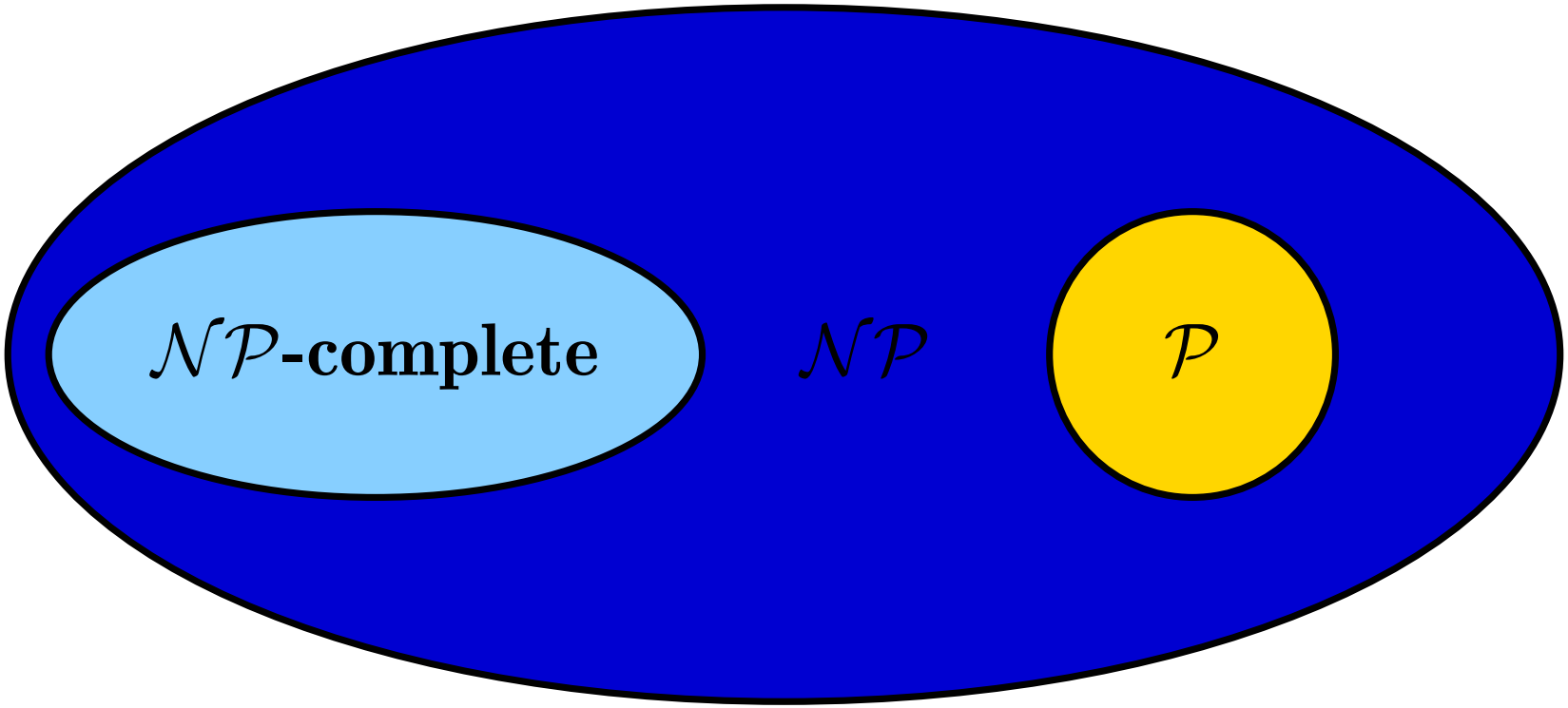
$$\mathcal{P} = \mathcal{NP}$$

$\mathcal{NP}$-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

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

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.

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?

Solve it!

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?

Resort to a suboptimal heuristic.
Bad, but sometimes the only choice.

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?

Develop an approximation algorithm.
Better.

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?

Determine whether all encountered problem instances are constrained.
Wonderful when it works.

One example

O. Coudert, "Exact coloring of real-life graphs is easy," in *Proc. Design Automation Conf.*, June 1997, pp. 121–126.

Graph coloring is very hard.

There are relatively efficient approaches to maximal clique identification.

Real-life graphs are nearly all one perfect.

Section outline

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.

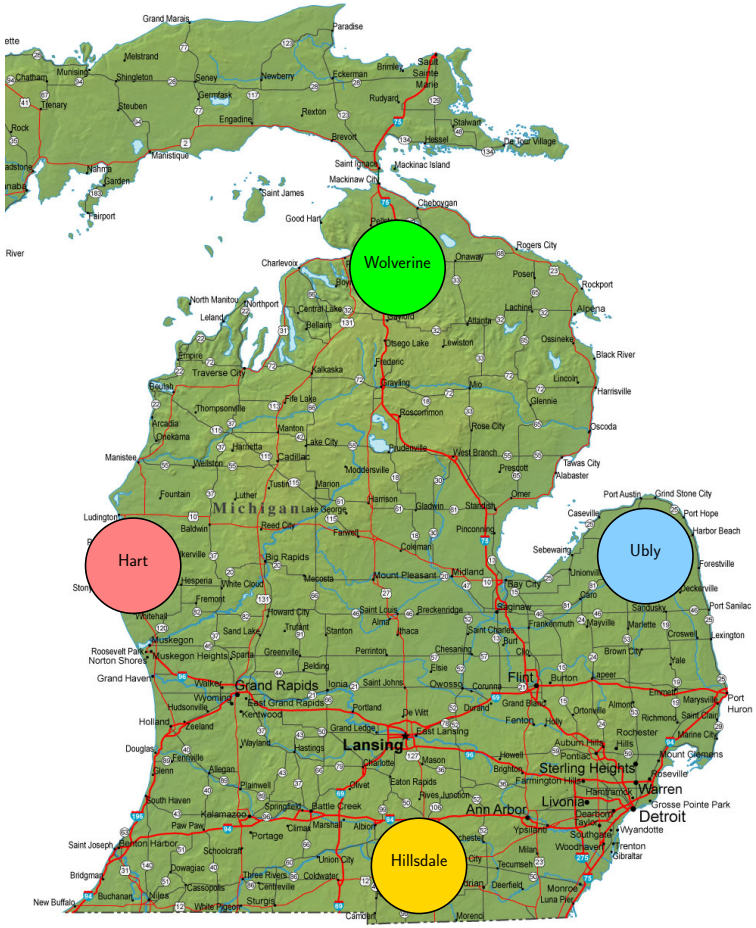
Example problem

Traveling salesman problem

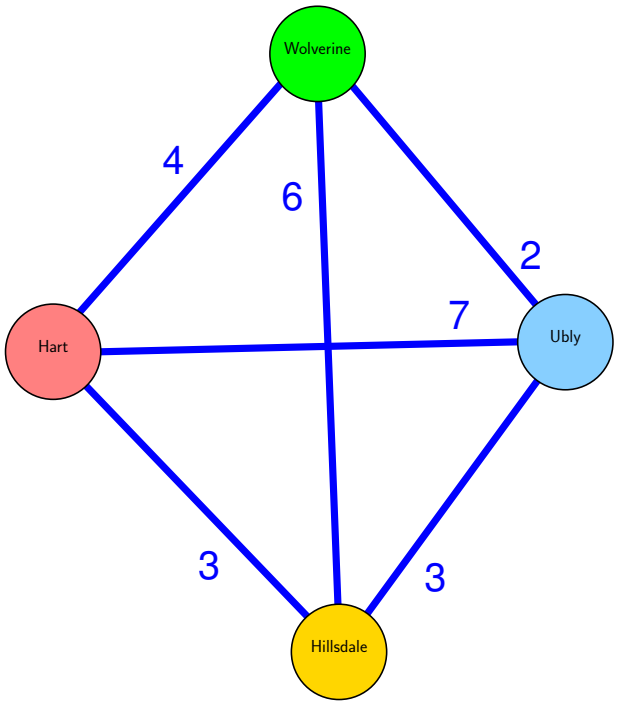
- Find shortest path visiting all cities.
- Very closely related to the Hamiltonian Circuit problem, in which the first city is revisited at the end of the trip.

Hamiltonian circuit

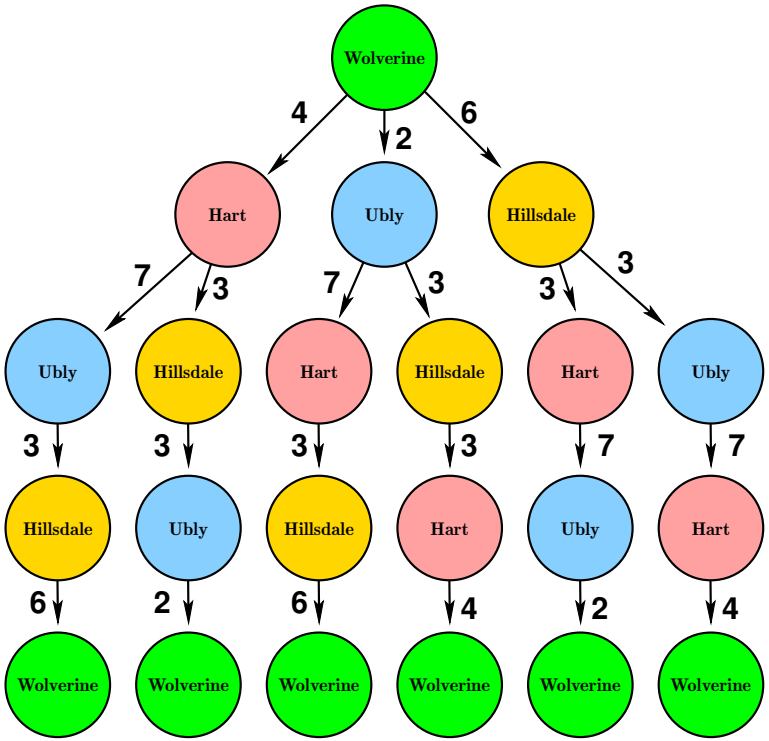
Hamiltonian circuit



Hamiltonian circuit

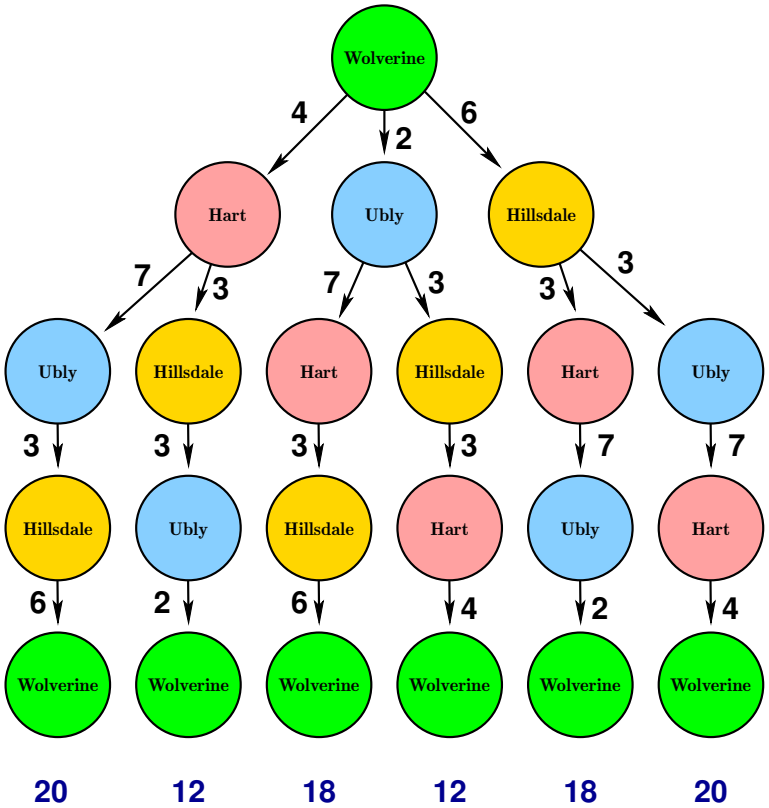


Enumeration



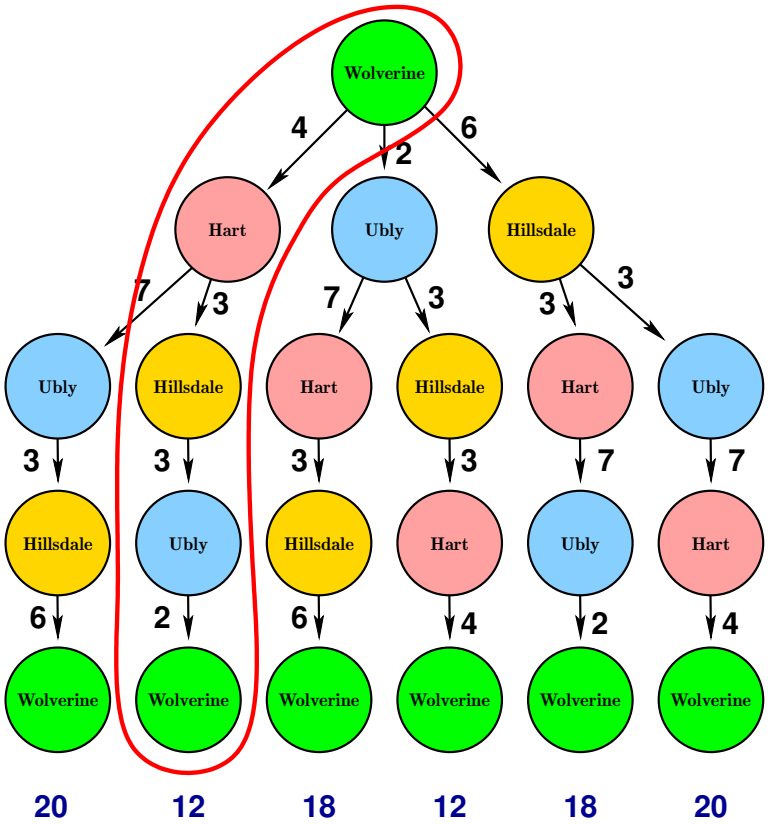
$$t(n) \geq c \cdot k^n \text{ for } n > n_0$$

Enumeration



$$t(n) \geq c \cdot k^n \text{ for } n > n_0$$

Enumeration



$$t(n) \geq c \cdot k^n \text{ for } n > n_0$$

Branch and bound

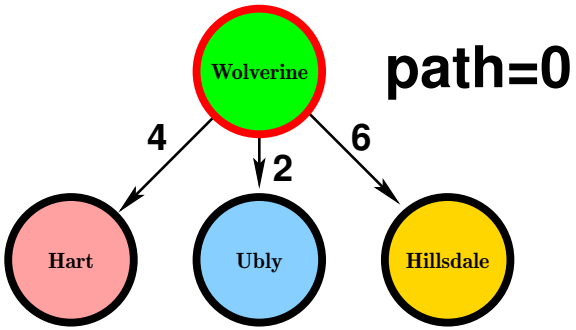
Keep track of minimal encountered cost.

When a path has a higher cost, terminate.

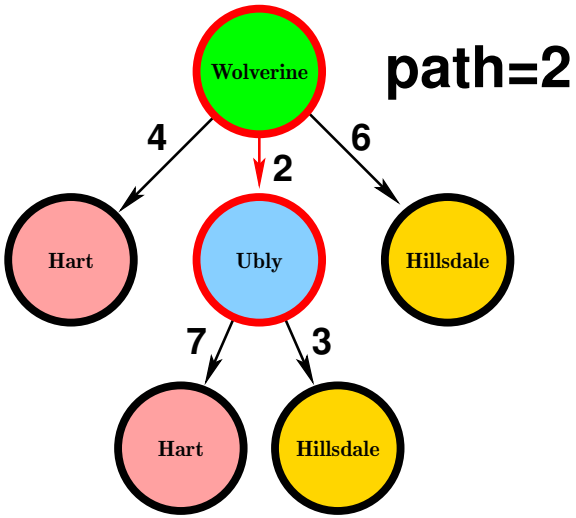
Branch and bound

path=0

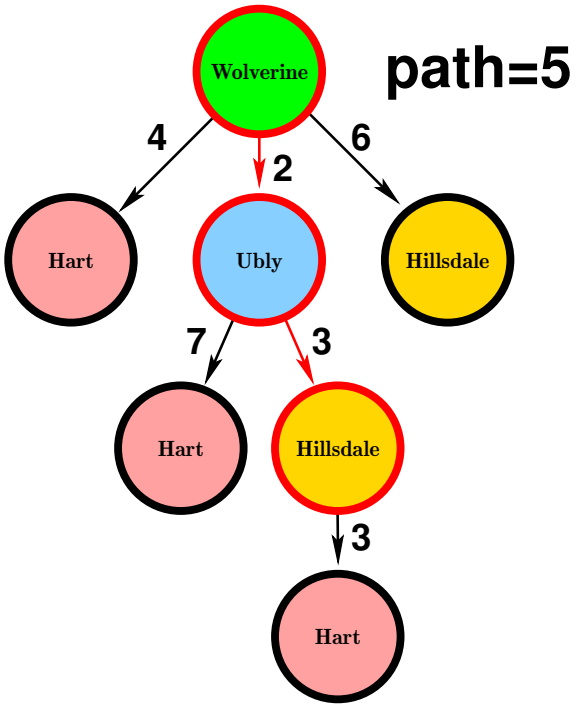
Branch and bound



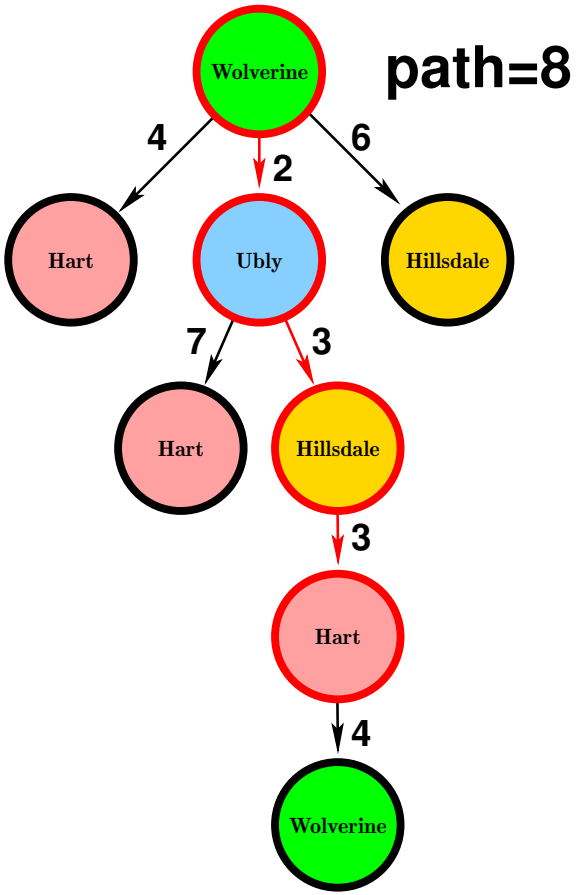
Branch and bound



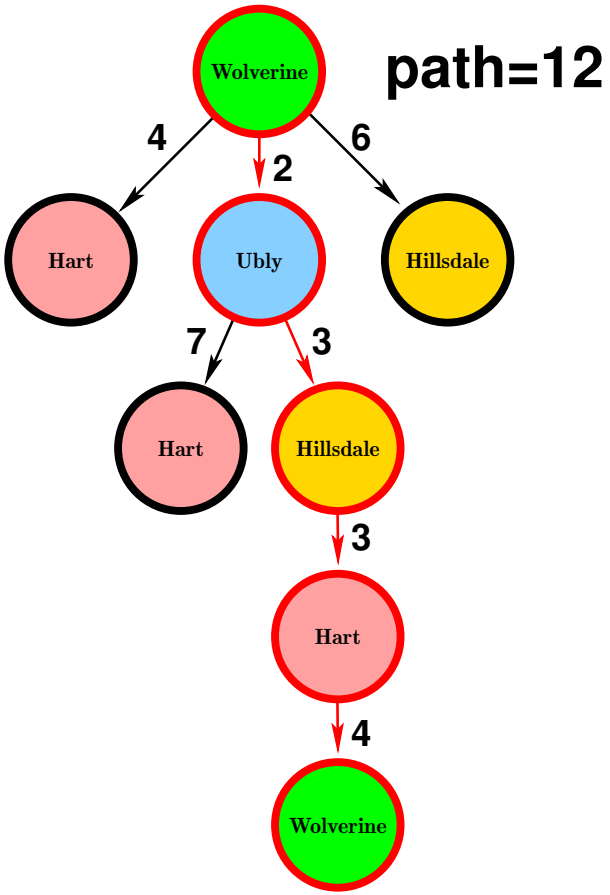
Branch and bound



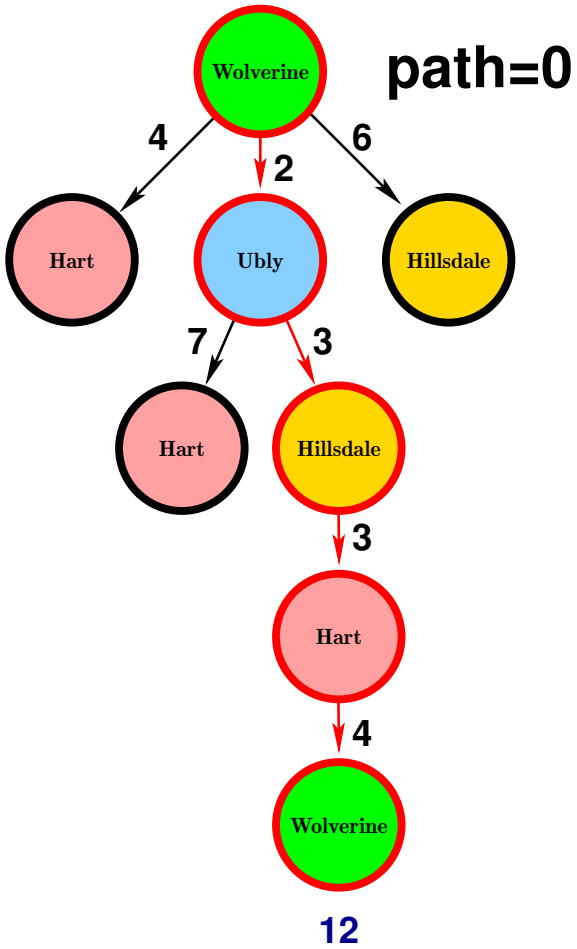
Branch and bound



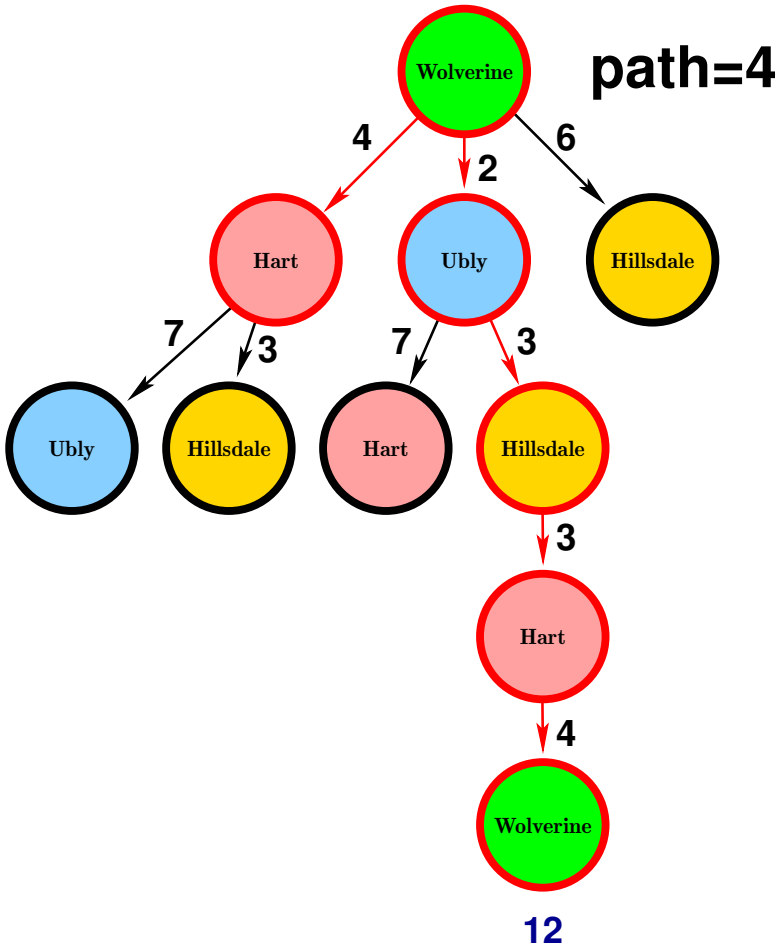
Branch and bound



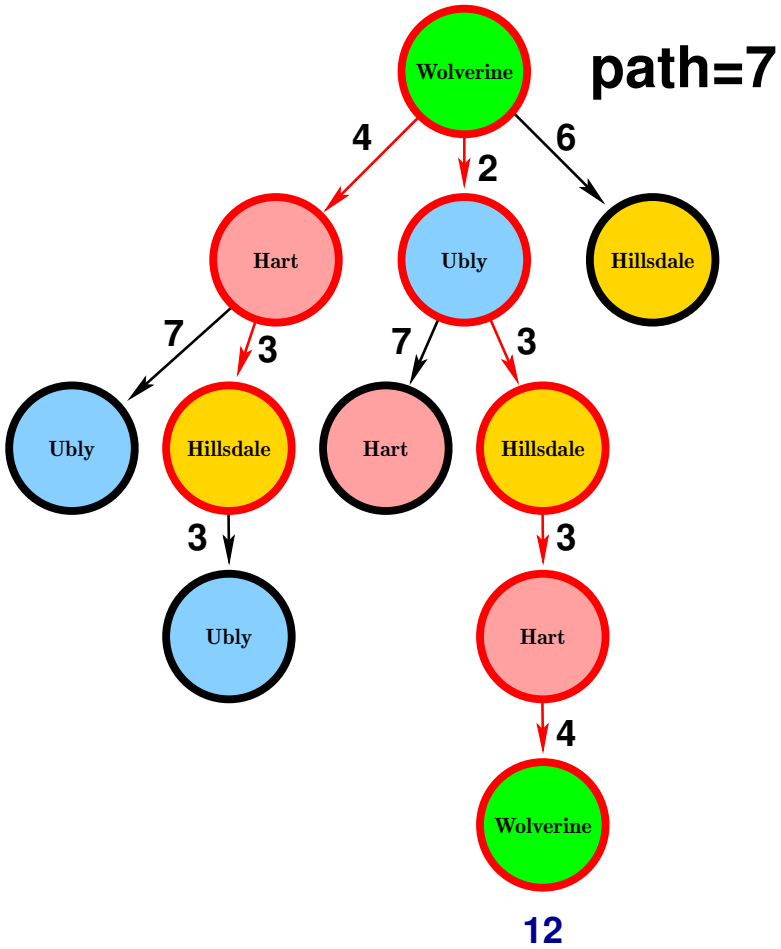
Branch and bound



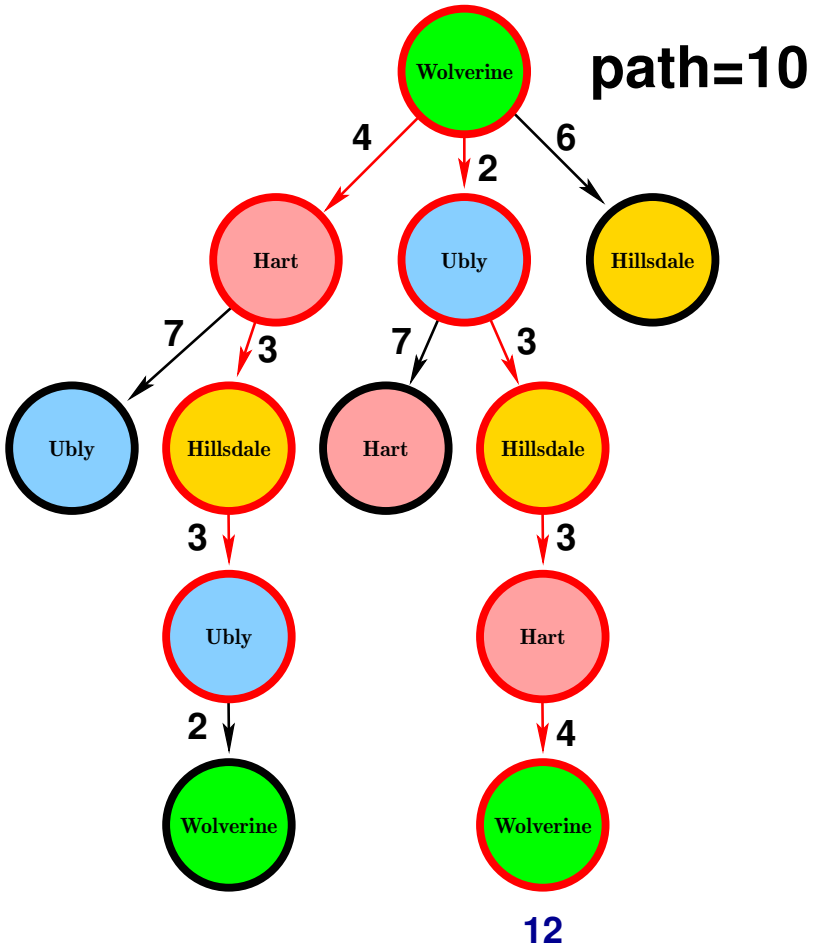
Branch and bound



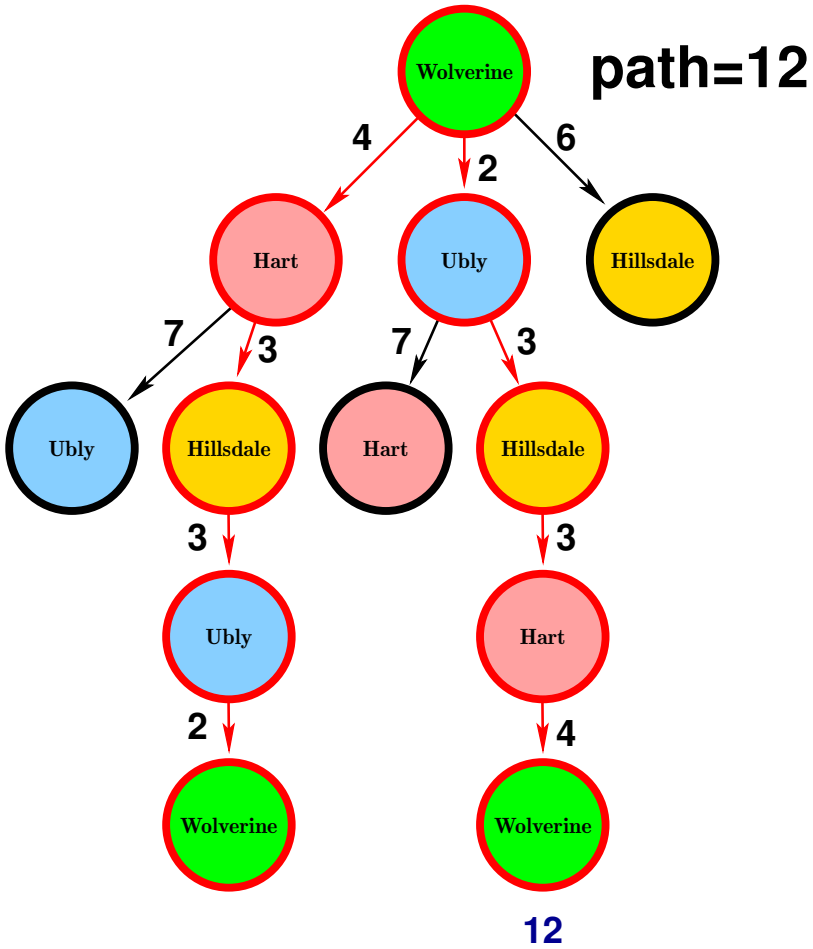
Branch and bound



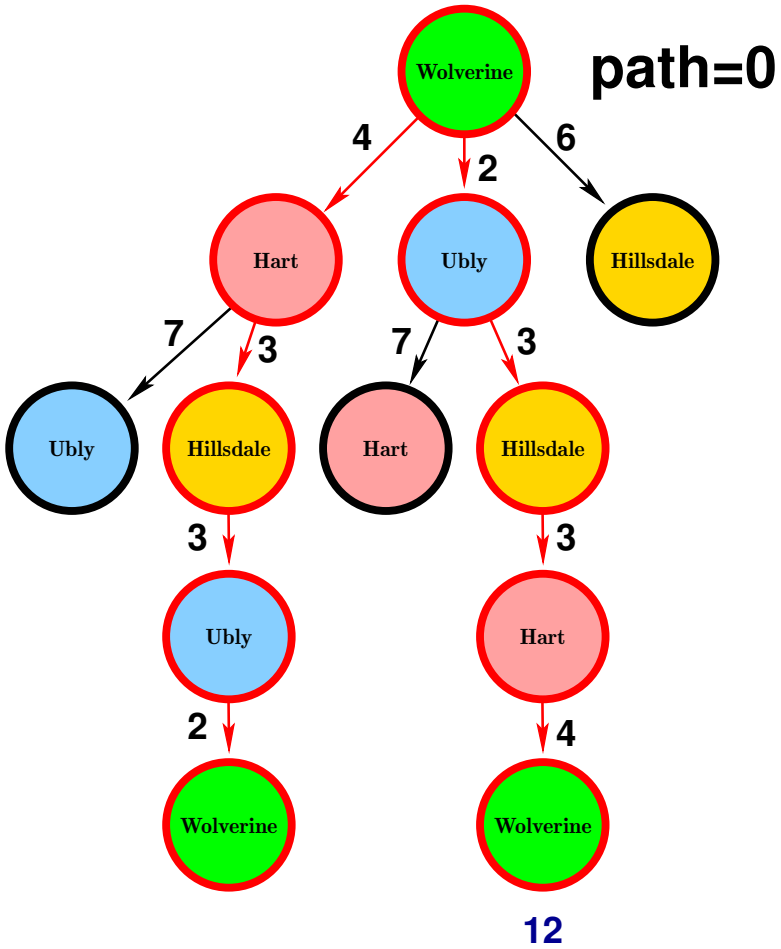
Branch and bound



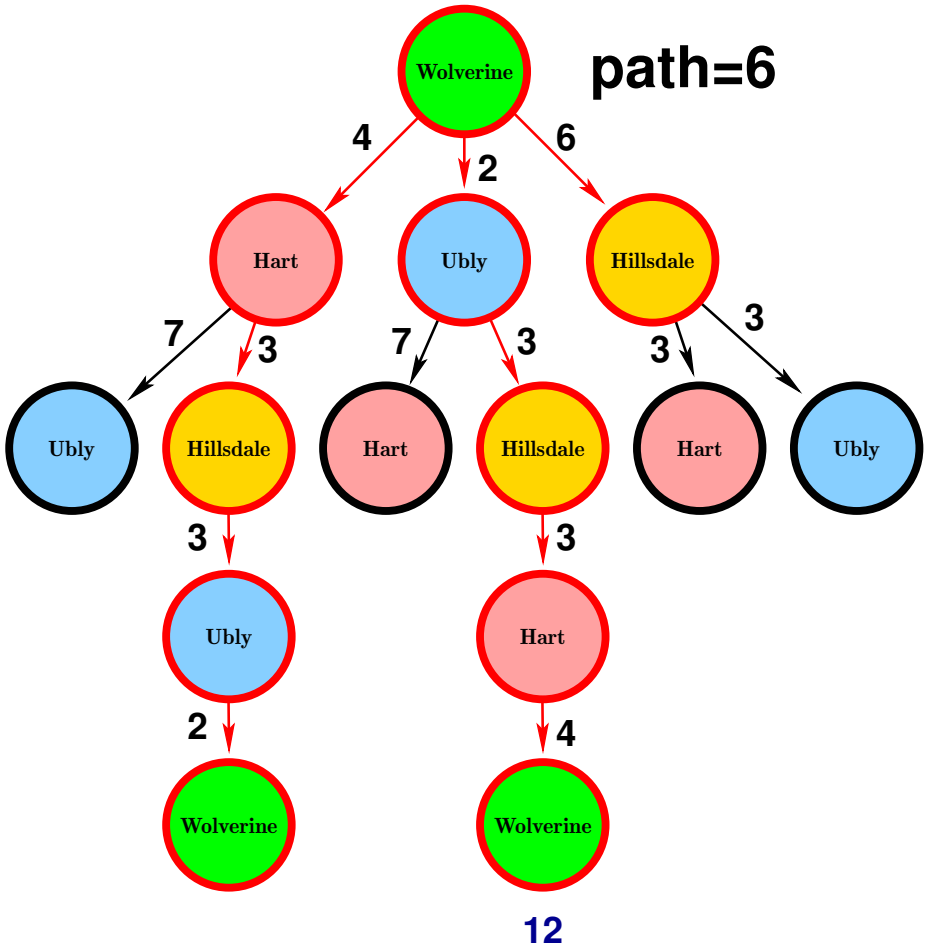
Branch and bound



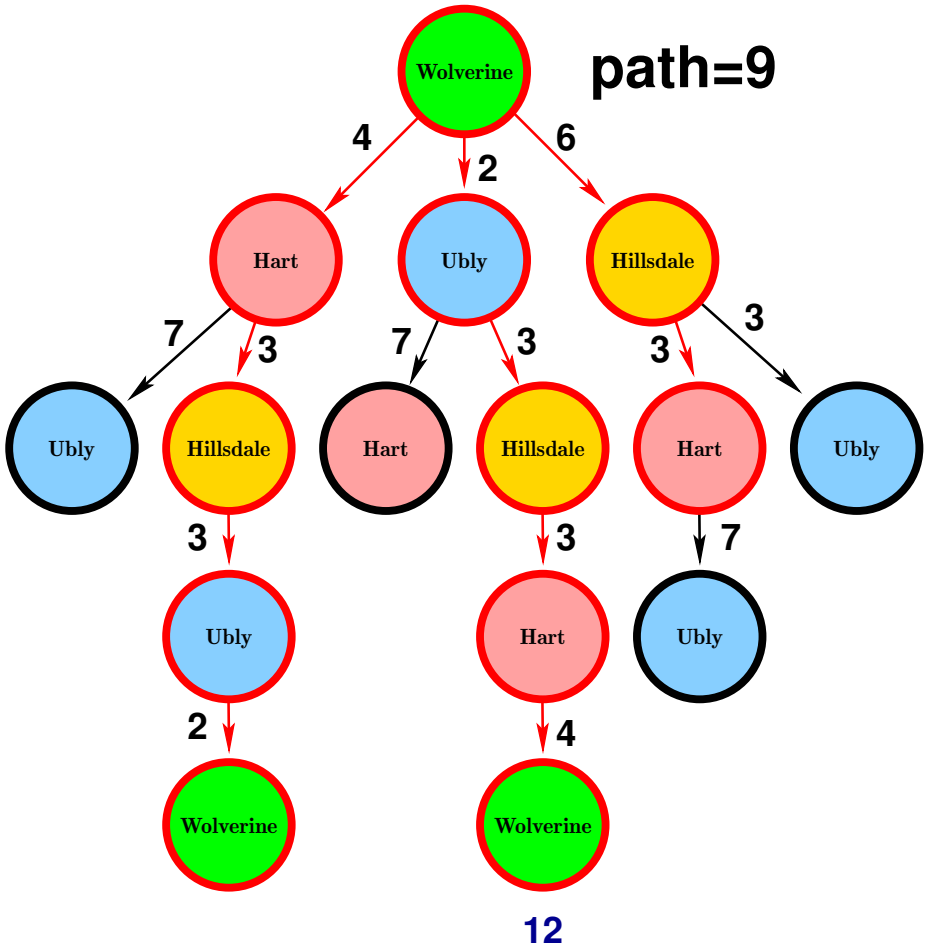
Branch and bound



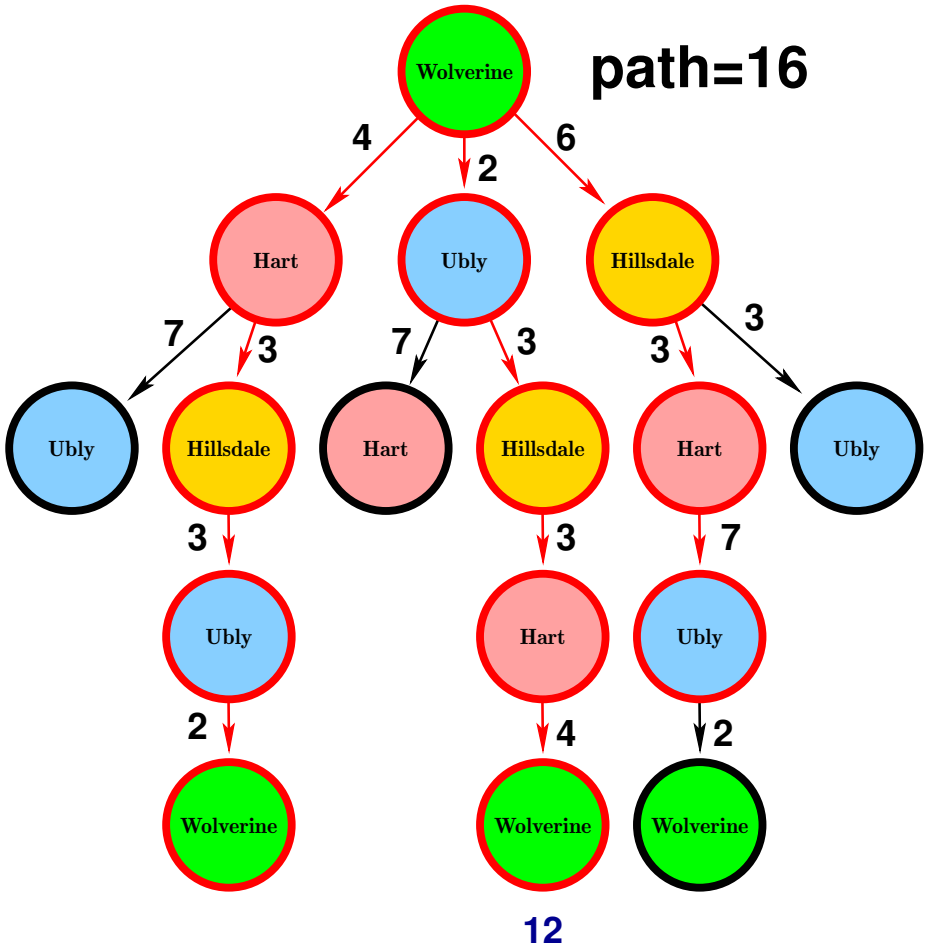
Branch and bound



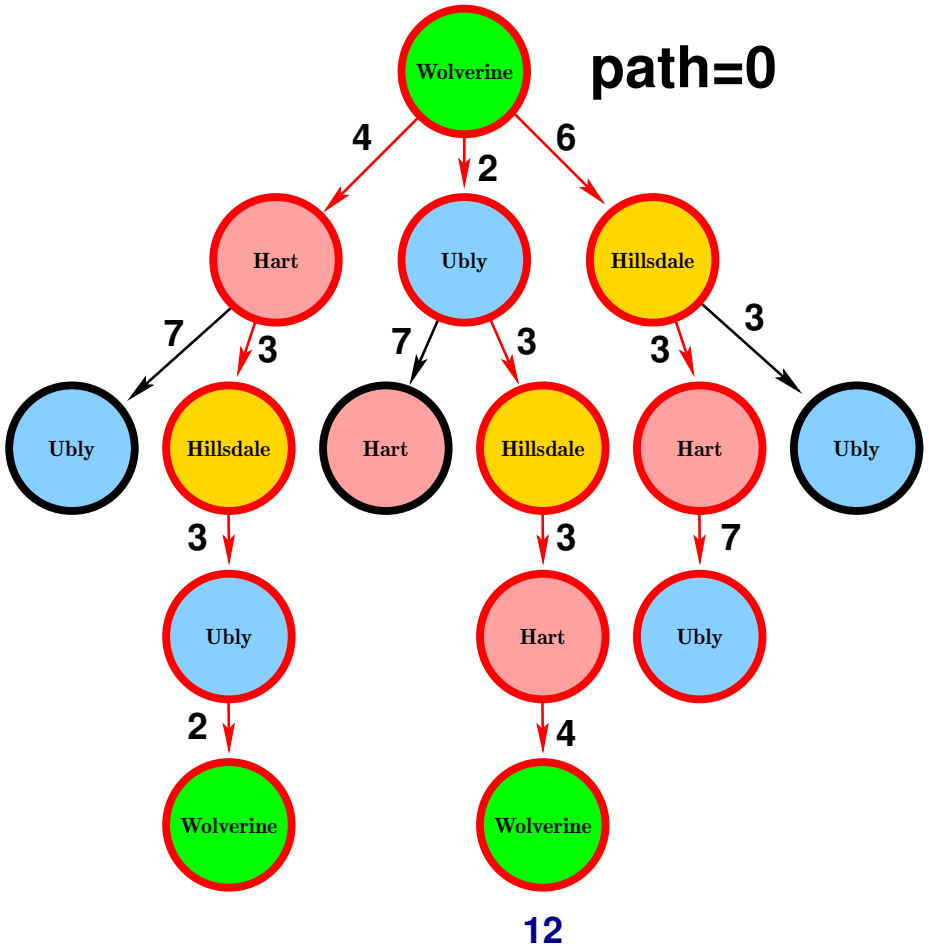
Branch and bound



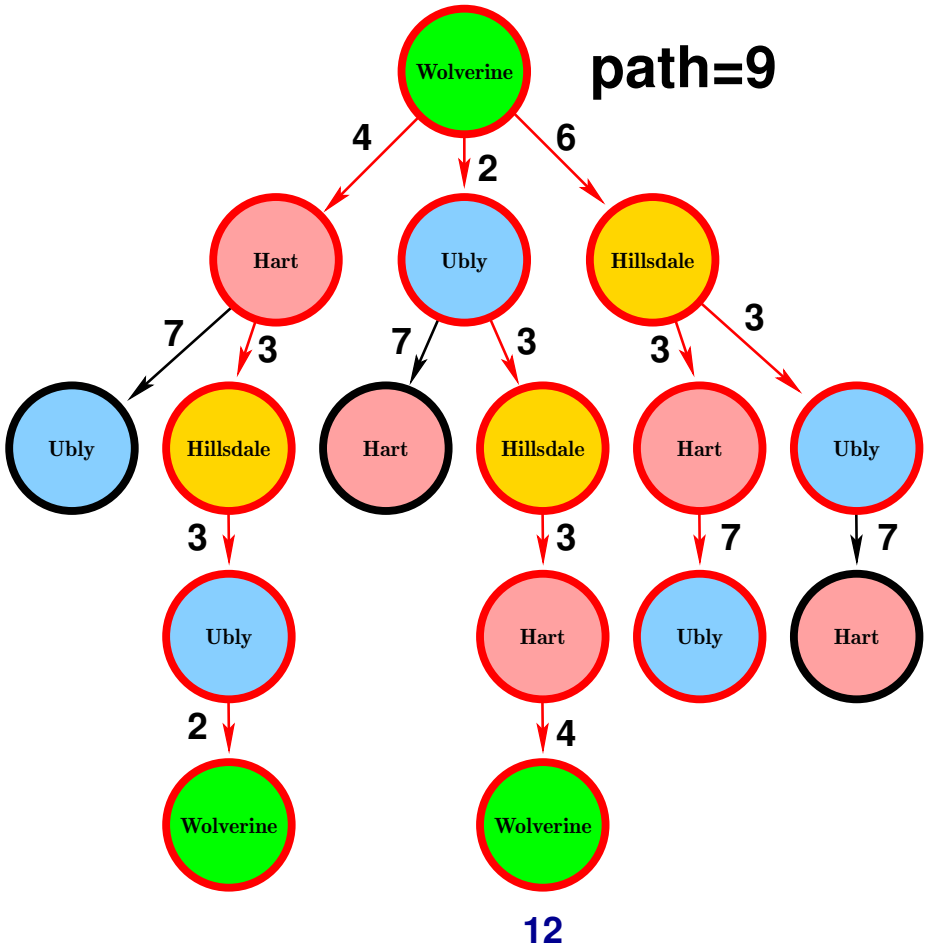
Branch and bound



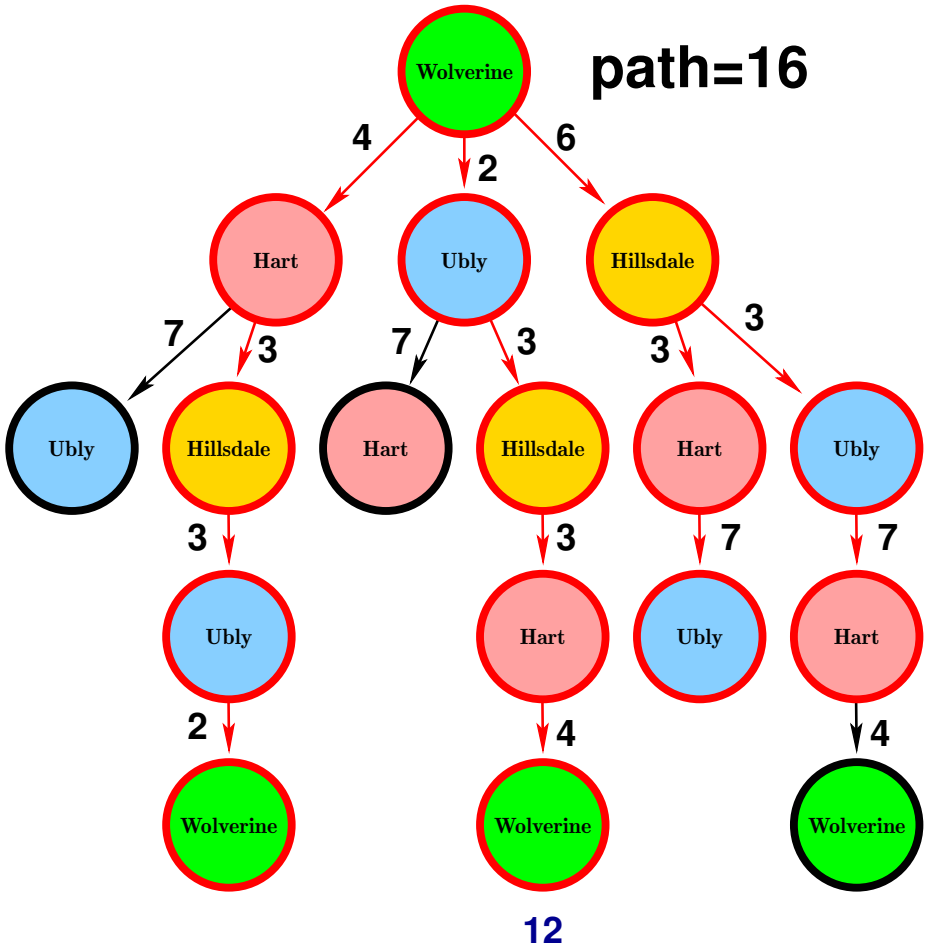
Branch and bound



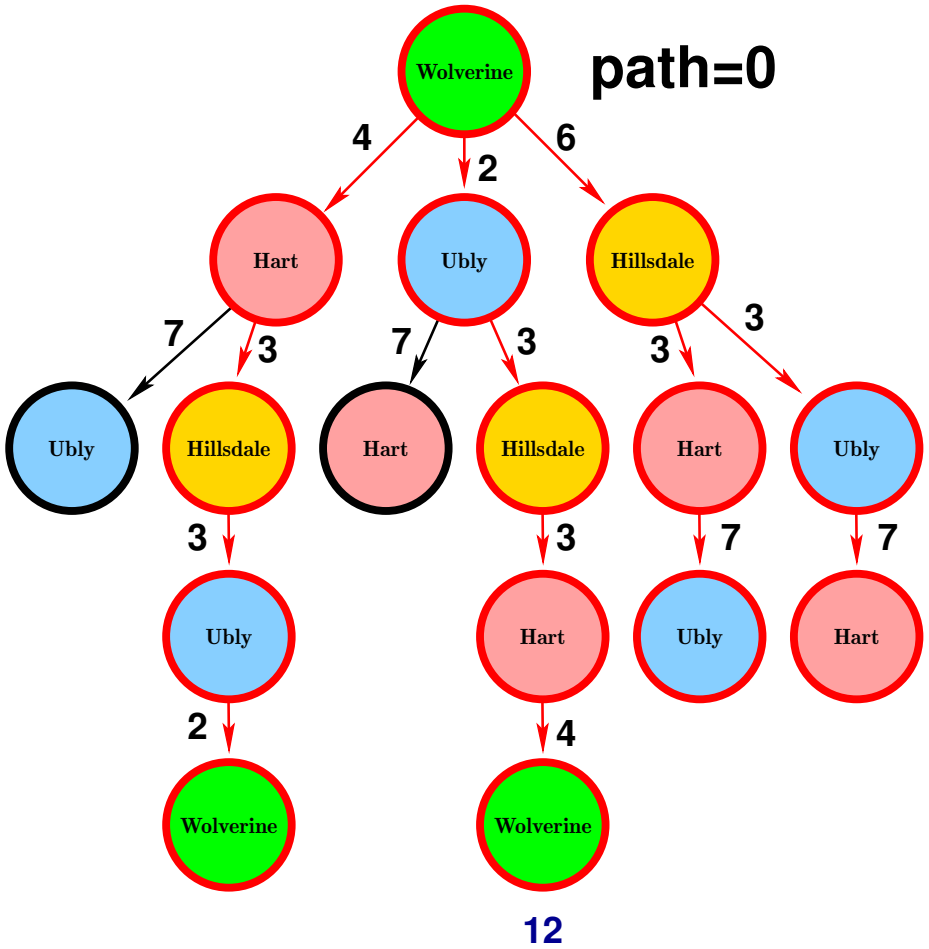
Branch and bound



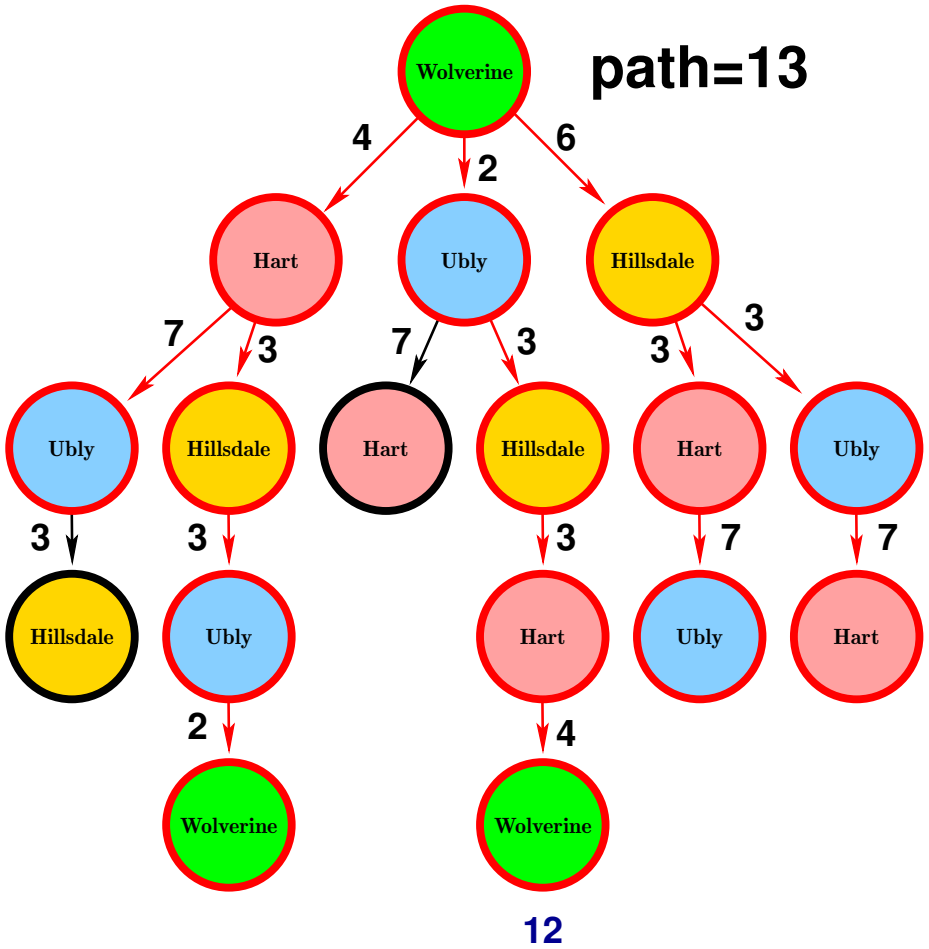
Branch and bound



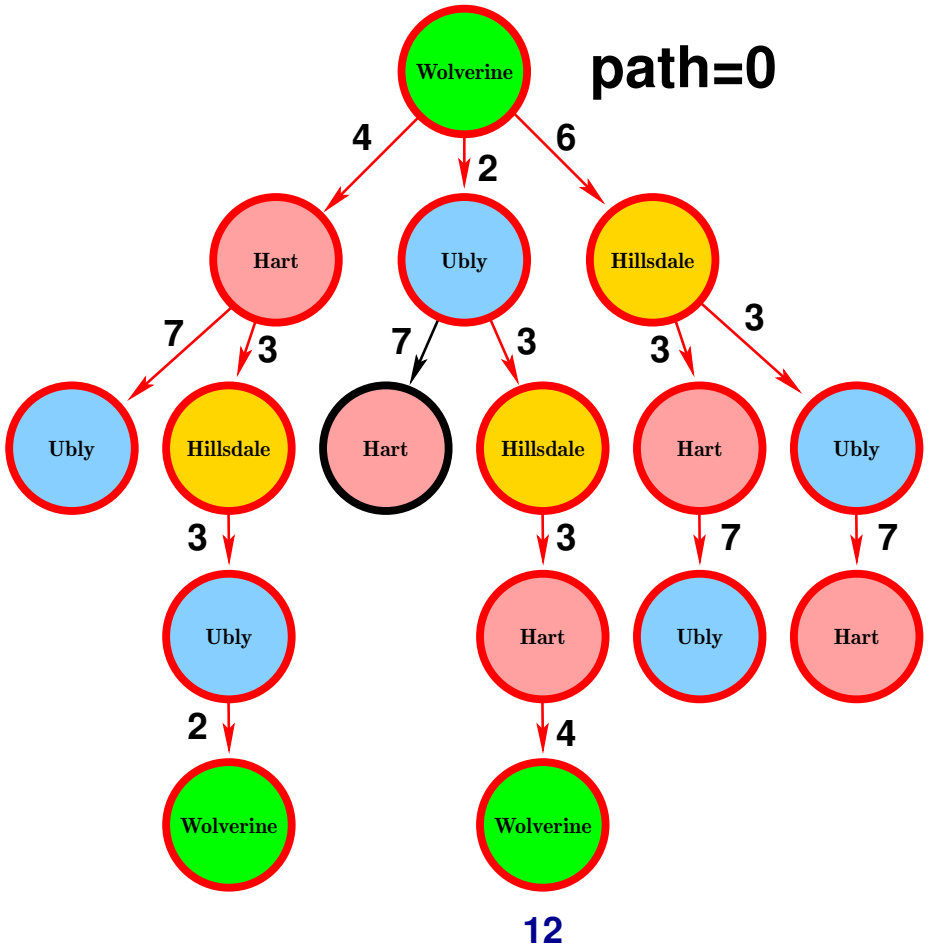
Branch and bound



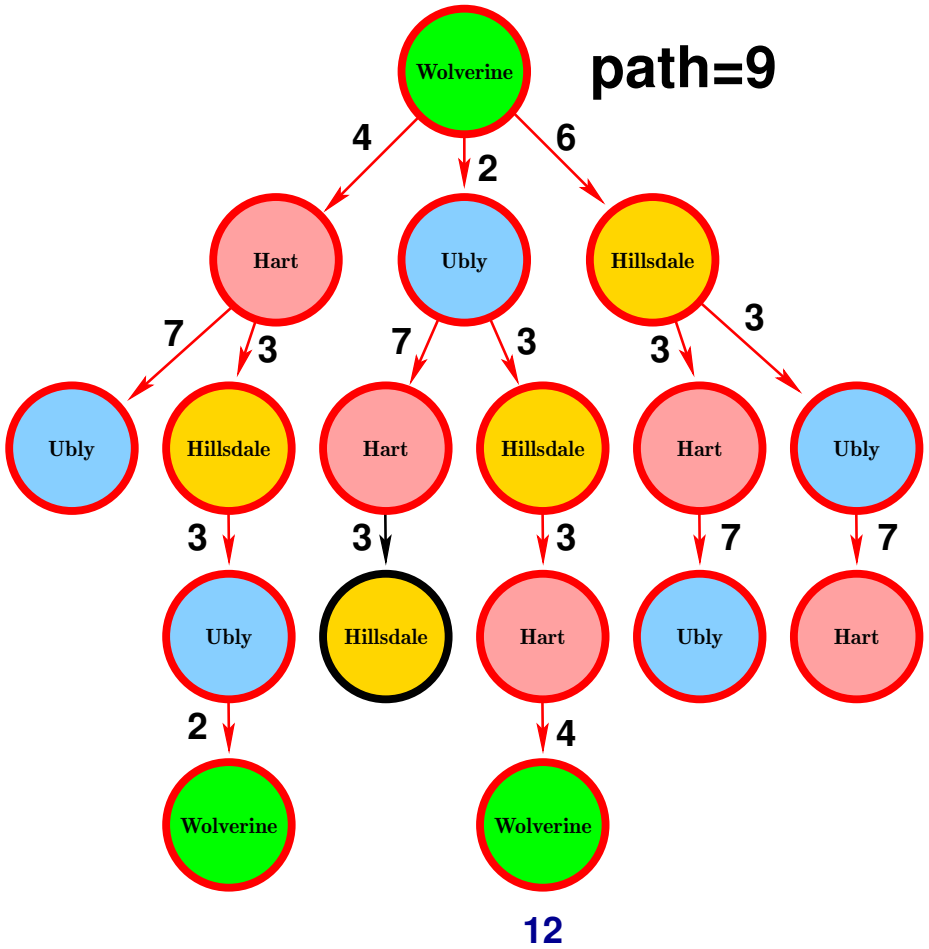
Branch and bound



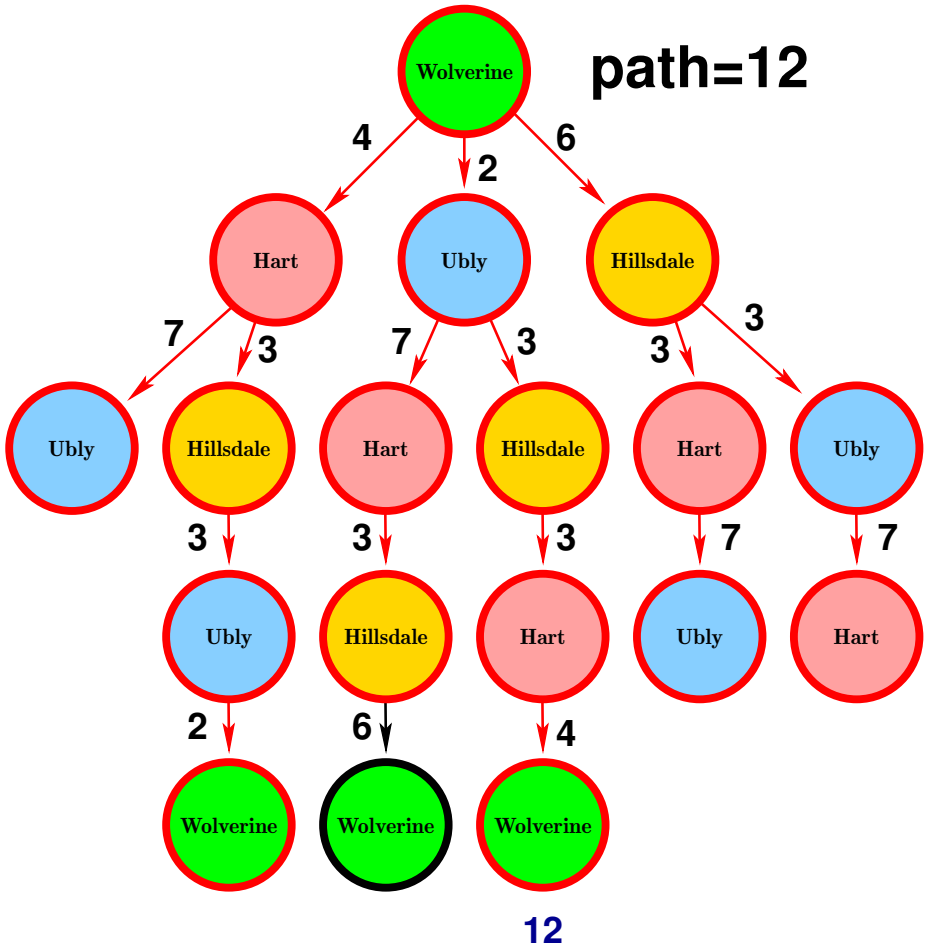
Branch and bound



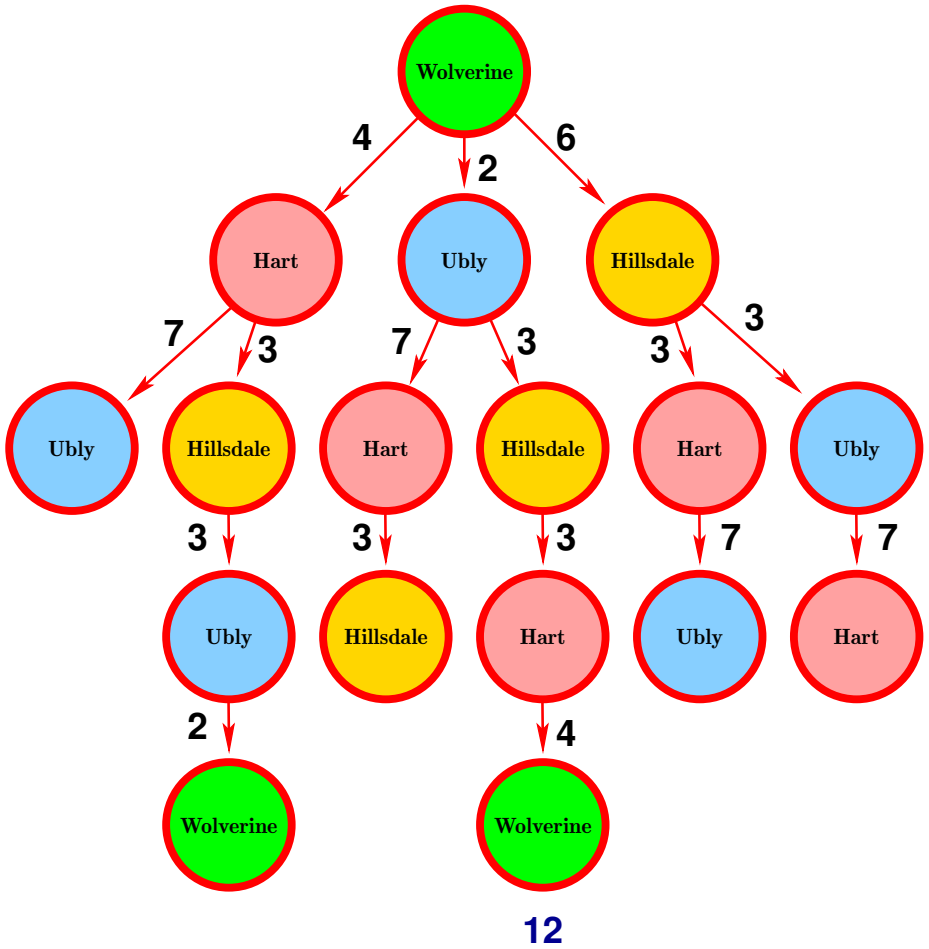
Branch and bound



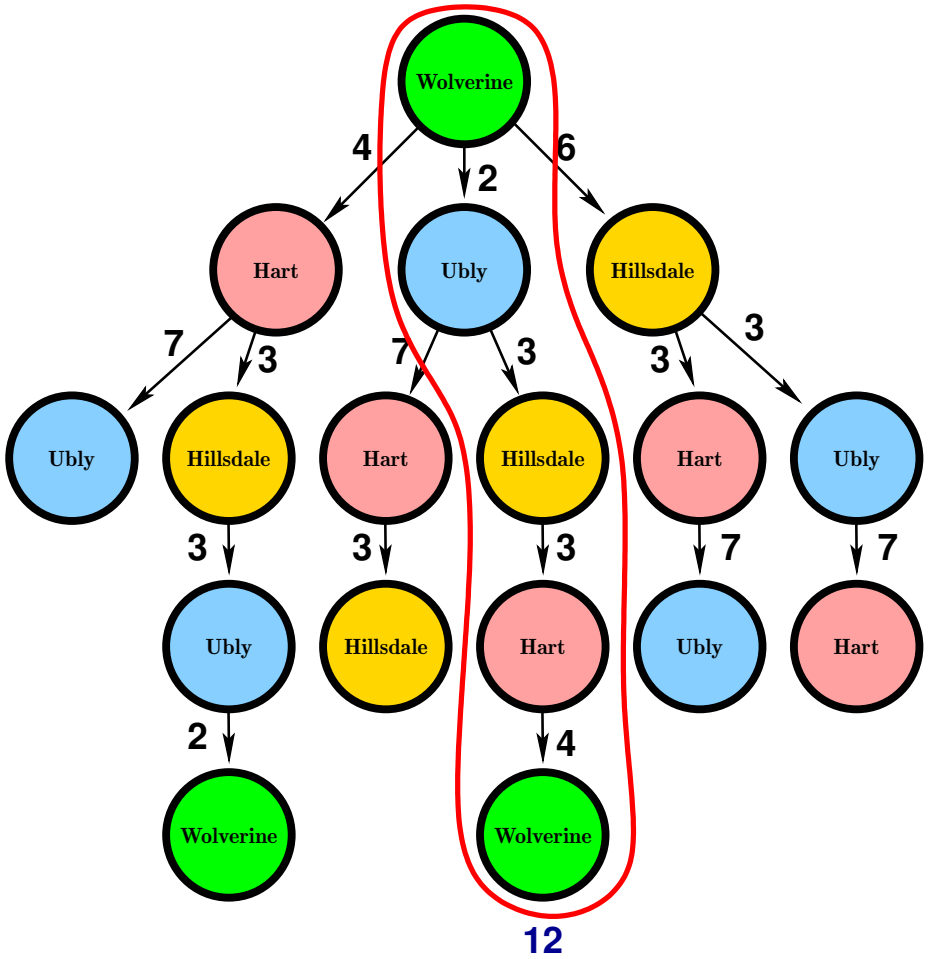
Branch and bound



Branch and bound



Branch and bound



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:

$$\begin{aligned}
a_{11} \cdot x_1 + a_{12} \cdot x_2 + \cdots + a_{1n} \cdot x_n &\leq, =, \geq \quad b_1 \\
a_{21} \cdot x_1 + a_{22} \cdot x_2 + \cdots + a_{2n} \cdot x_n &\leq, =, \geq \quad b_2 \\
&\cdots\cdots\cdots\cdots \\
a_{n1} \cdot x_1 + a_{1n} \cdot x_2 + \cdots + a_{nn} \cdot x_n &\leq, =, \geq \quad b_n \\
\forall x_i \in x, x_i \geq 0 \qquad & \forall a_{jk} \in A, a_{jk} \in R
\end{aligned}$$

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 partial ILP formulation for TSP

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

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

We'll consider a scheduling problem in more detail later.

Backtracking iterative improvement

Allows B steps of backtracking.

Can be incomplete.

Complete if $B =$ the problem decision depth.

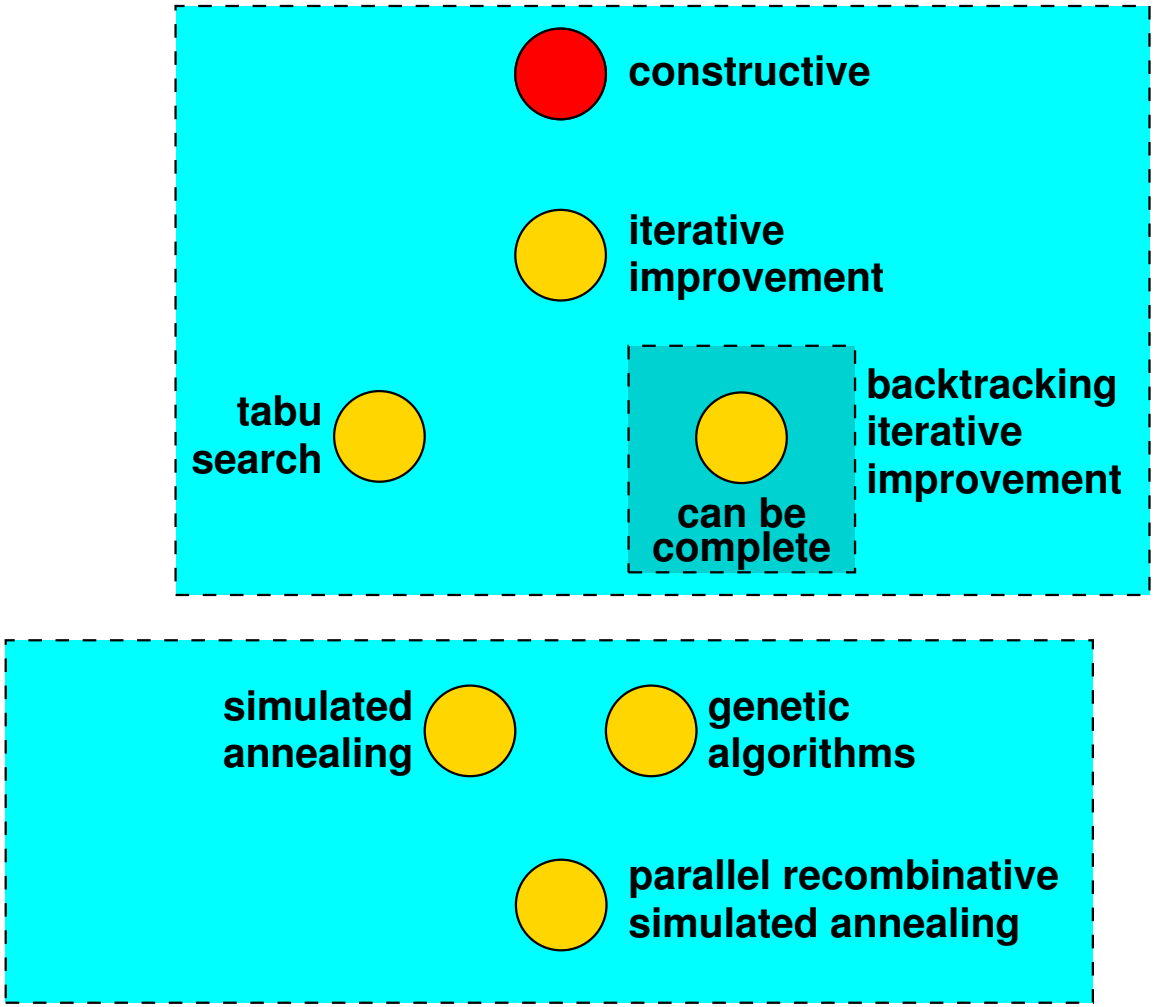
Allows use of problem-speficic heuristics for ordering.

Incomplete if $B <$ decision depth.

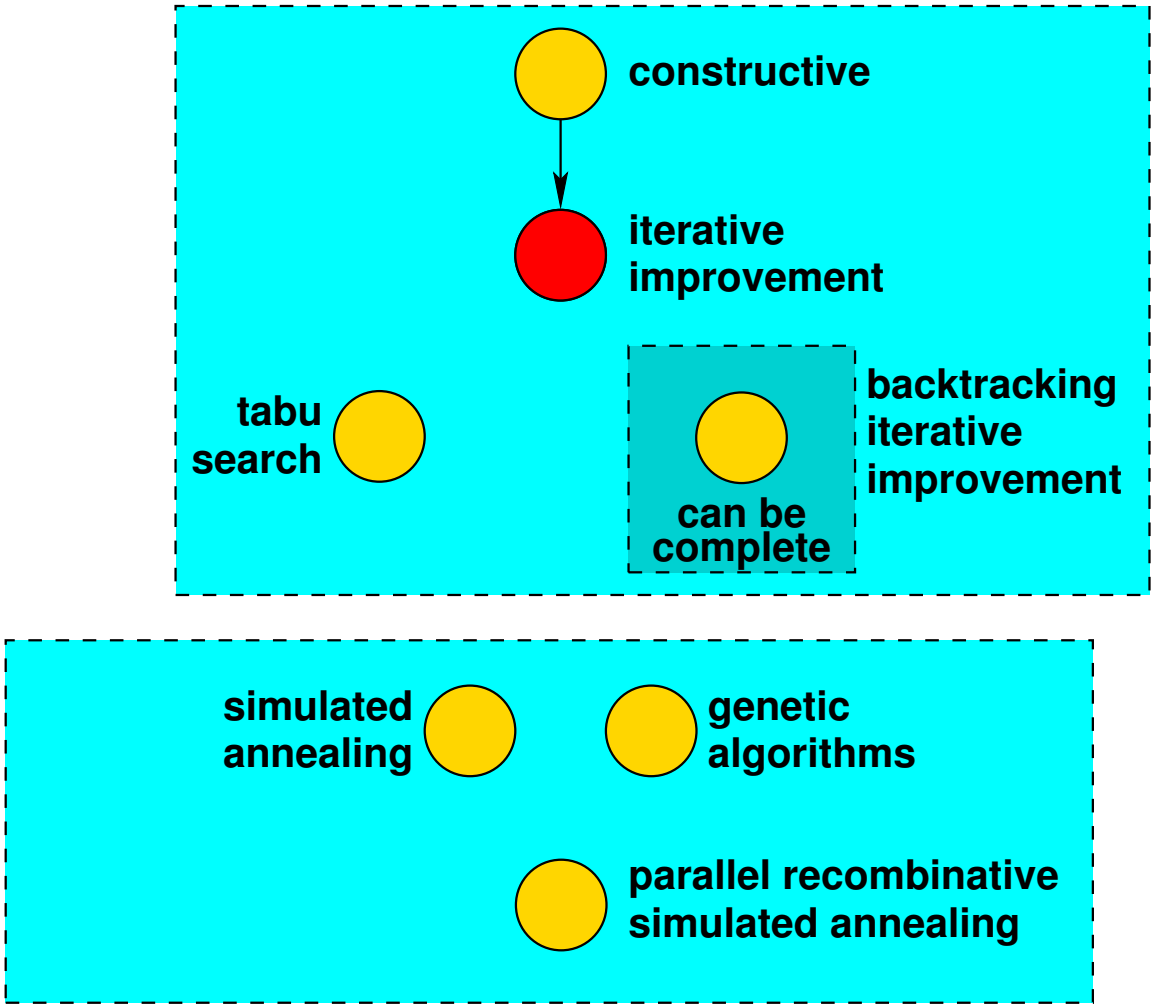
More on this later.

Section outline

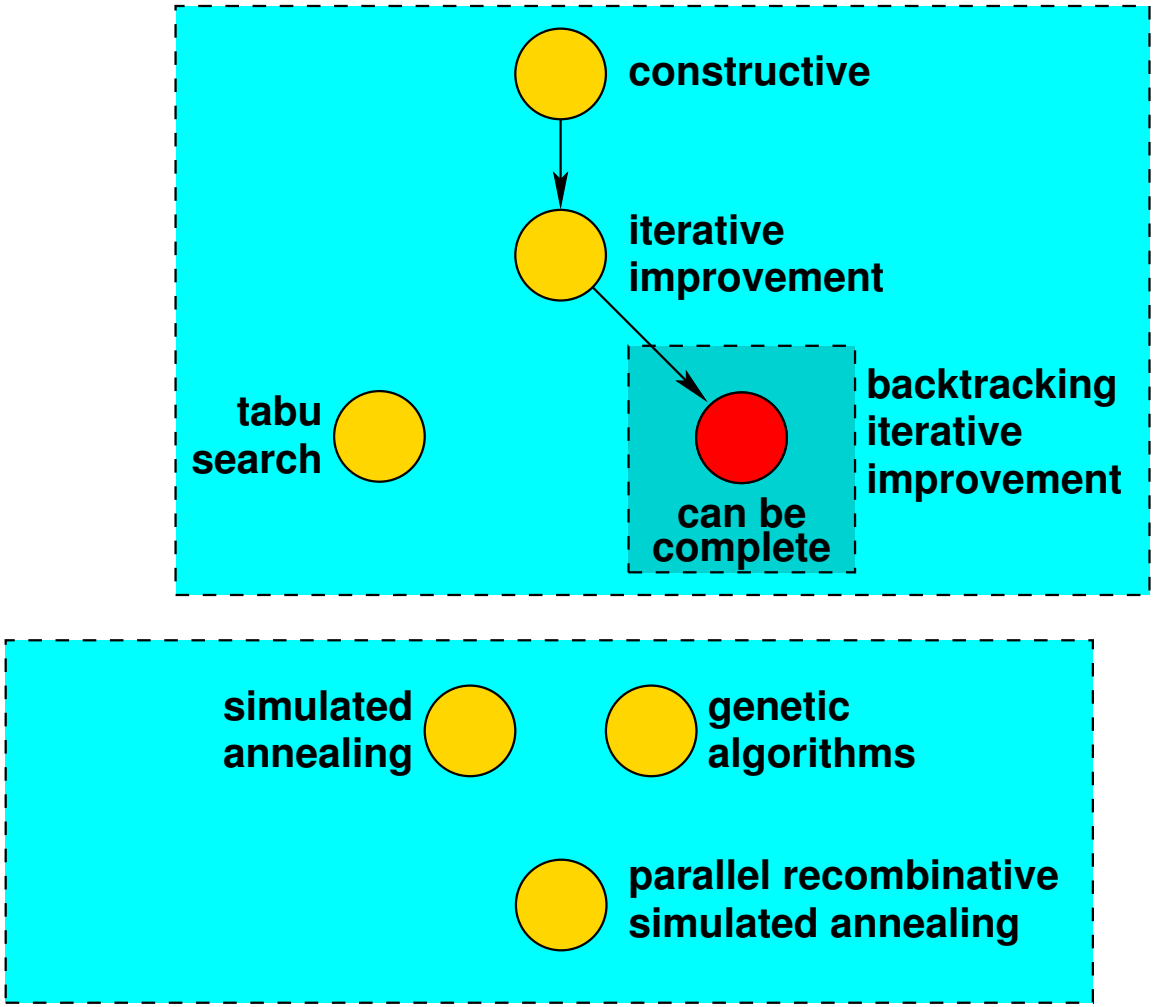
Optimization techniques



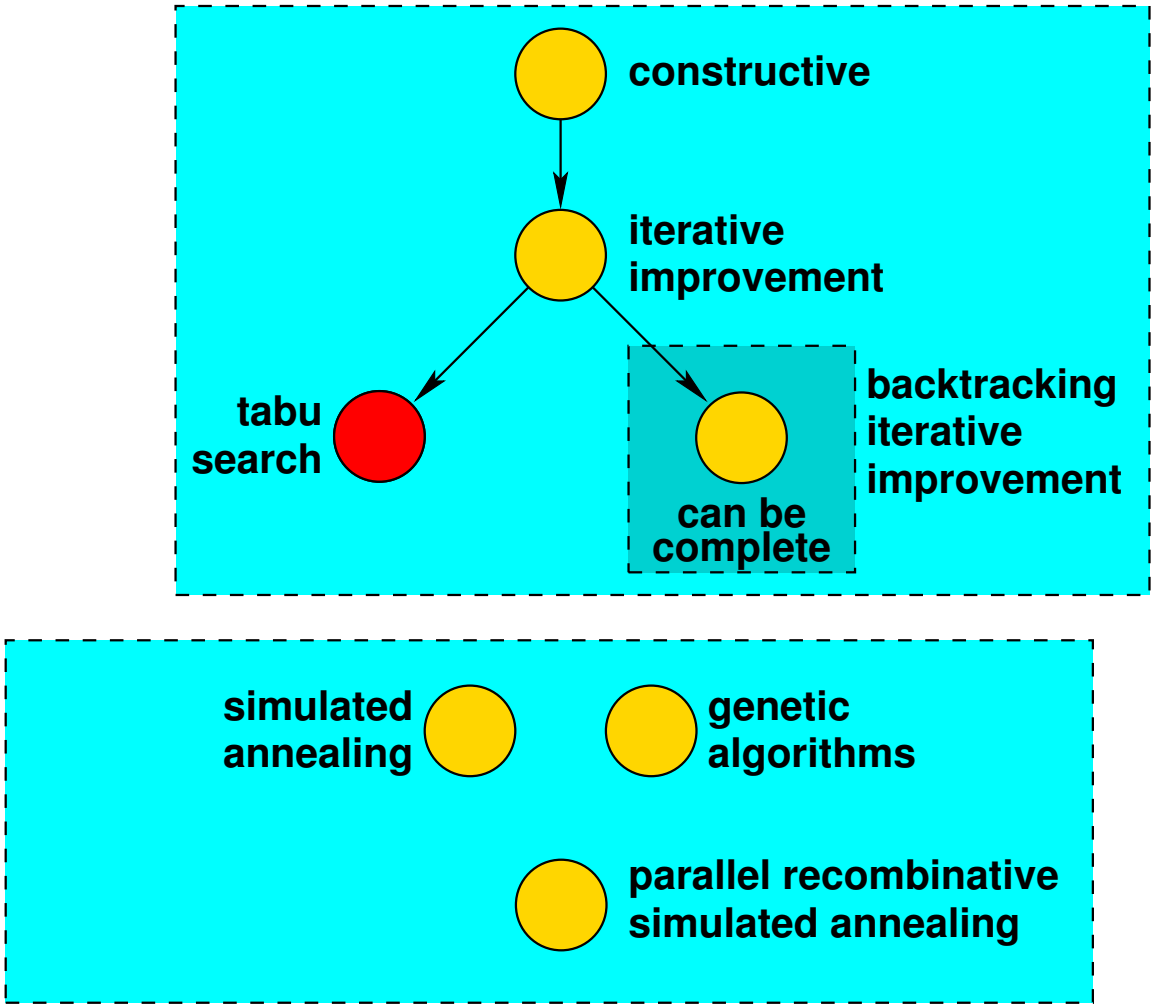
Optimization techniques



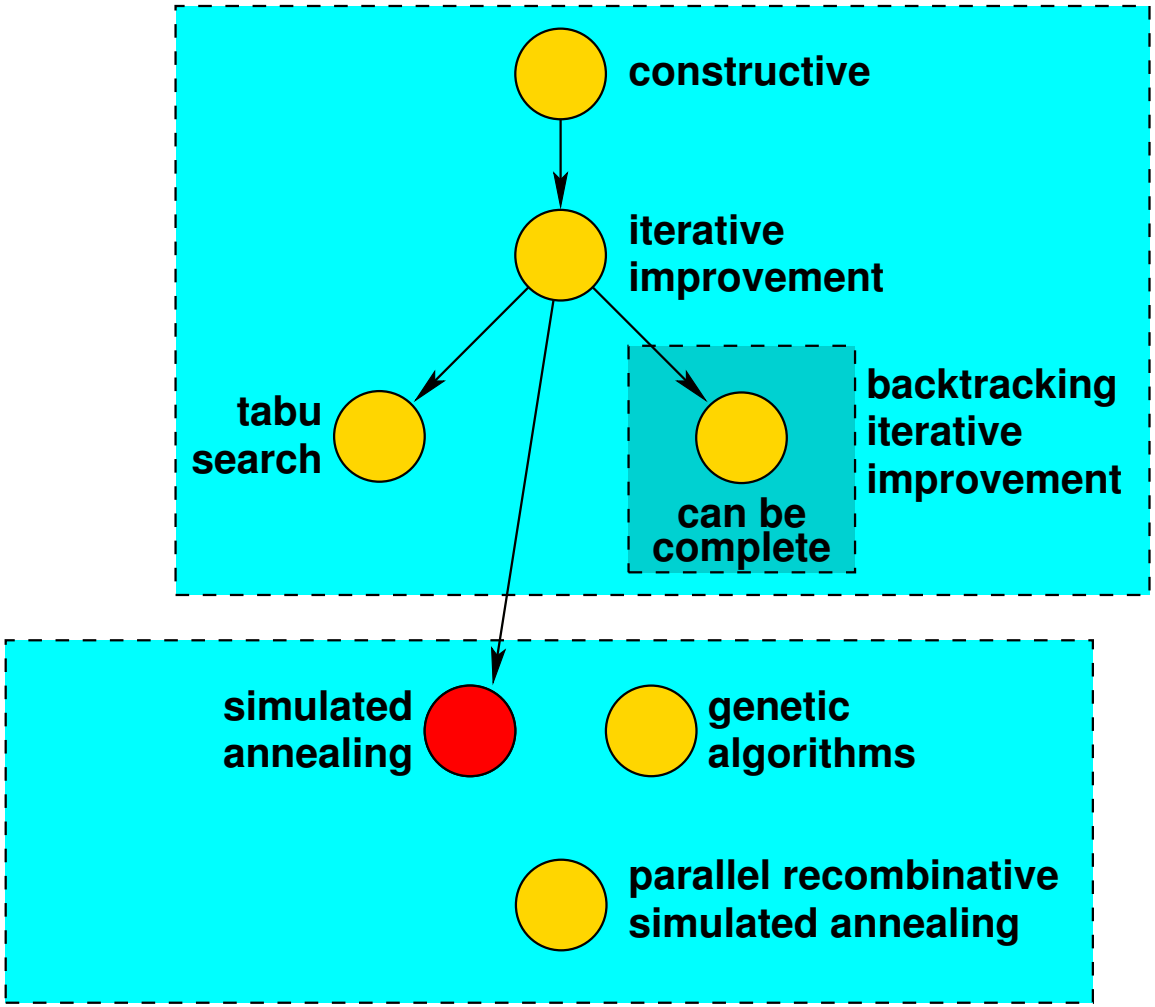
Optimization techniques



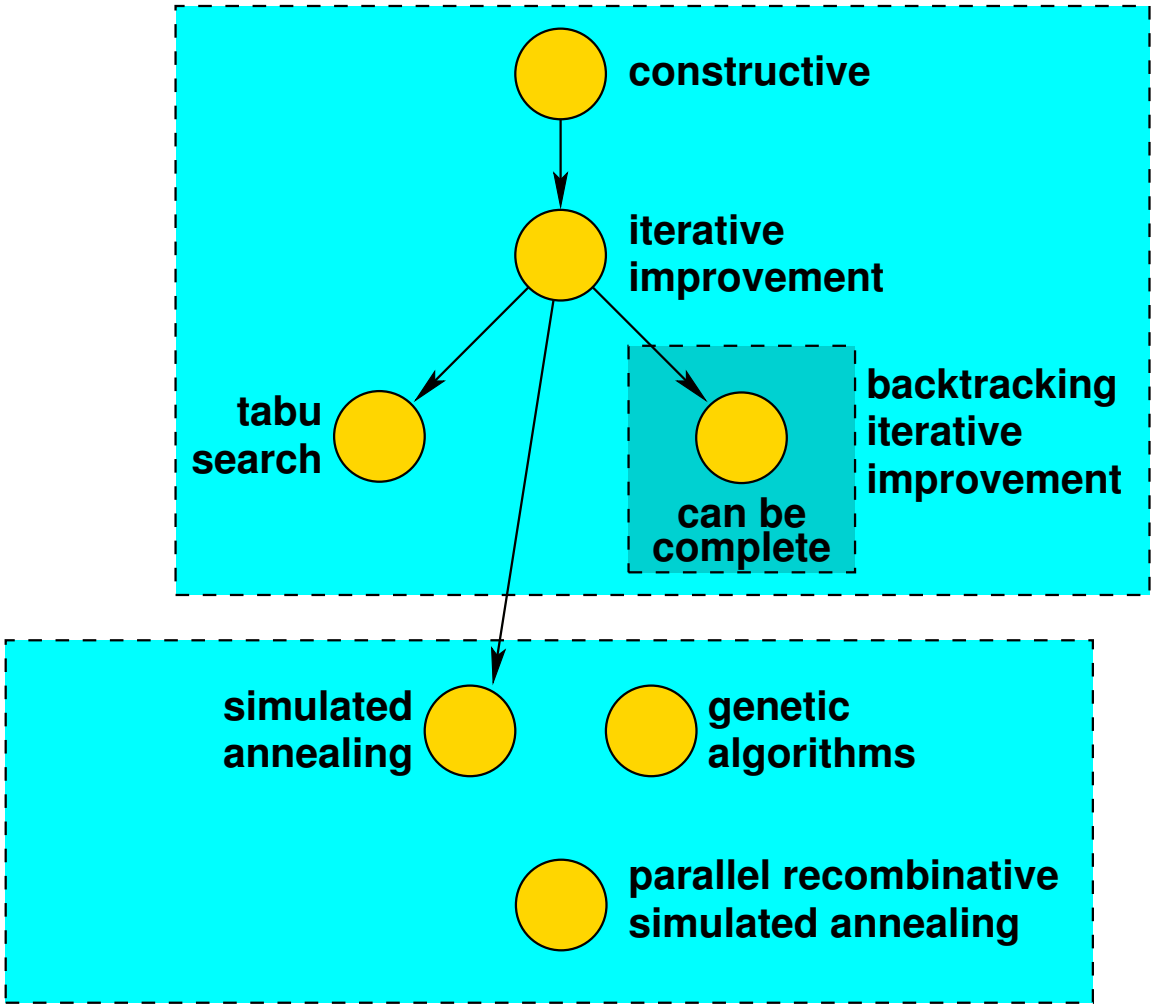
Optimization techniques



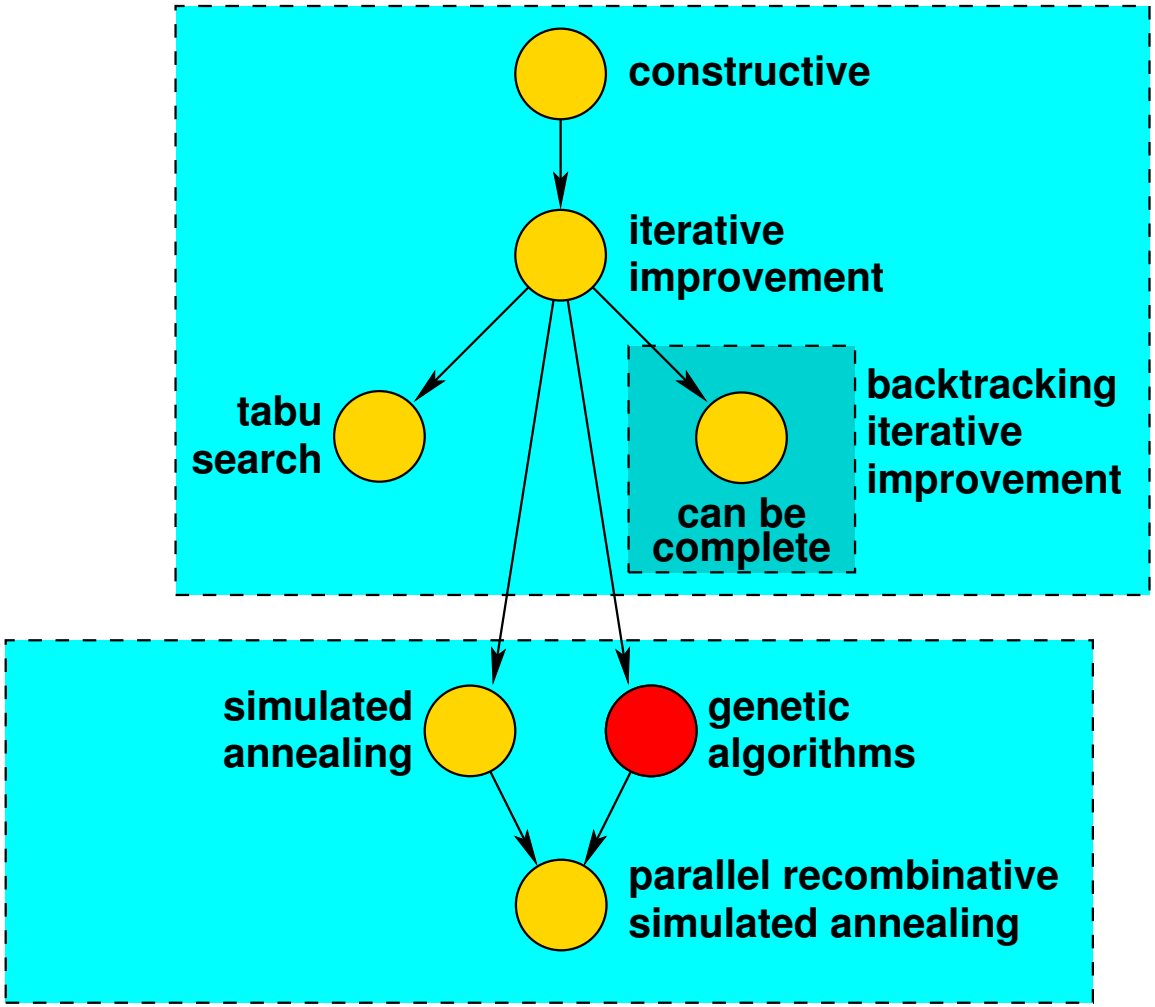
Optimization techniques



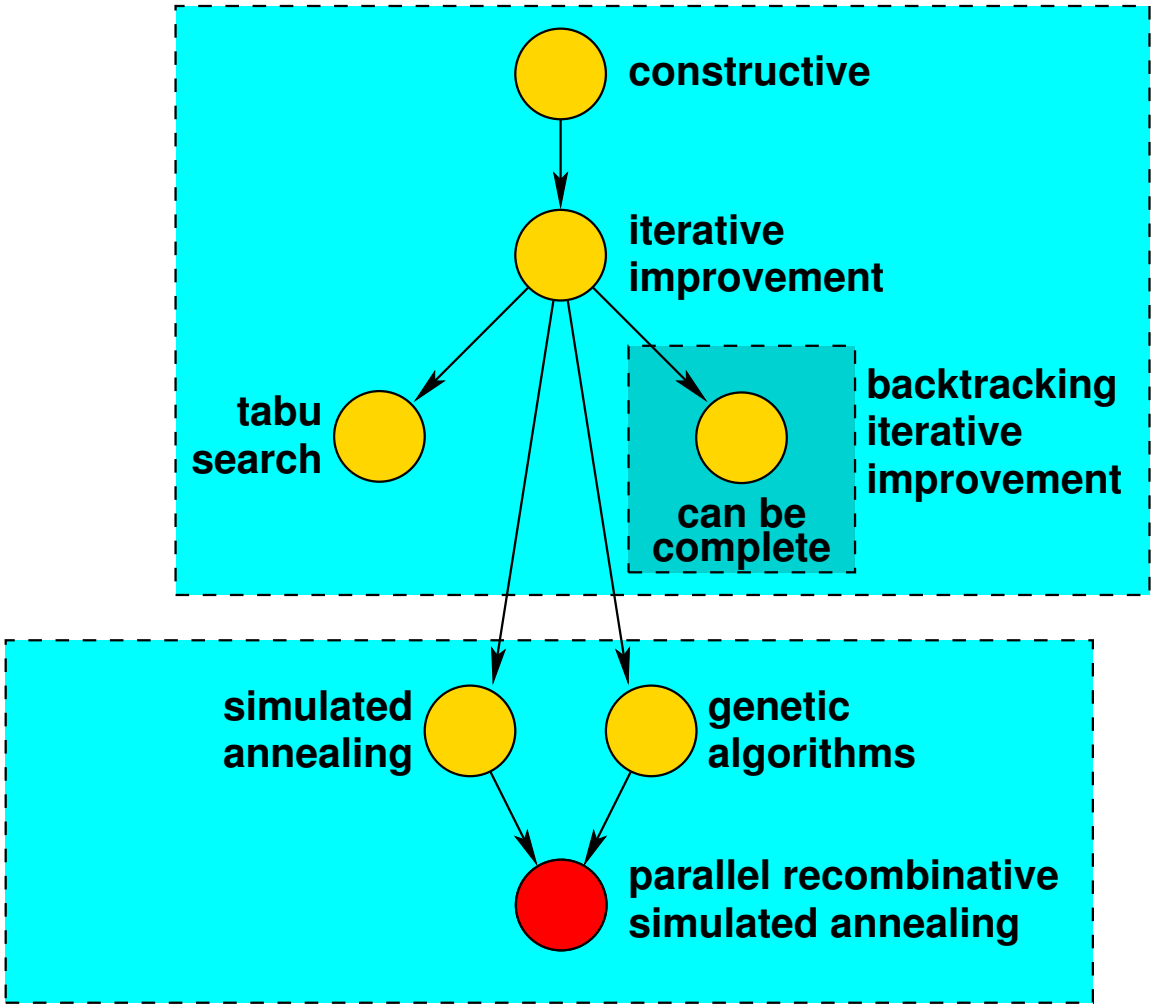
Optimization techniques



Optimization techniques



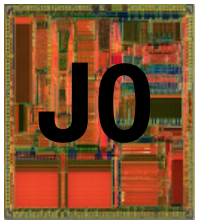
Optimization techniques

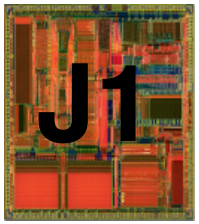


Constructive algorithms

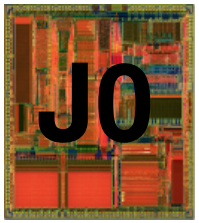
- Build solution piece by piece.
- Once complete solution is generated, don't change.
- Typically fast.
- Easy to use problem-specific information.
- Easy to implement.
- Prone to becomming trapped in poor search space.

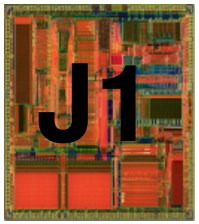
Constructive algorithms example



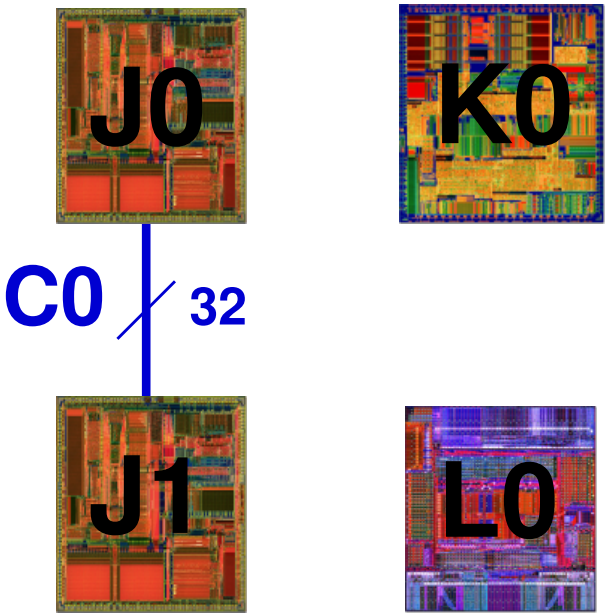


Constructive algorithms example

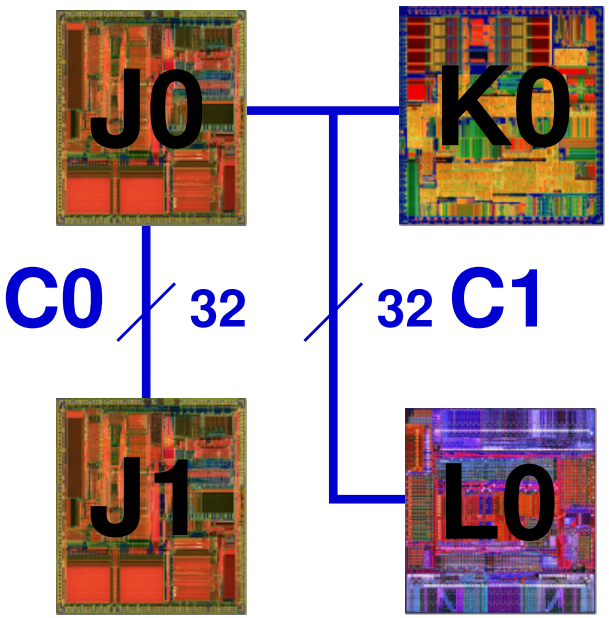




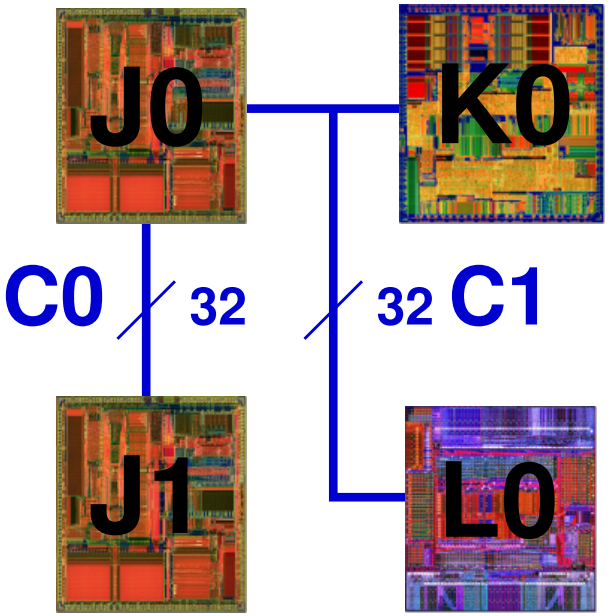
Constructive algorithms example

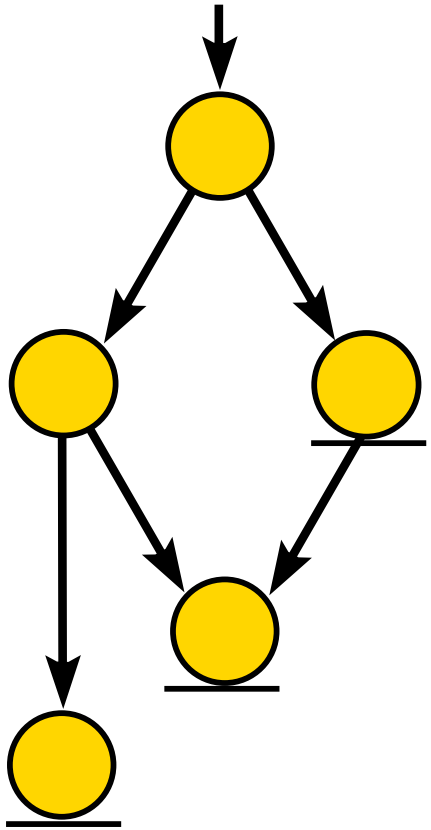


Constructive algorithms example

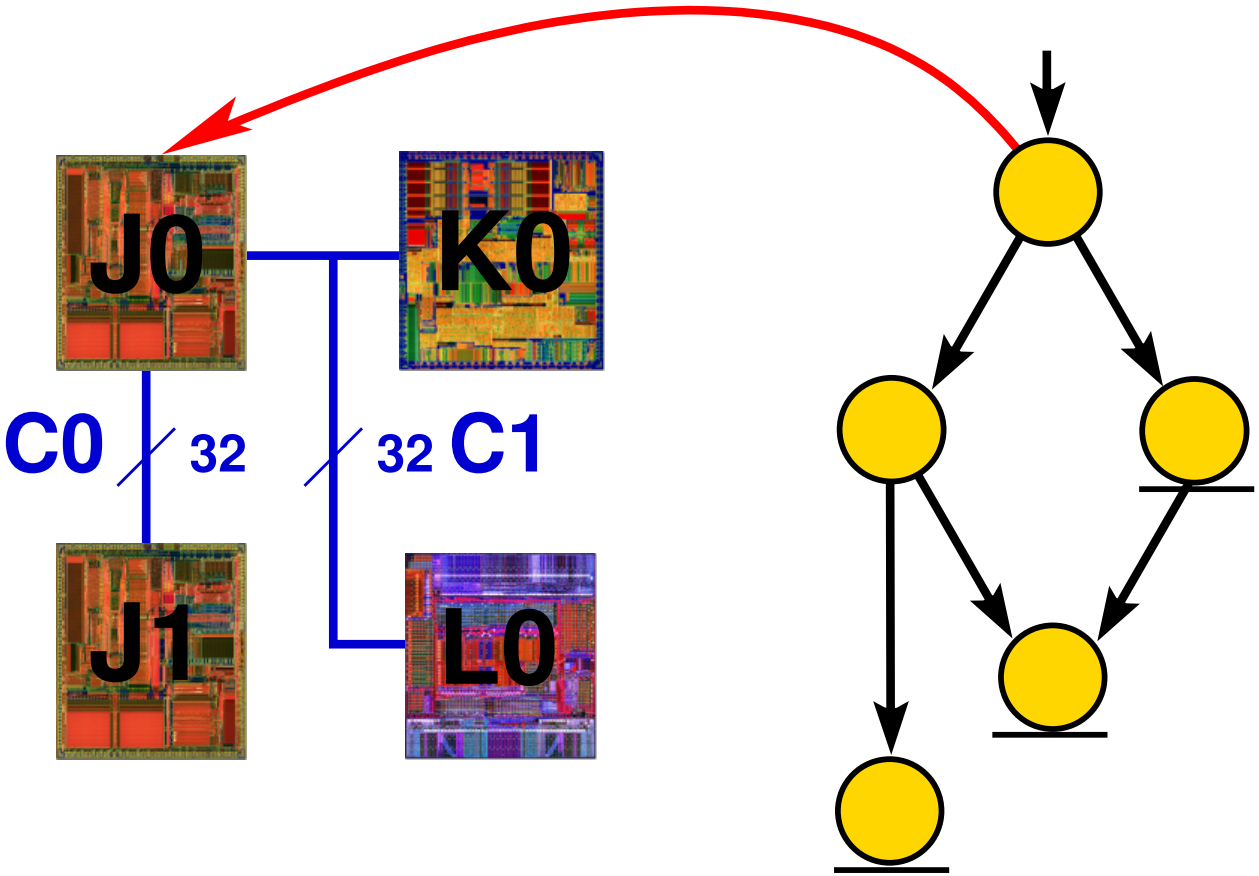


Constructive algorithms example

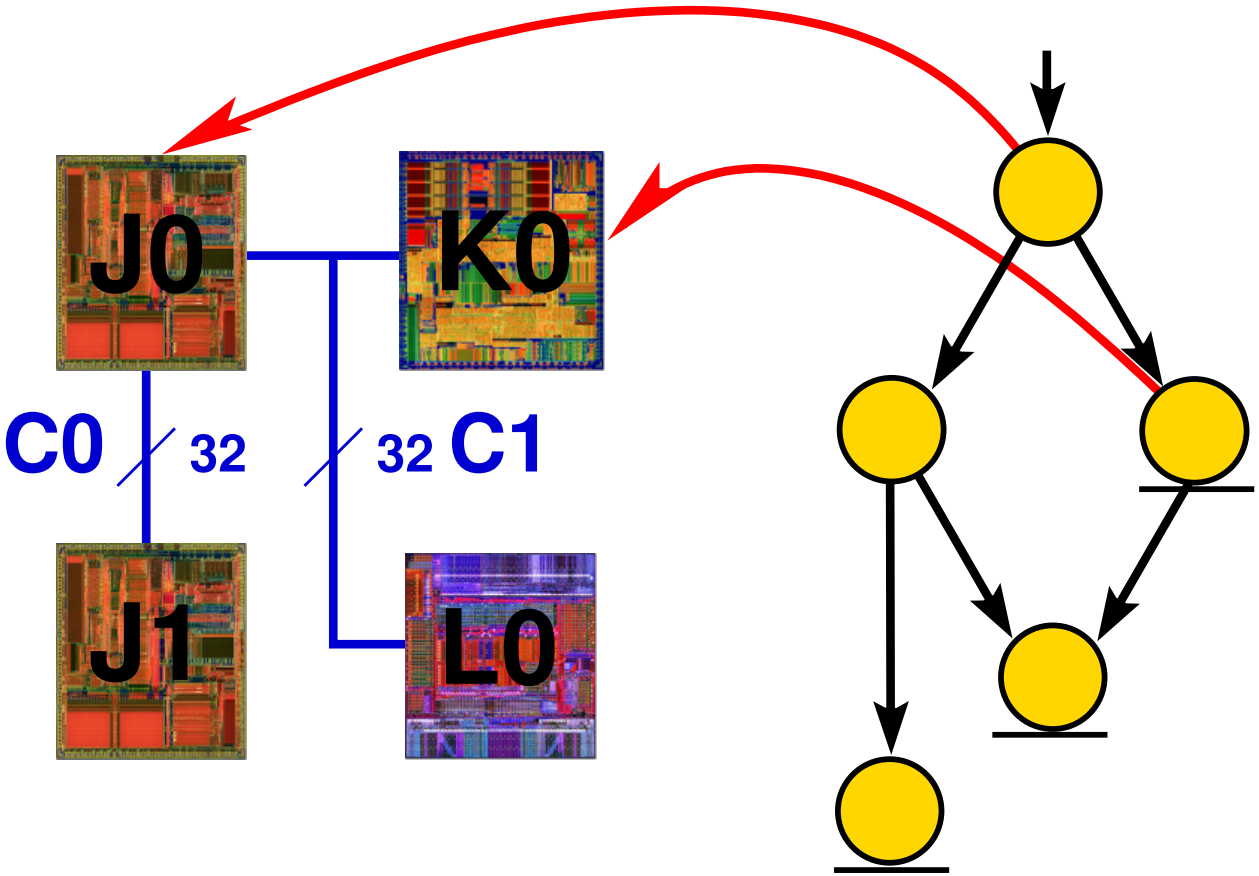




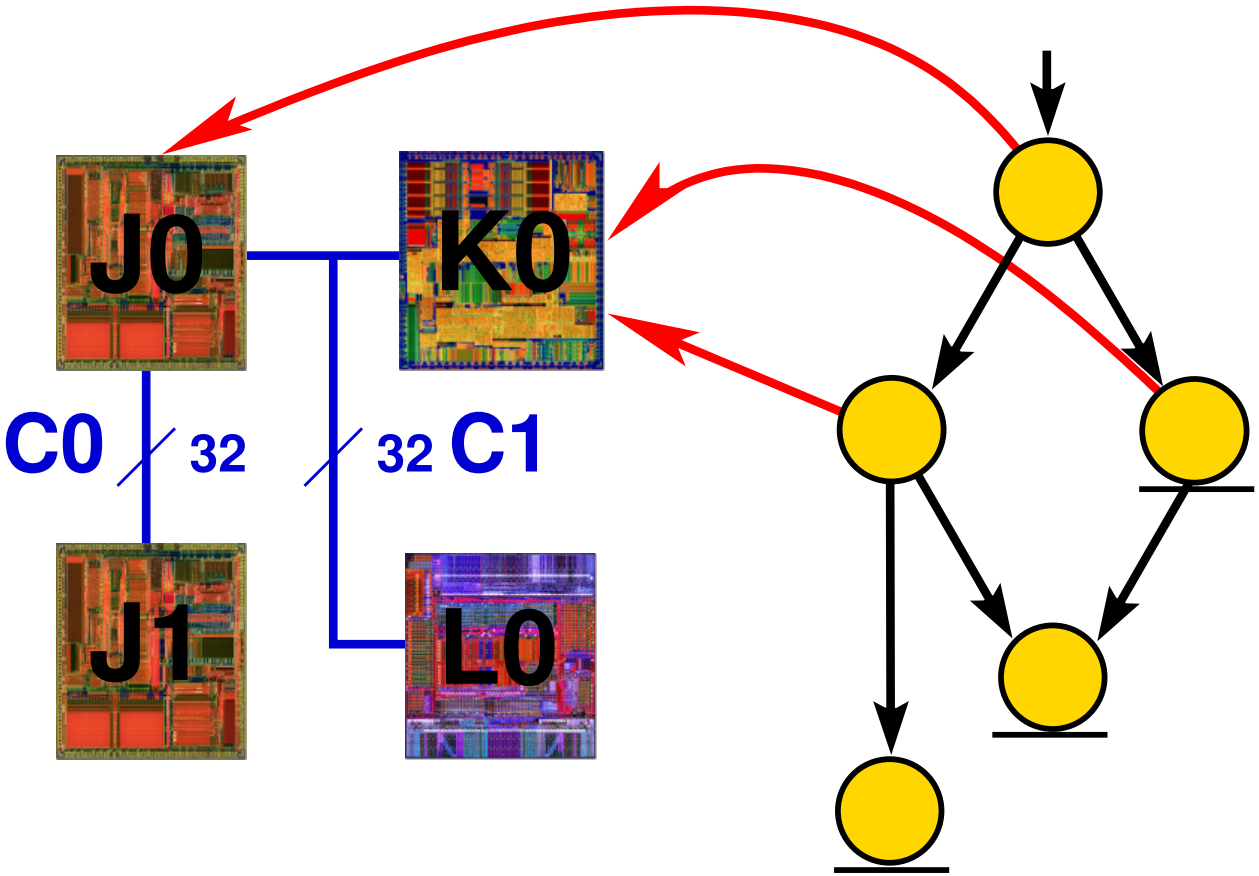
Constructive algorithms example



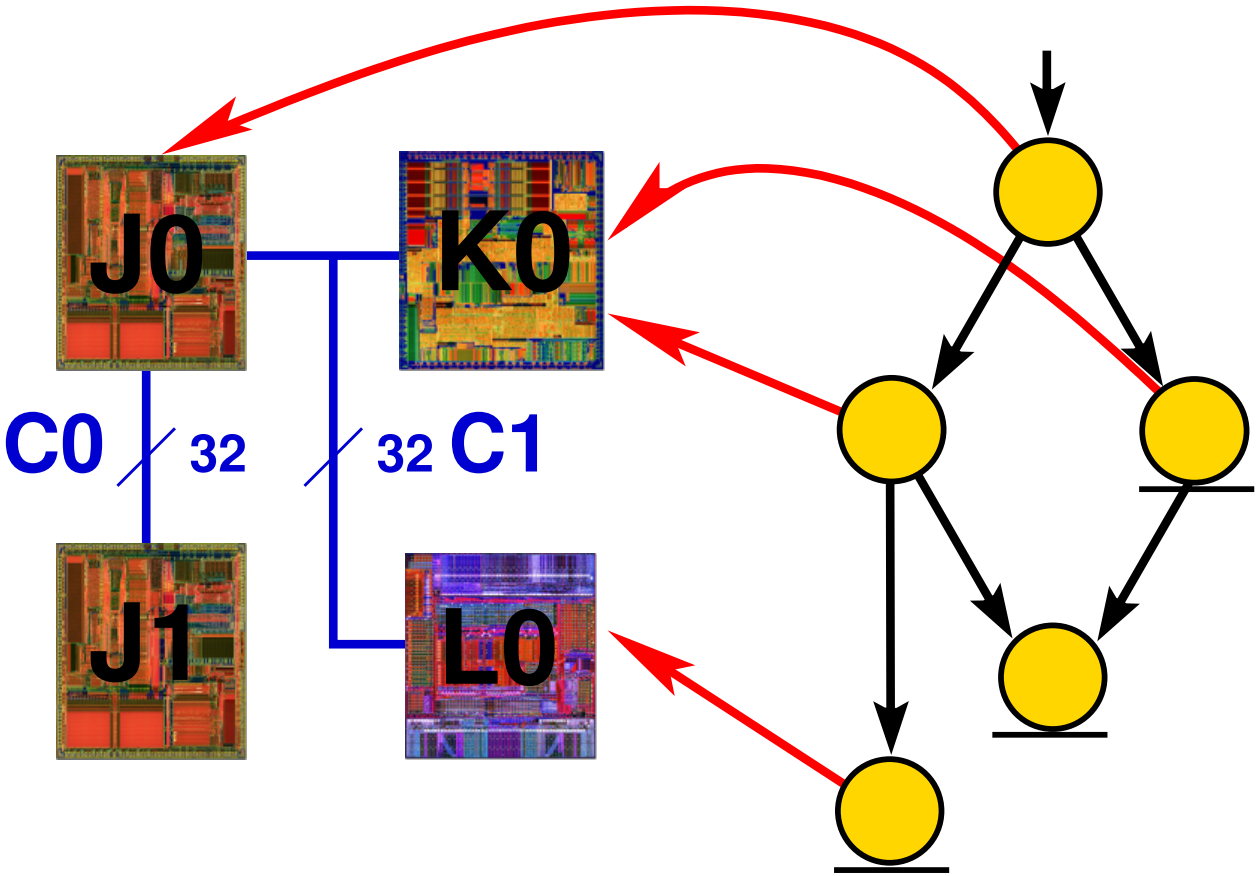
Constructive algorithms example



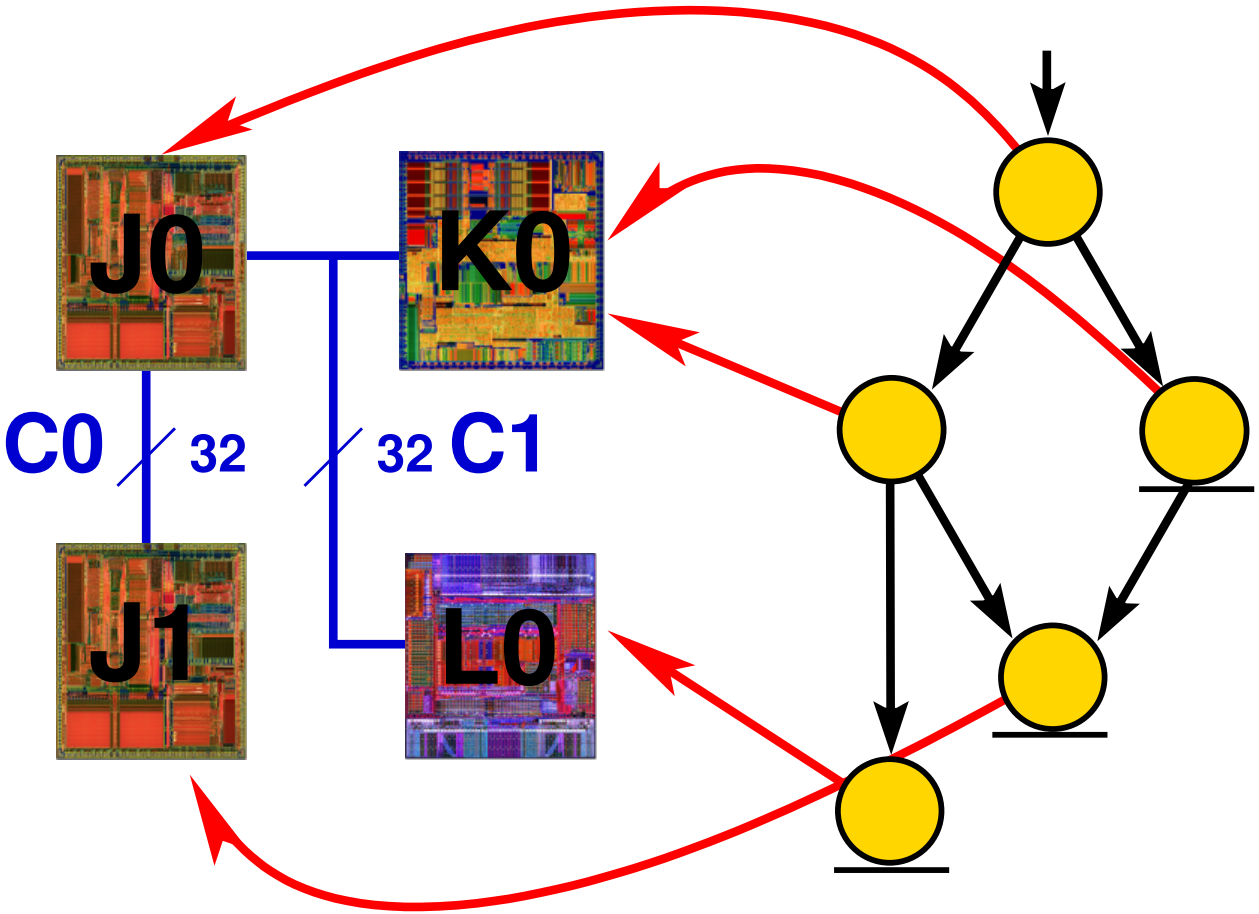
Constructive algorithms example



Constructive algorithms example



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.

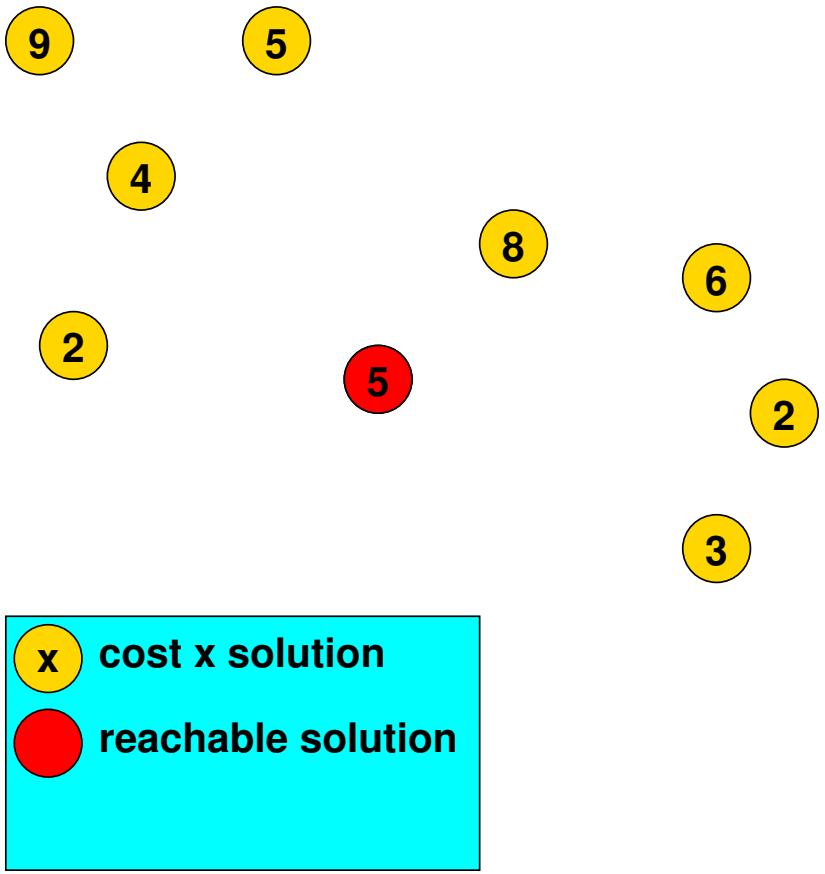
Typically fast.

Easy to use problem-specific information.

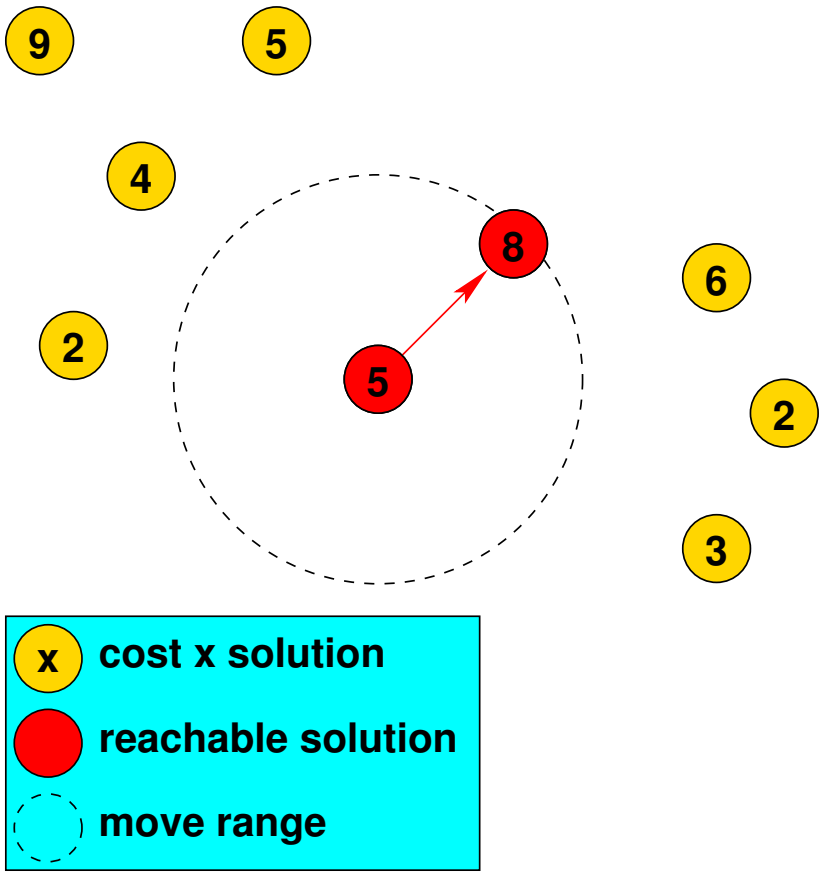
Easy to implement.

Prone to becoming trapped in local minima.

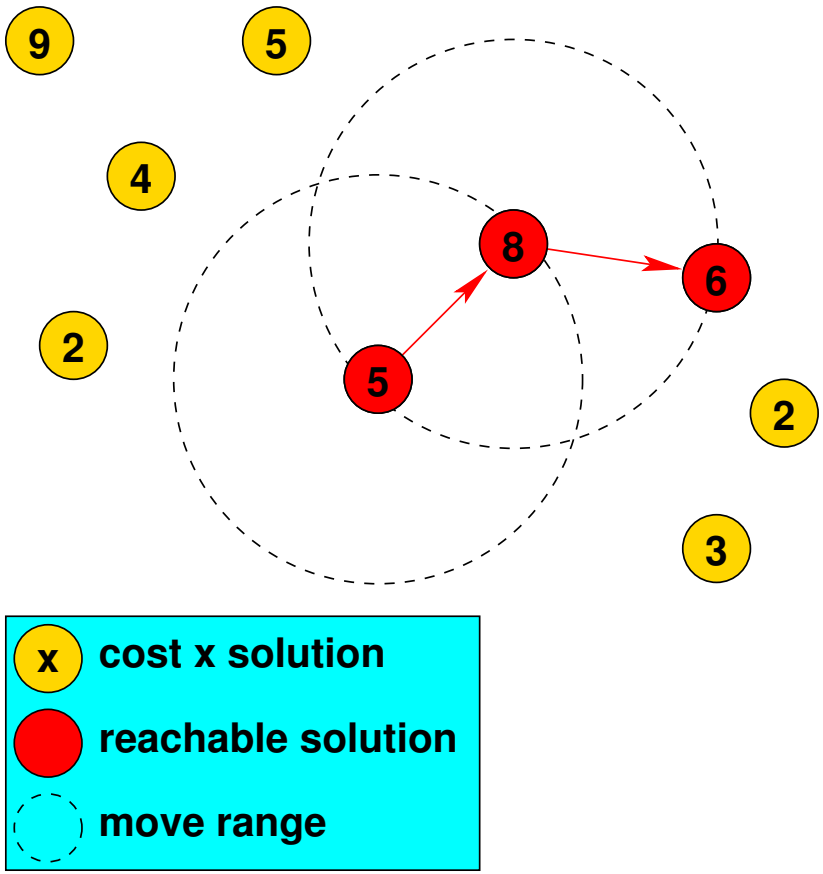
Local minima move size



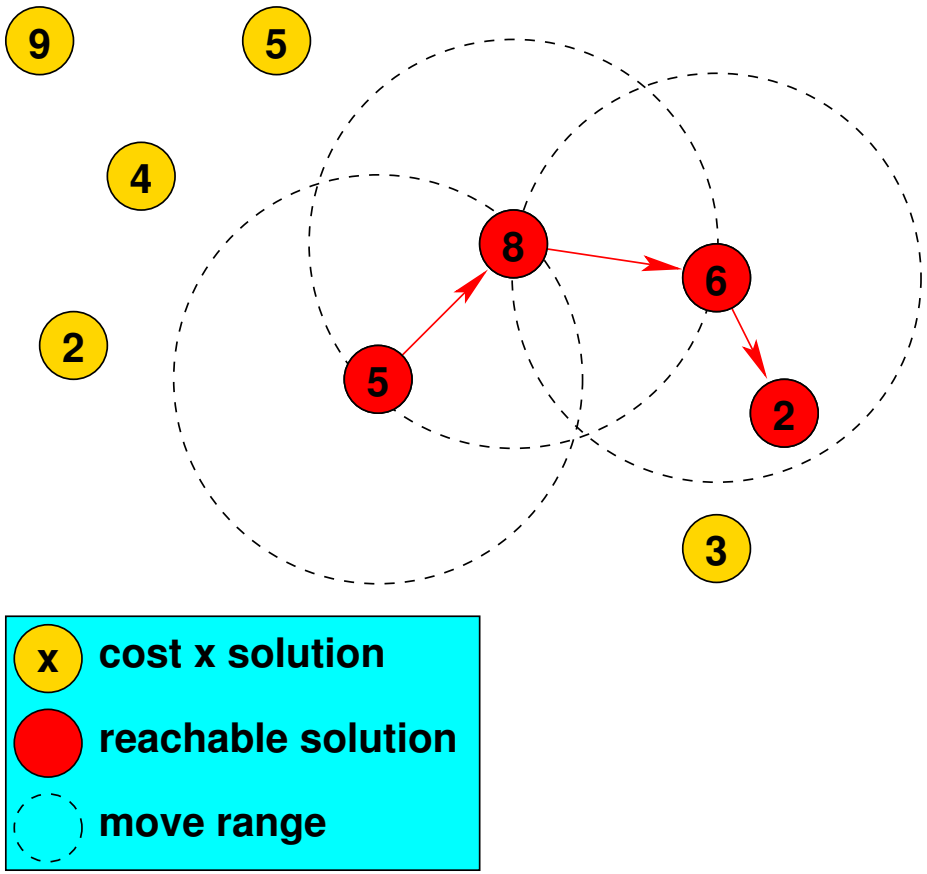
Local minima move size



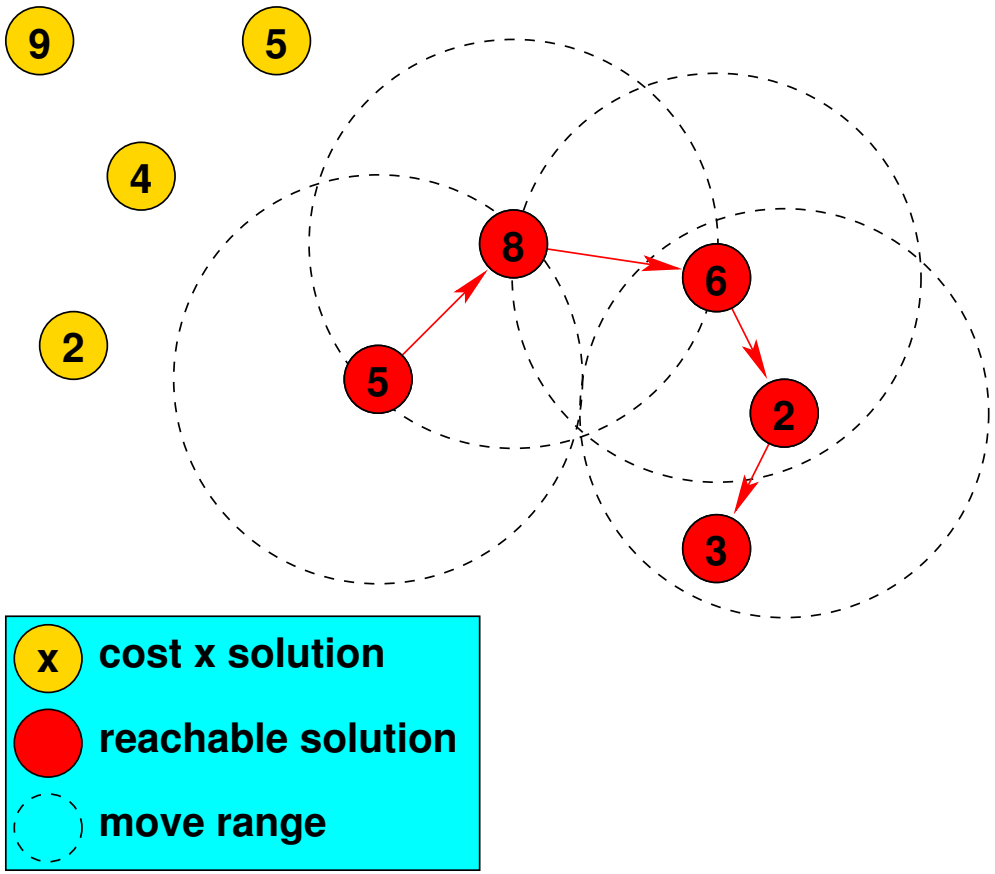
Local minima move size



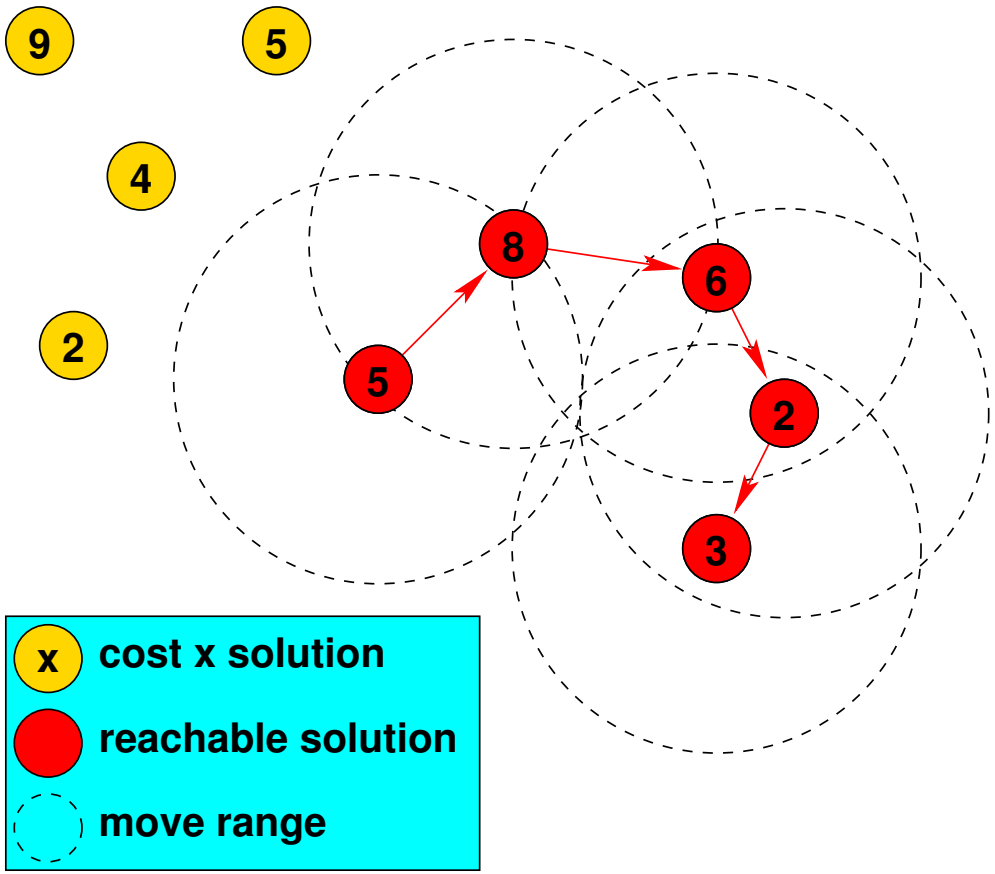
Local minima move size



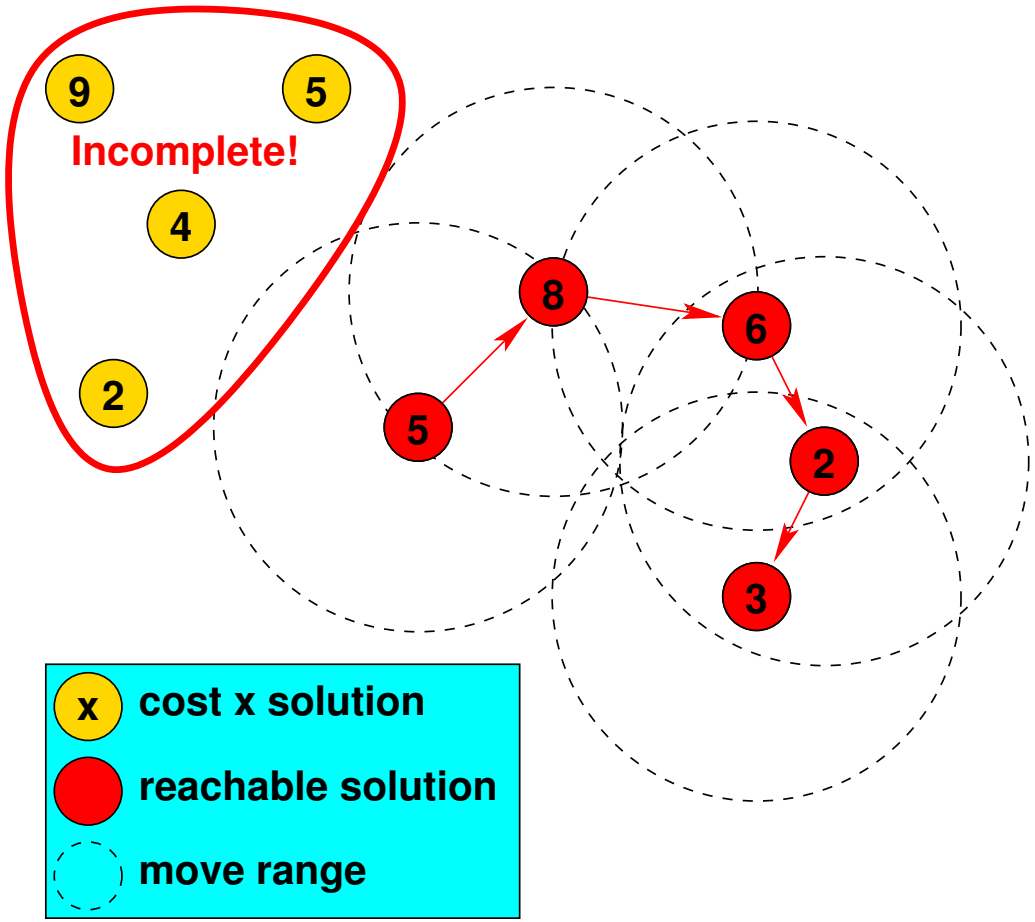
Local minima move size



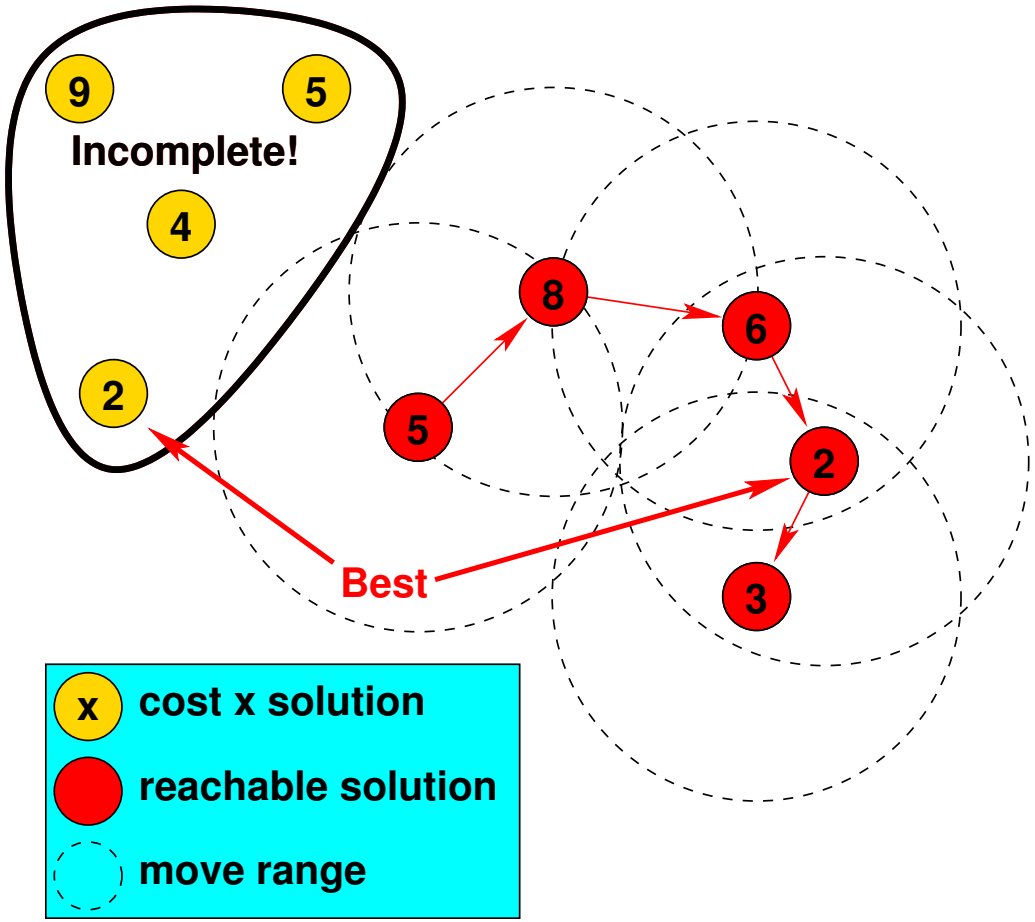
Local minima move size



Local minima move size



Local minima move size

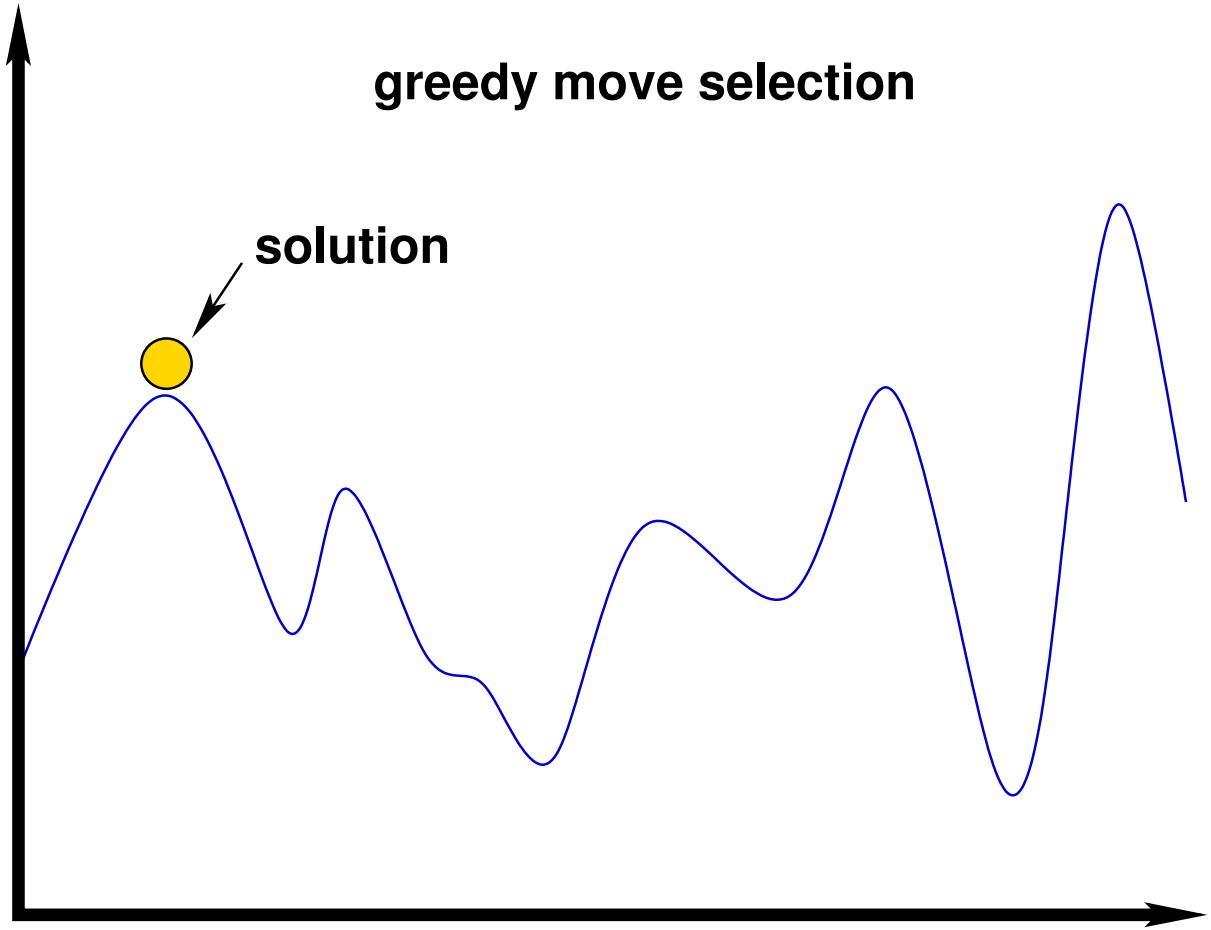


Local minima

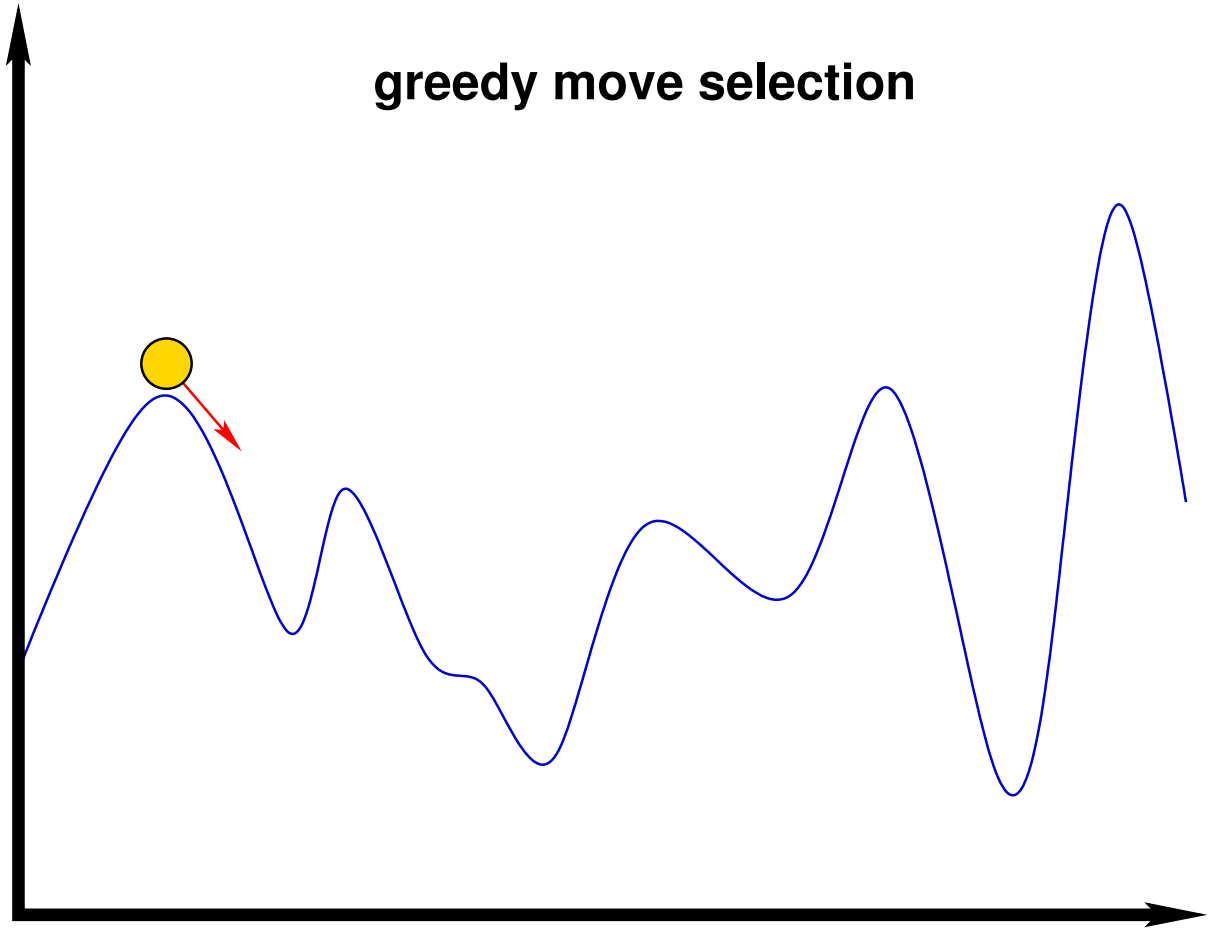
Even if all solutions reachable, may not get best solution.

Depends on move selection.

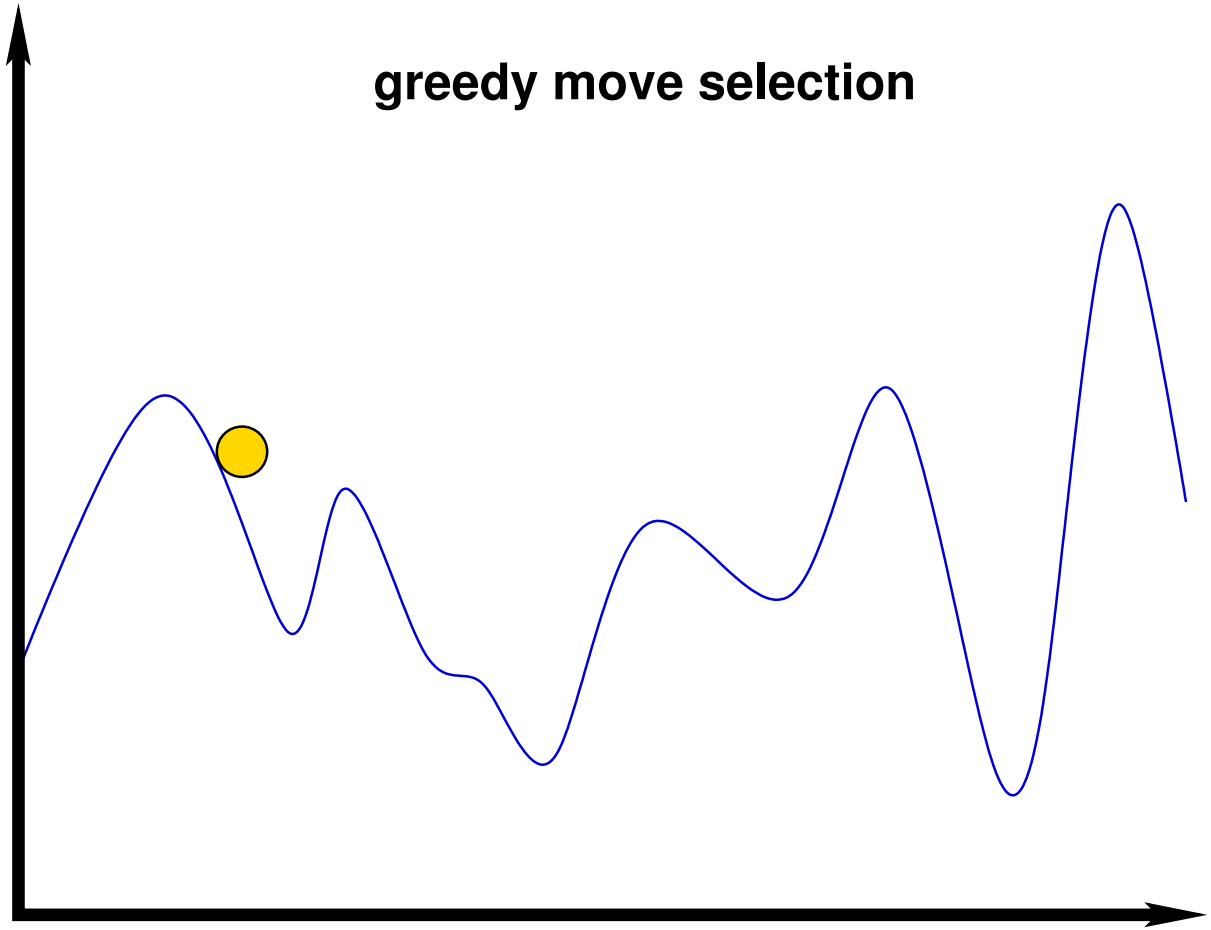
Trapped in local minima



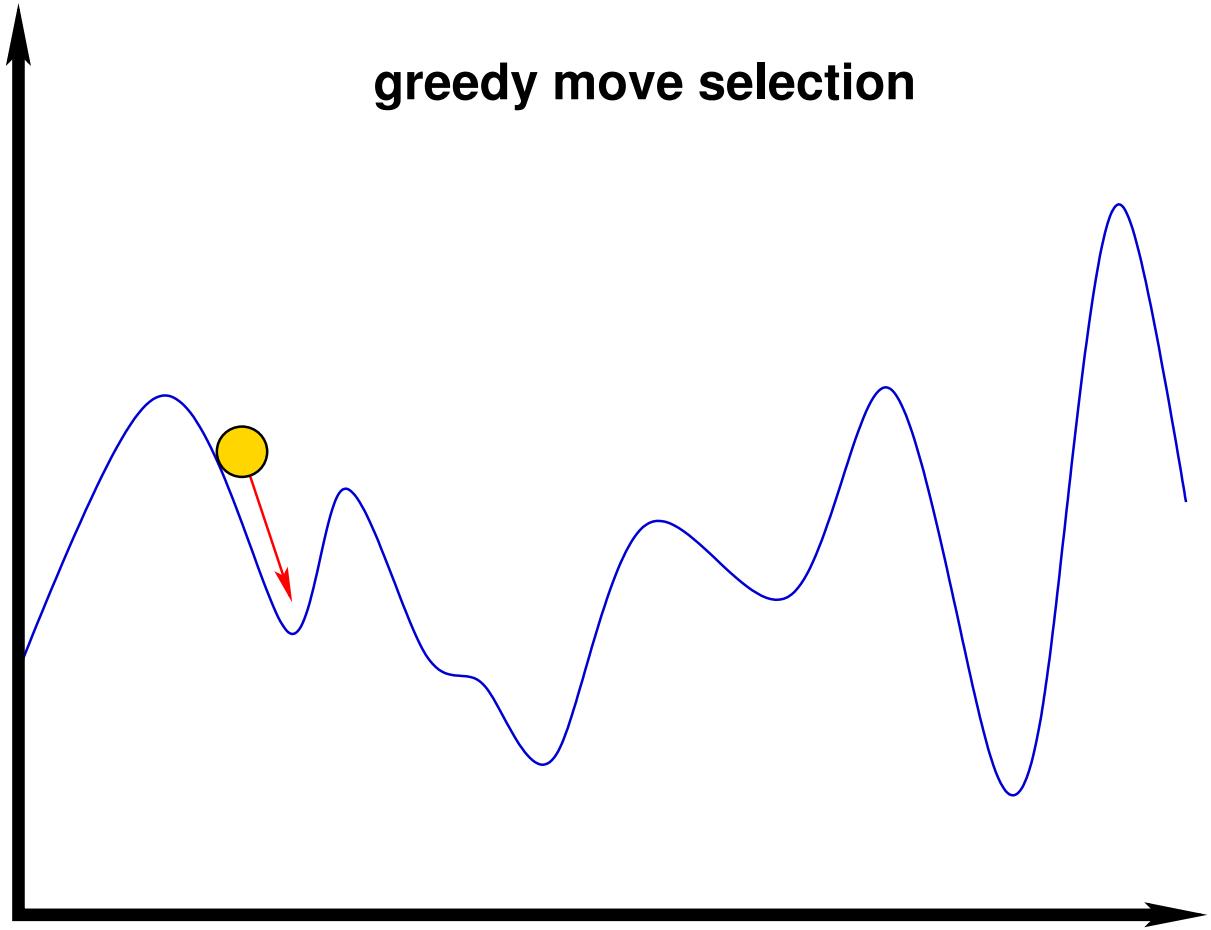
Trapped in local minima



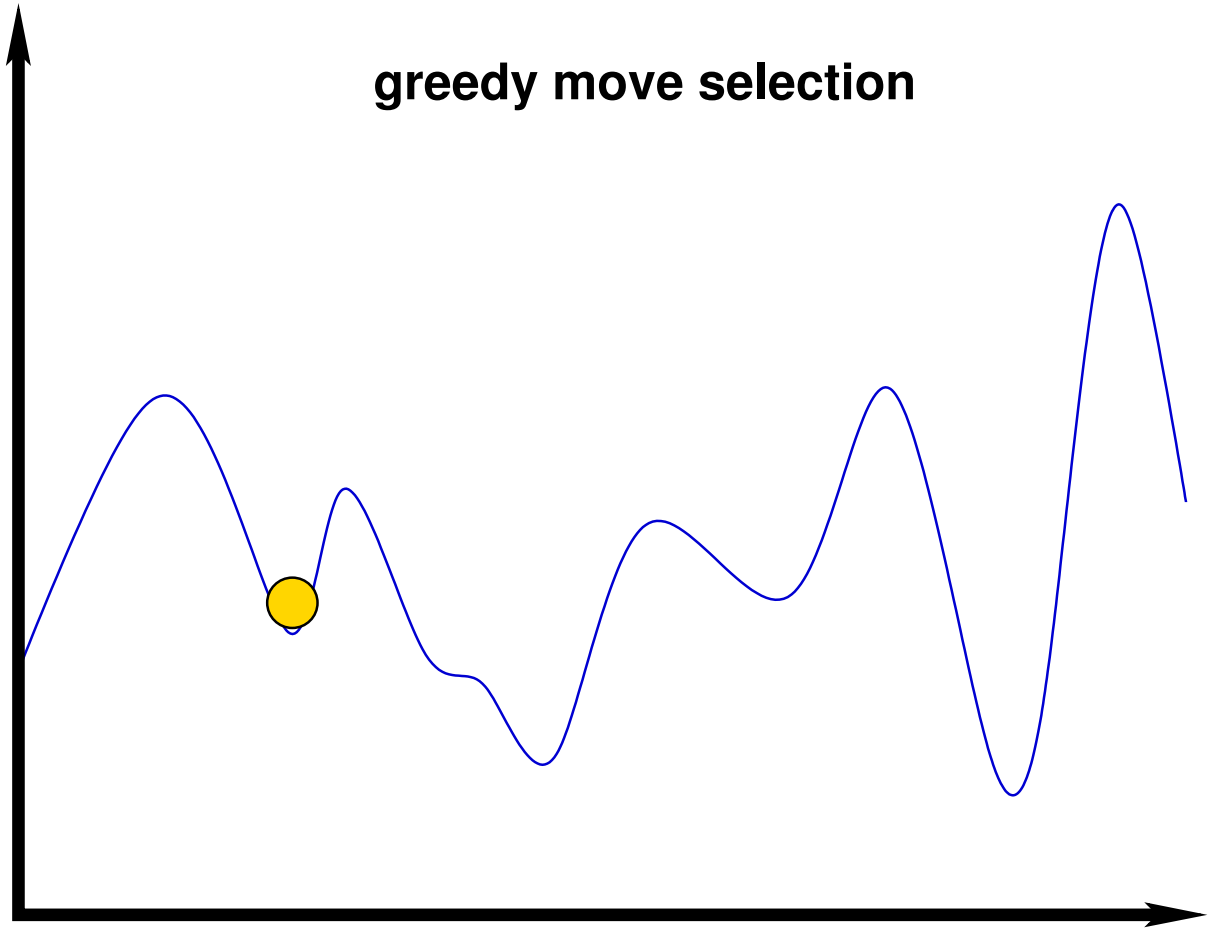
Trapped in local minima



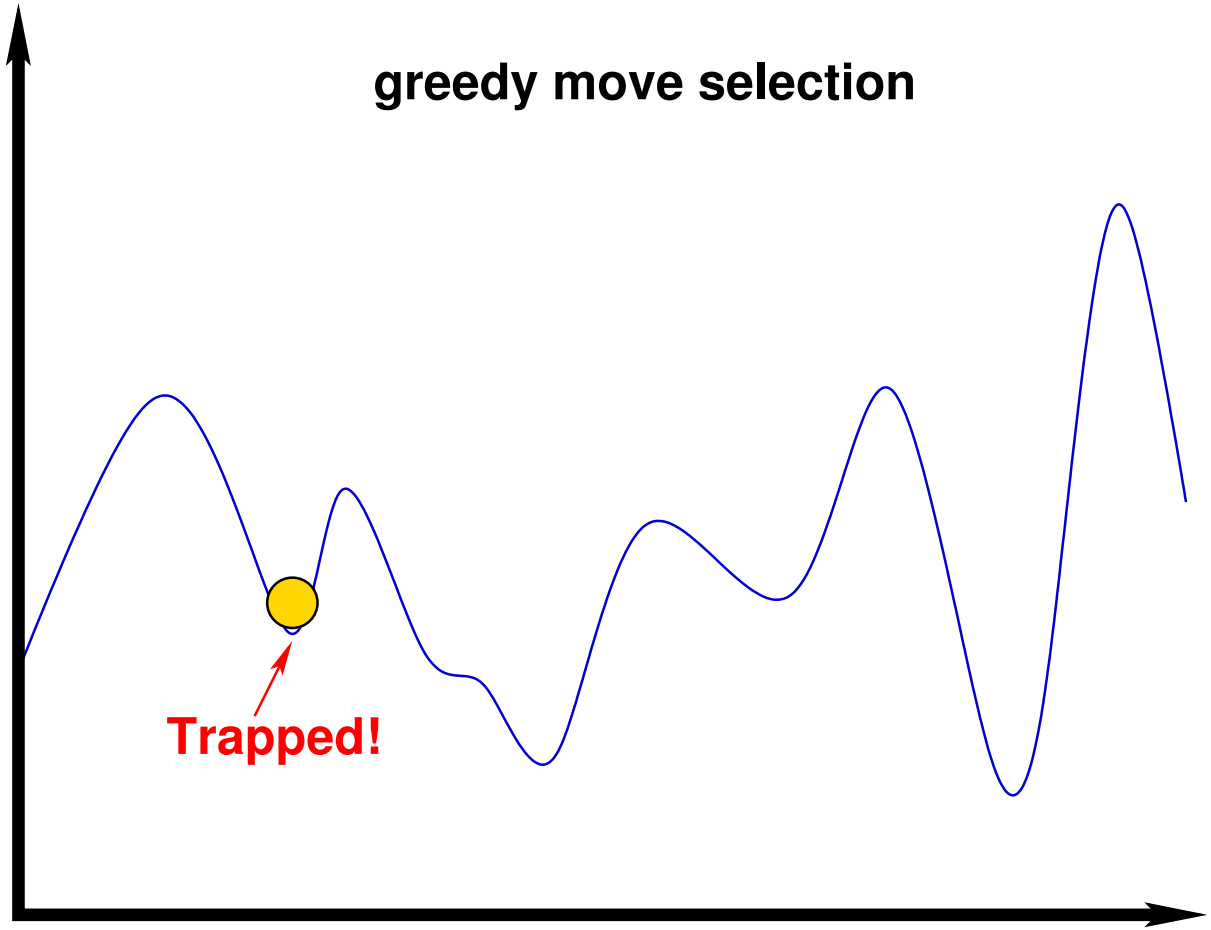
Trapped in local minima



Trapped in local minima



Trapped in local minima



Local mimina

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

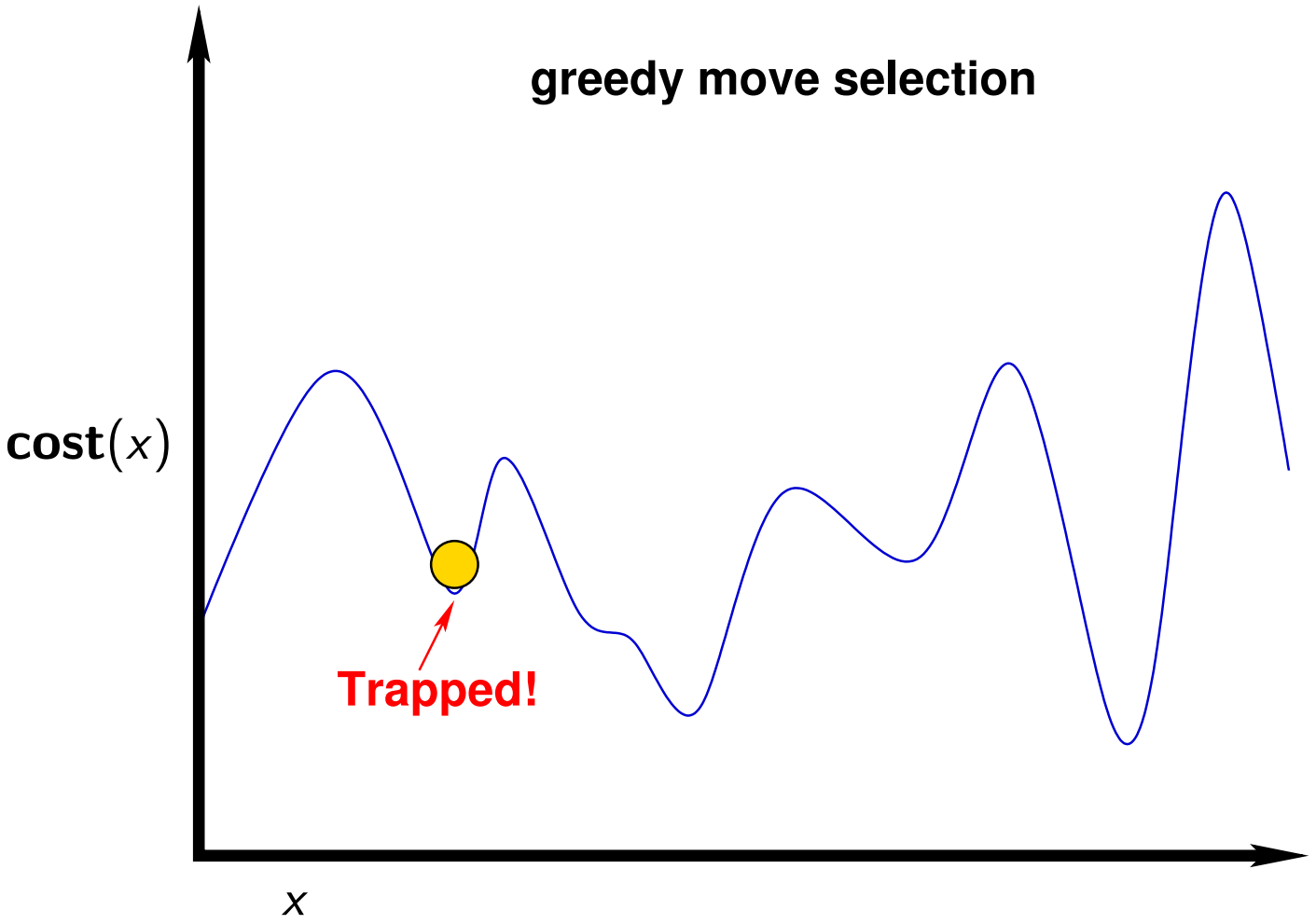
- All solutions are reachable.
- The backtracking depth $\geq$ search depth.
- . . . however, this can be slow.

Even if incomplete, backtracking can improve quality.

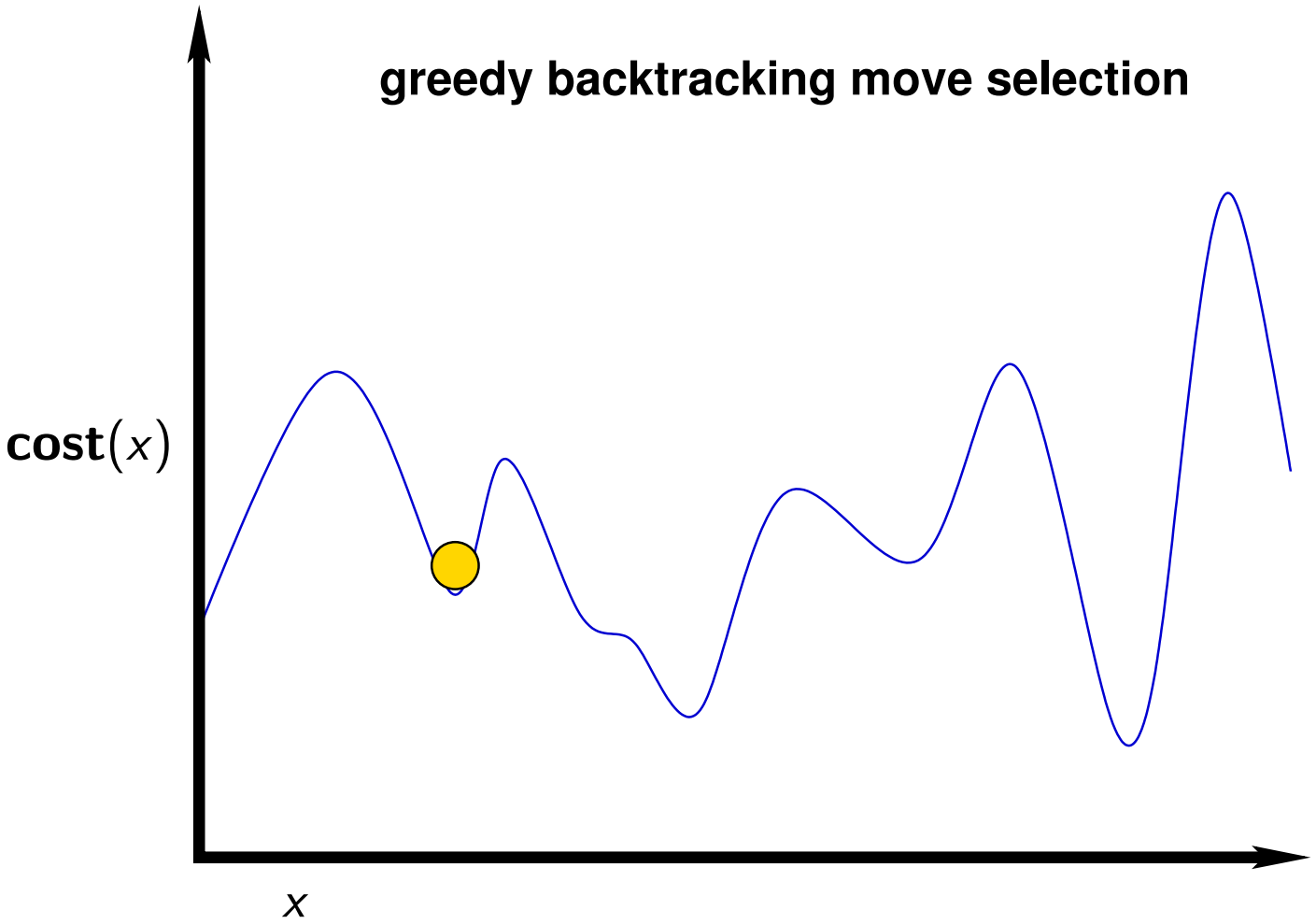
Can trade optimization time for solution quality.

Greedy iterative improvement if backtracking depth is zero.

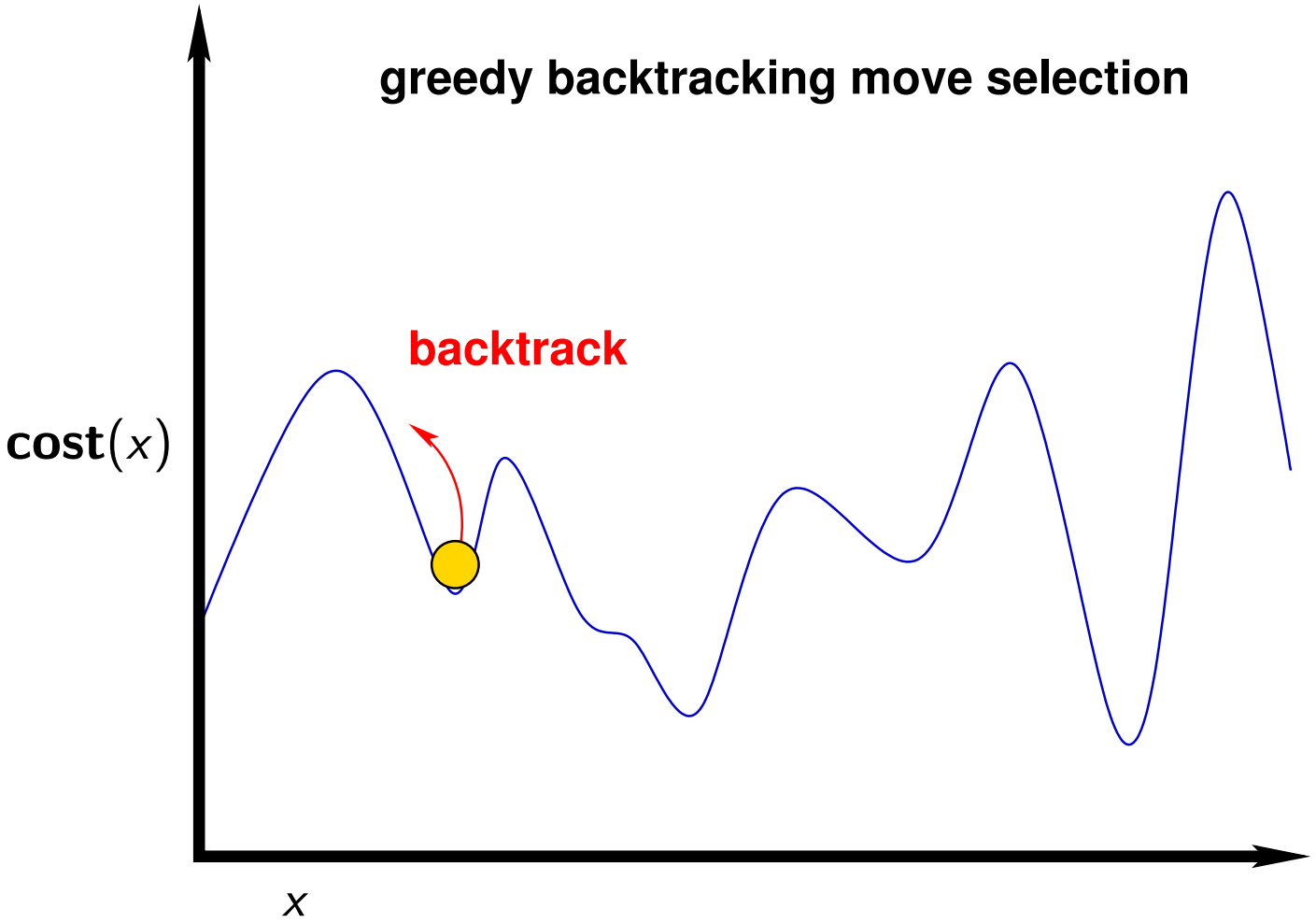
Backtracking



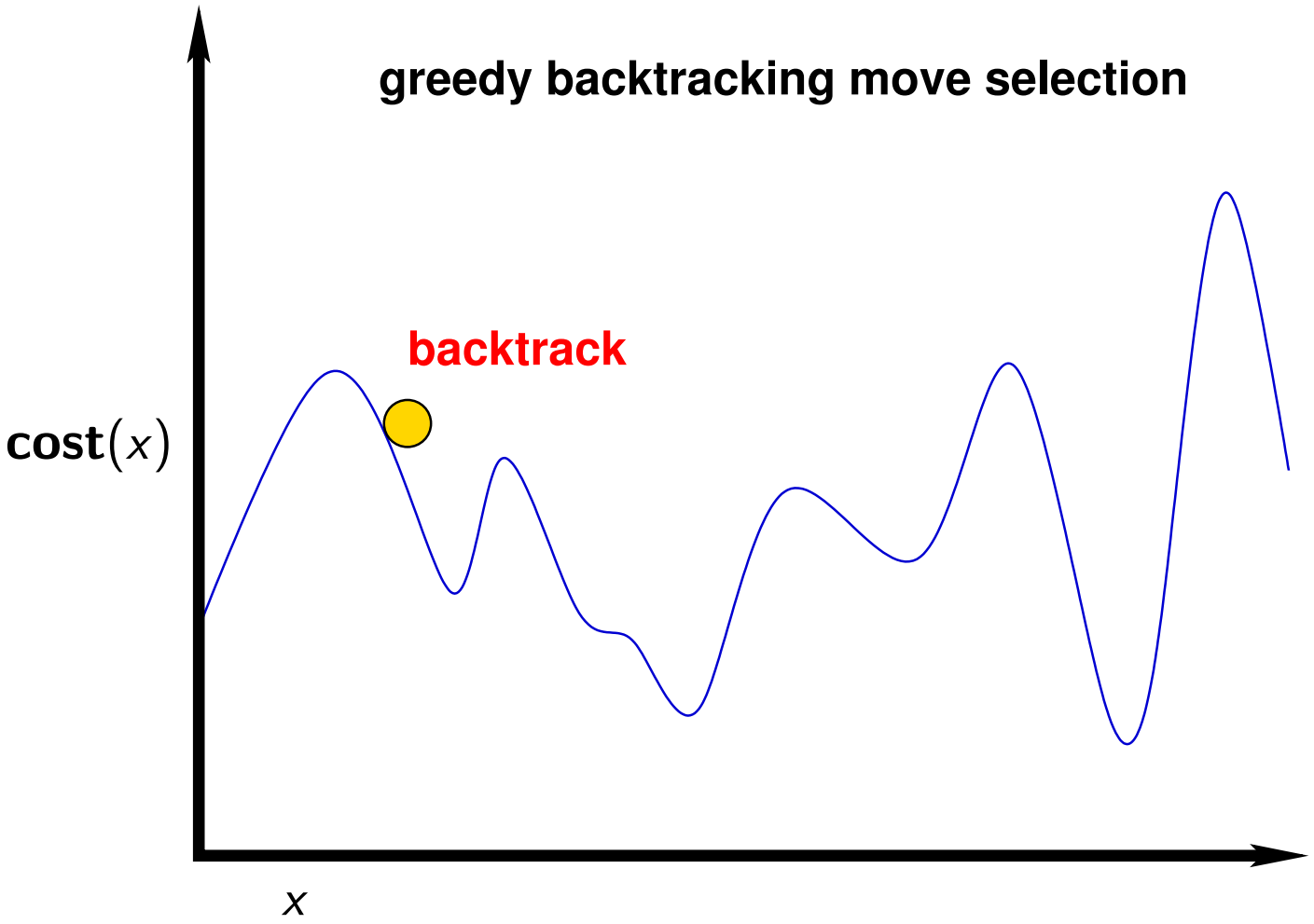
Backtracking



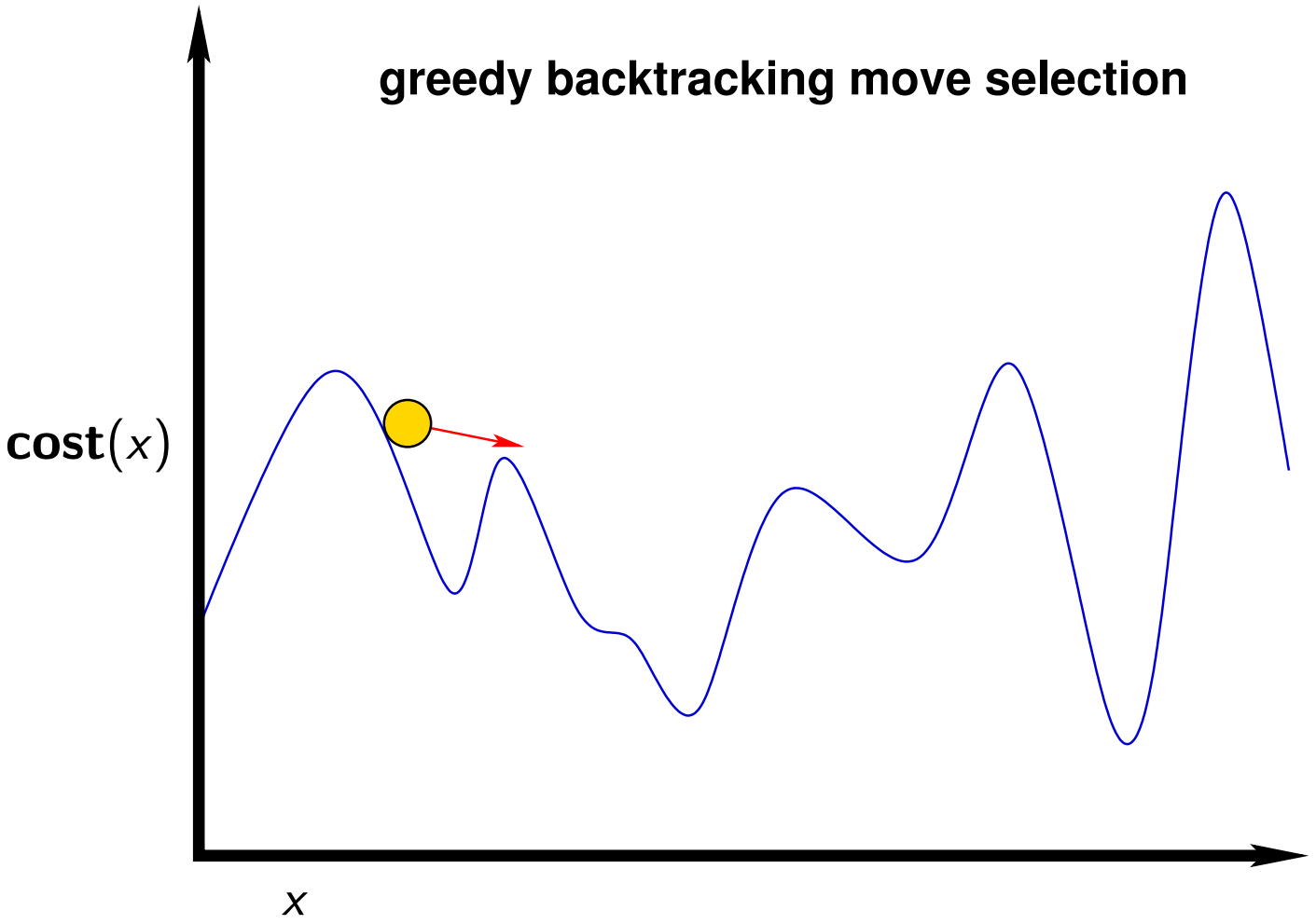
Backtracking



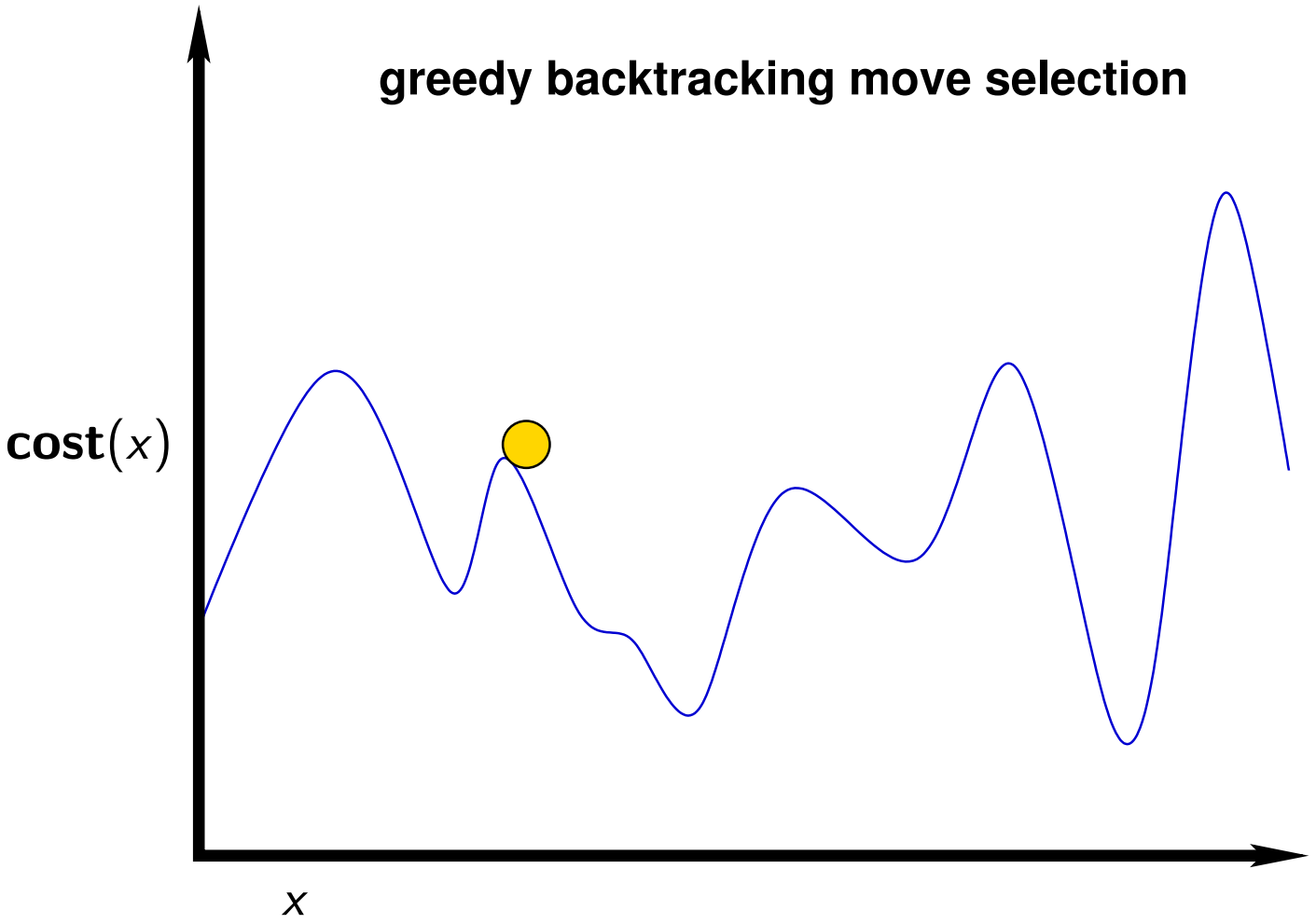
Backtracking



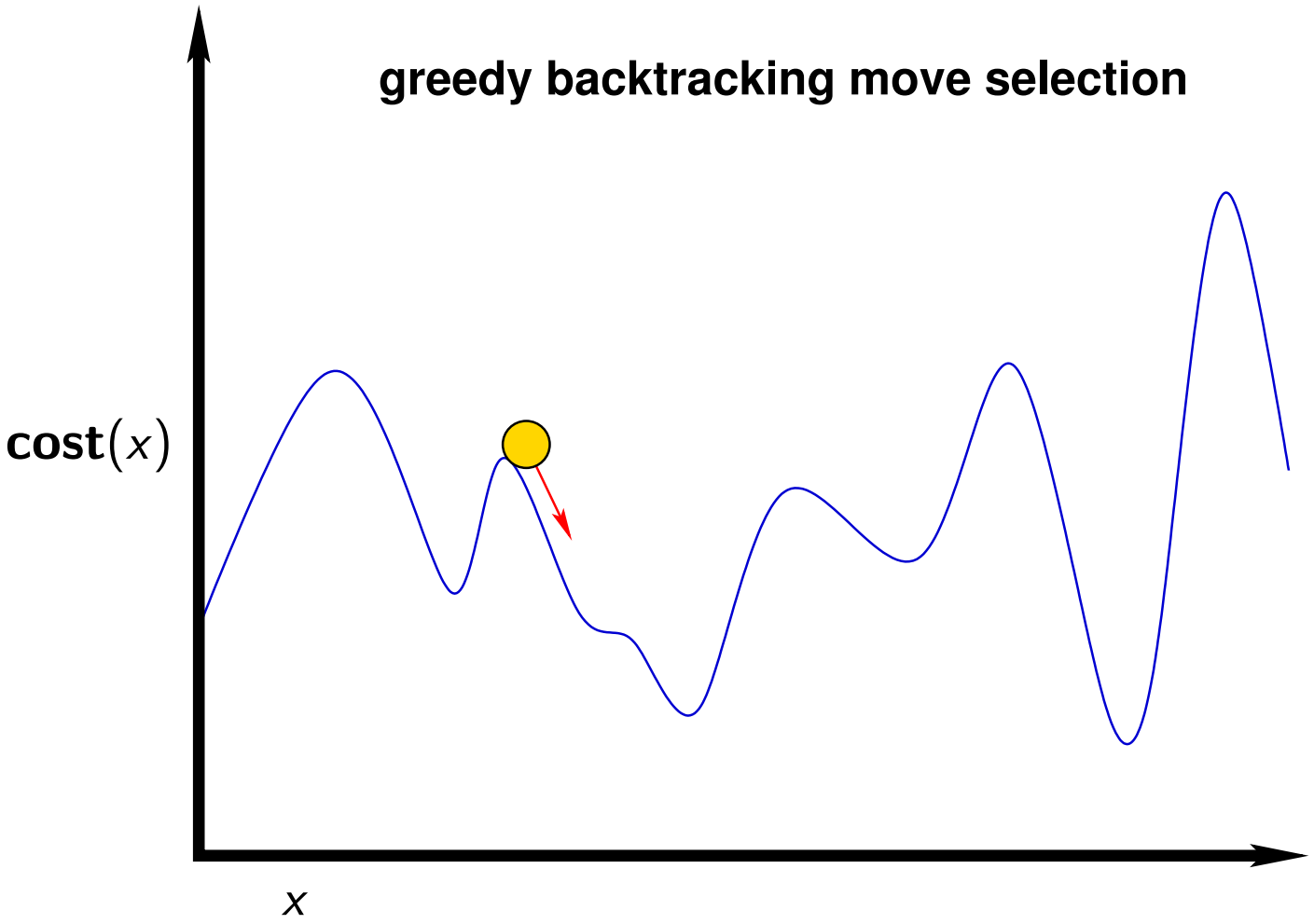
Backtracking



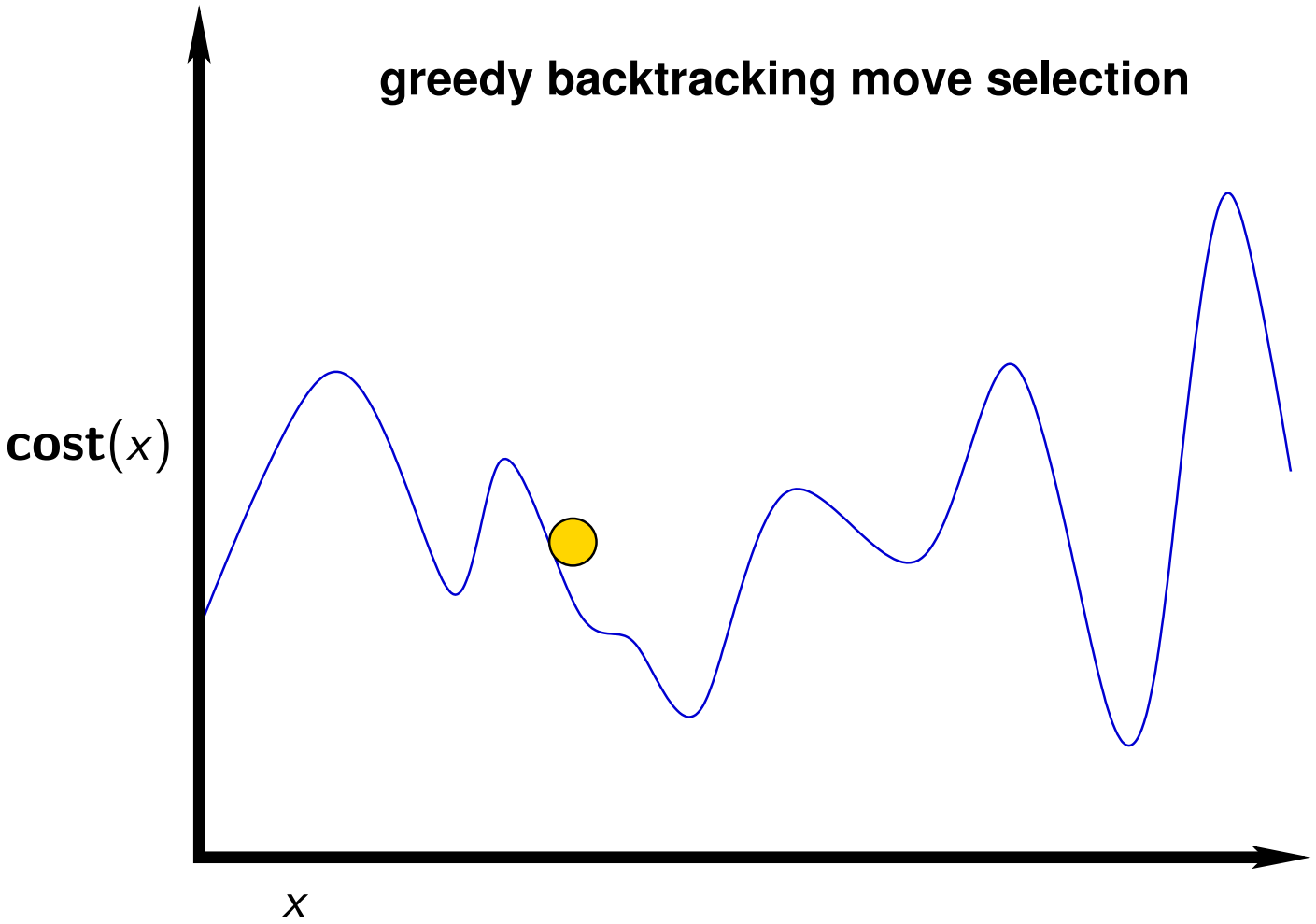
Backtracking



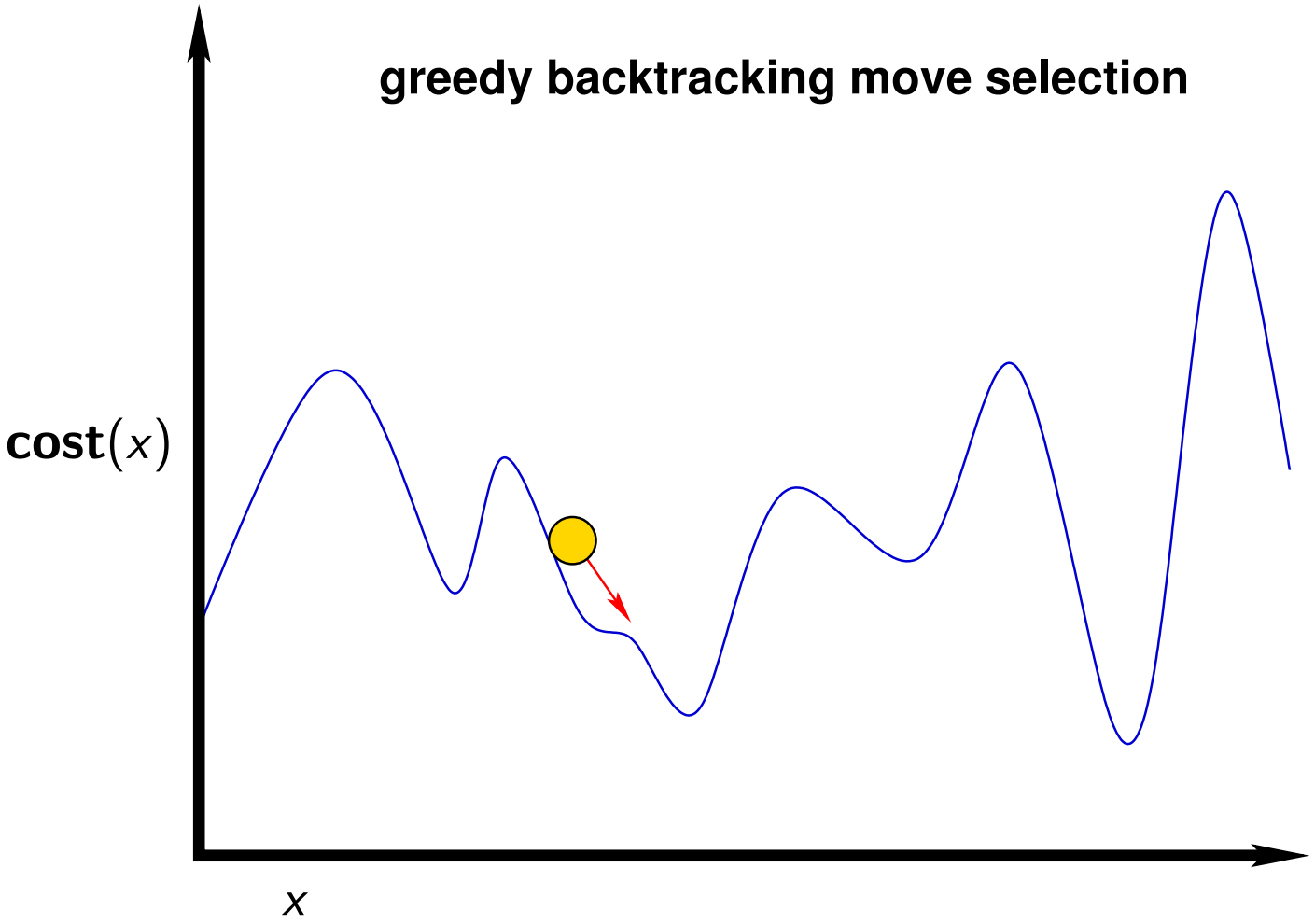
Backtracking



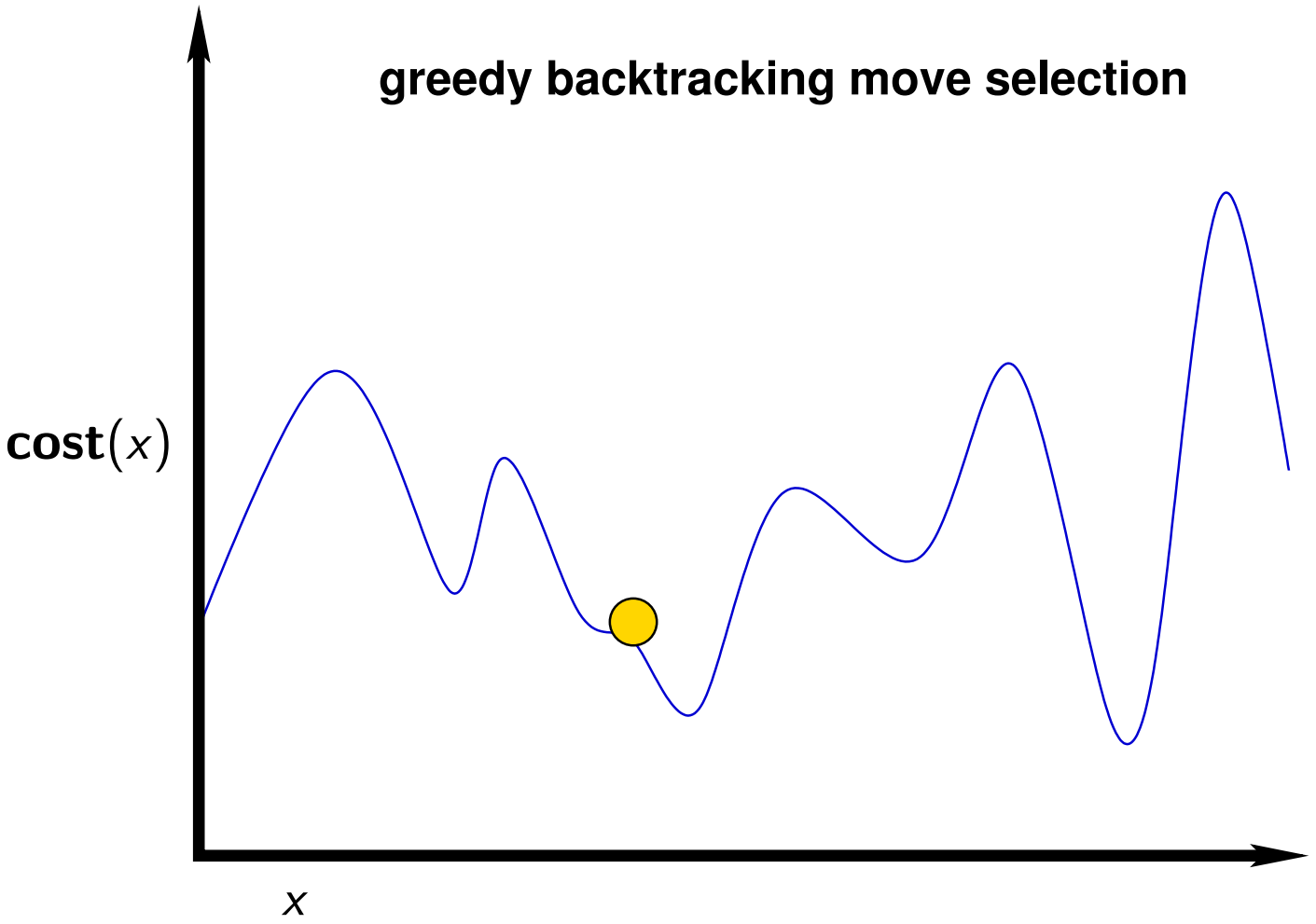
Backtracking



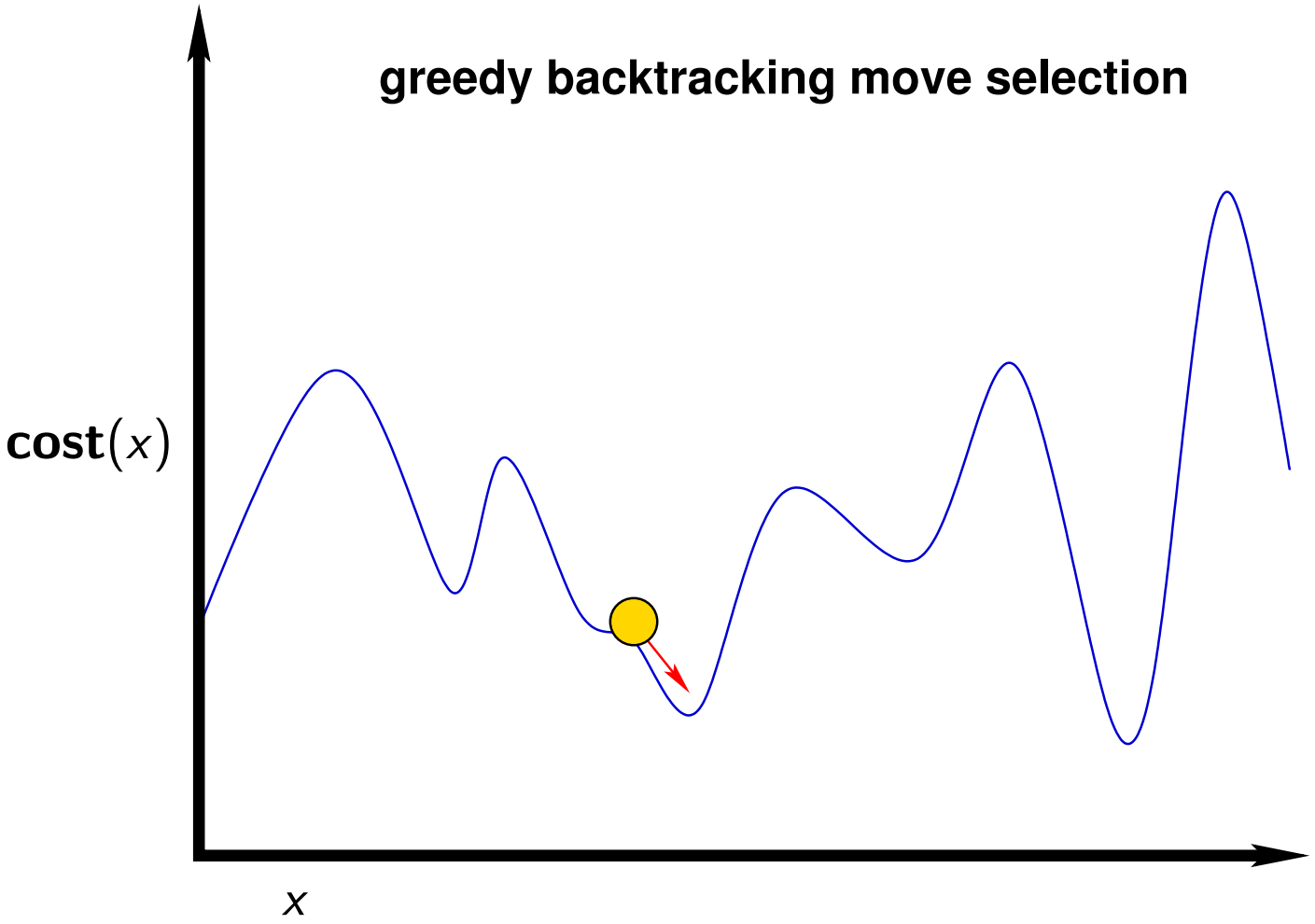
Backtracking



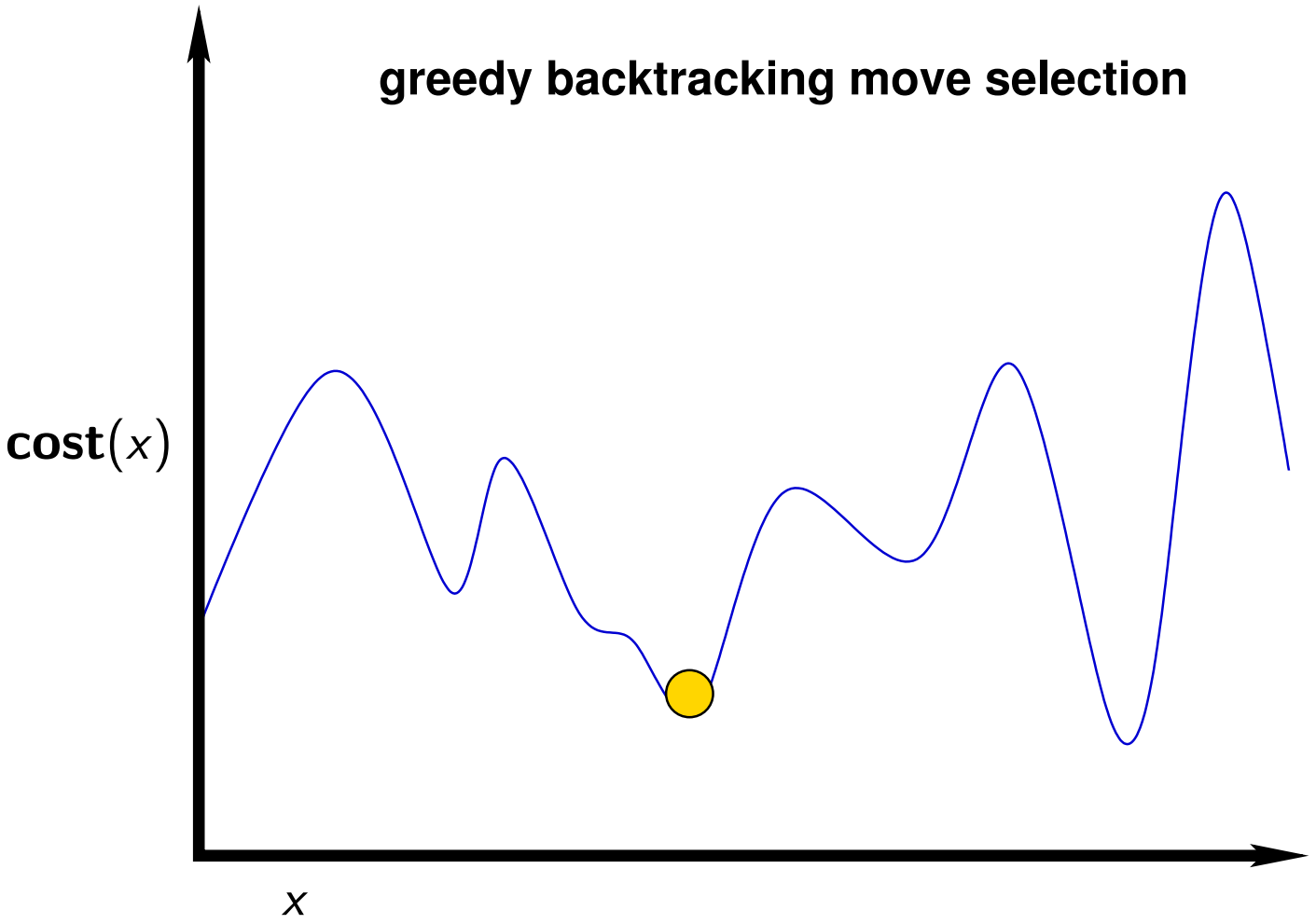
Backtracking



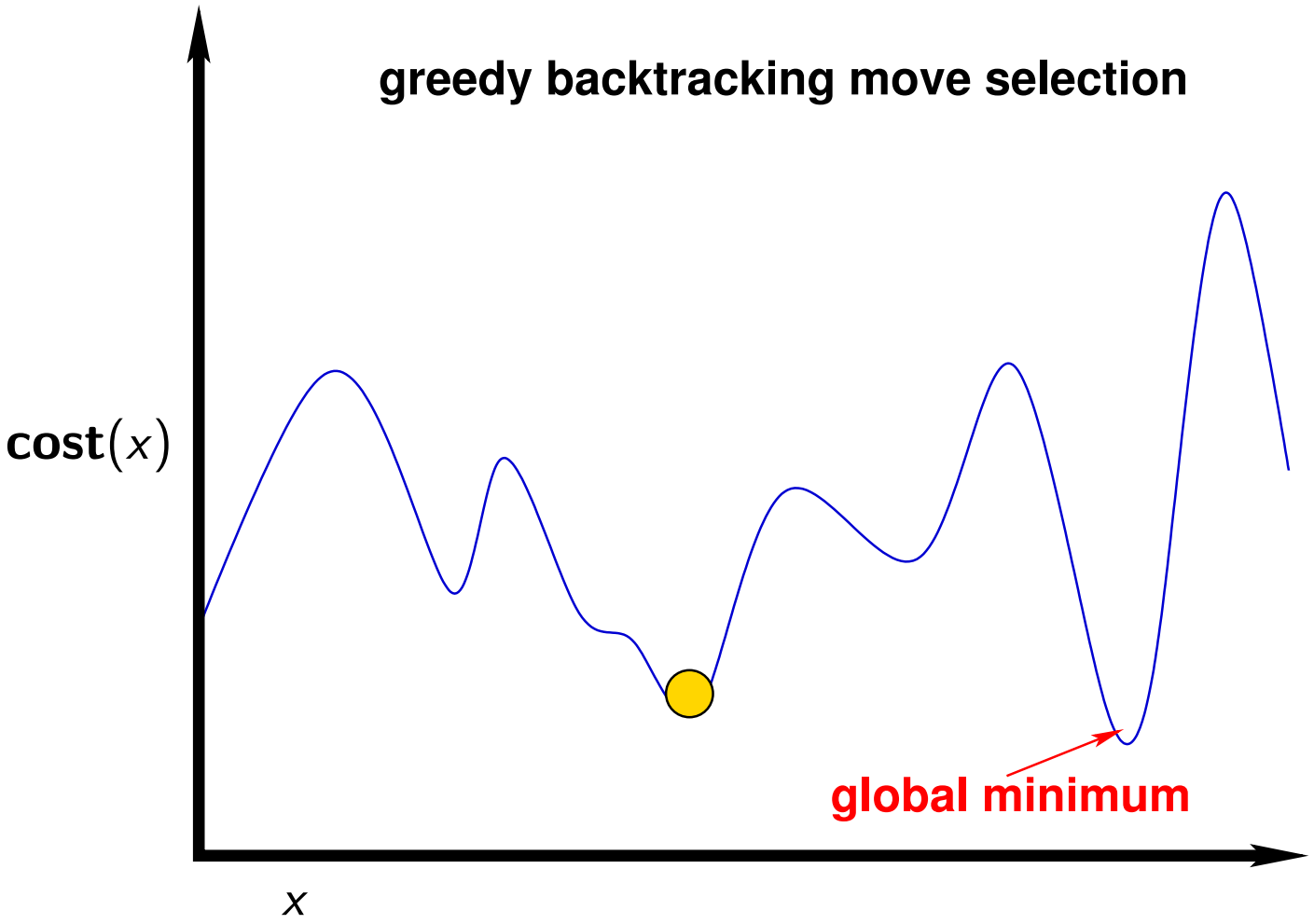
Backtracking



Backtracking



Backtracking



Tabu search

Similar to iterative improvement.

Iterative improvement can cycle.

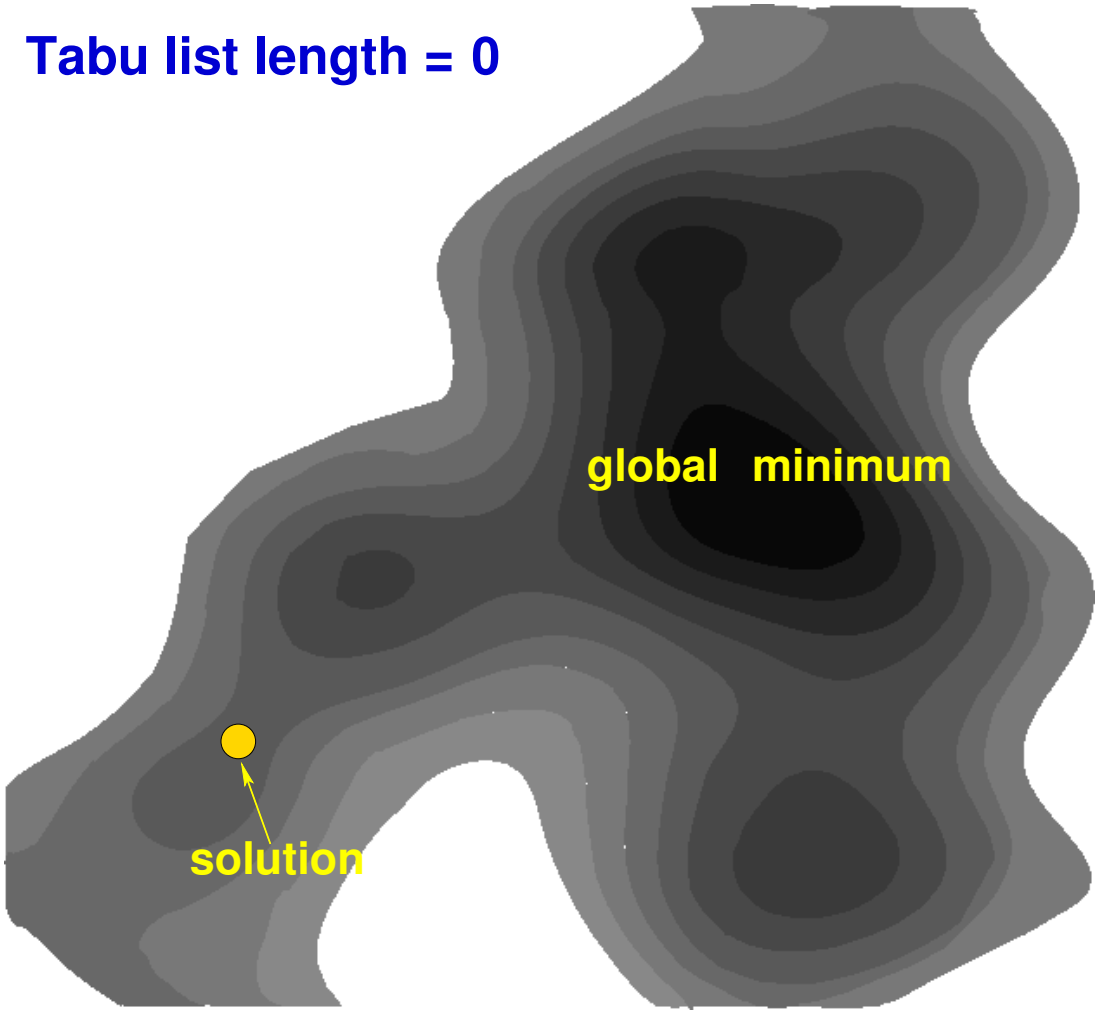
- Chooses largest cost decrease move...
- ...then chooses smallest cost increase move.

Tabu search has a *tabu list*.

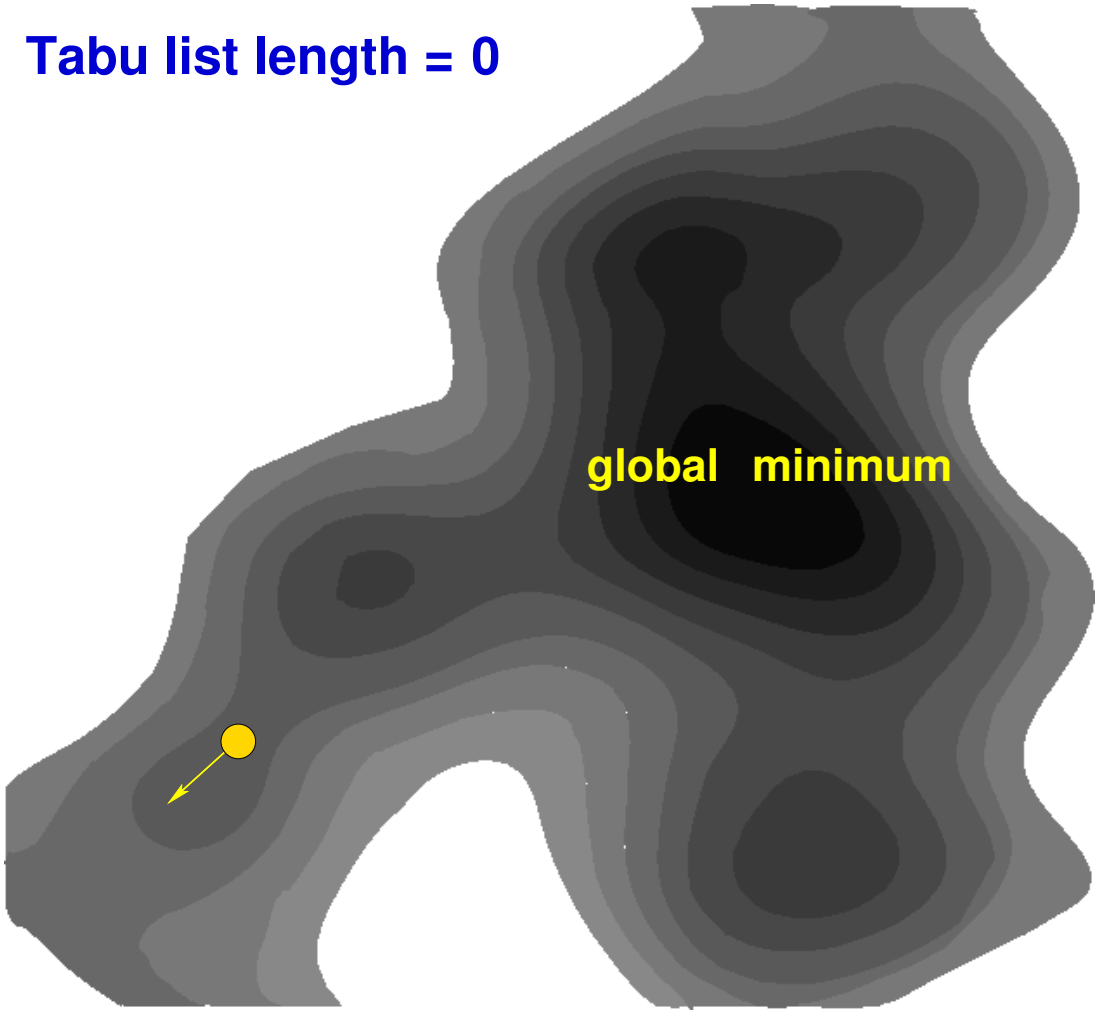
- Solutions to avoid.
- Solution characteristics to avoid.

Prevents iterative cycles

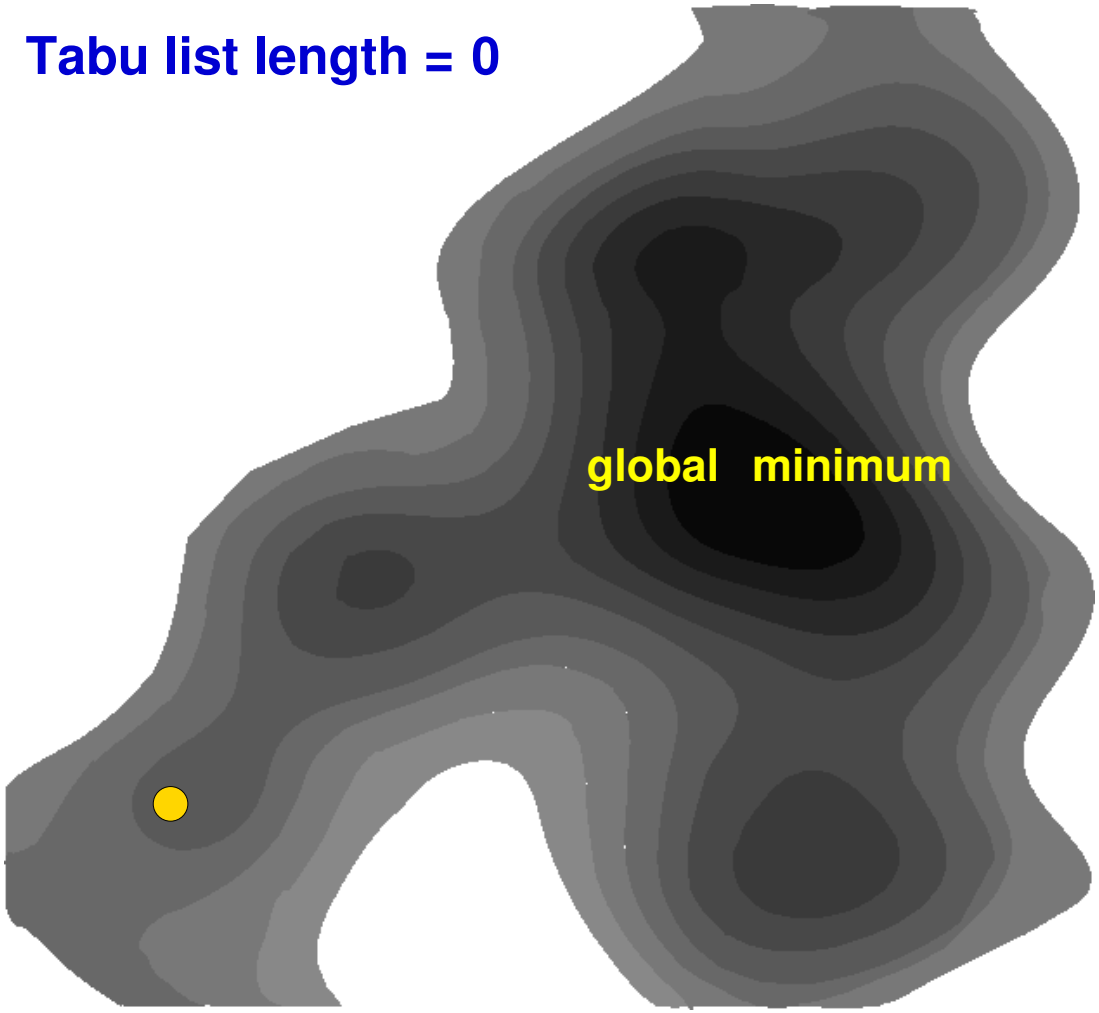
Tabu search example



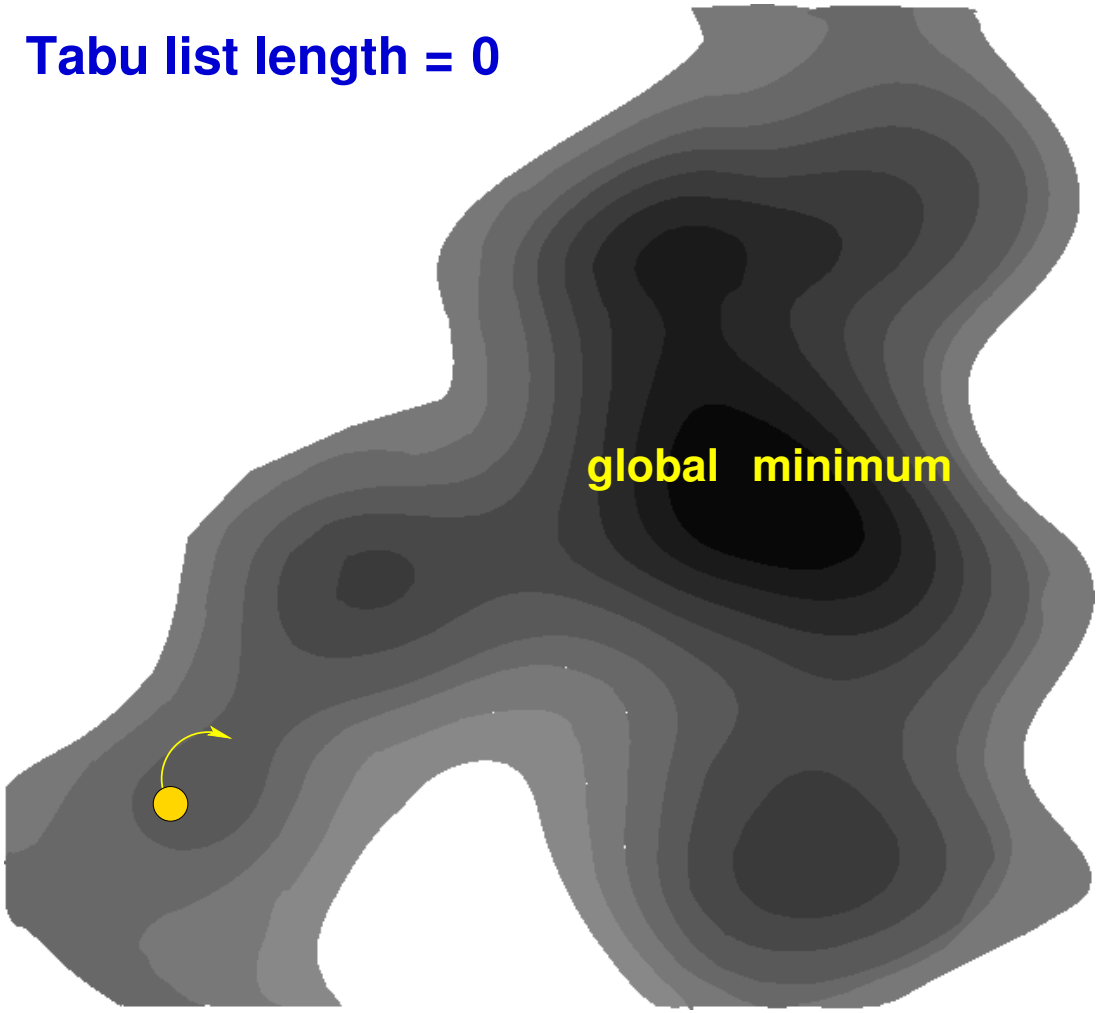
Tabu search example



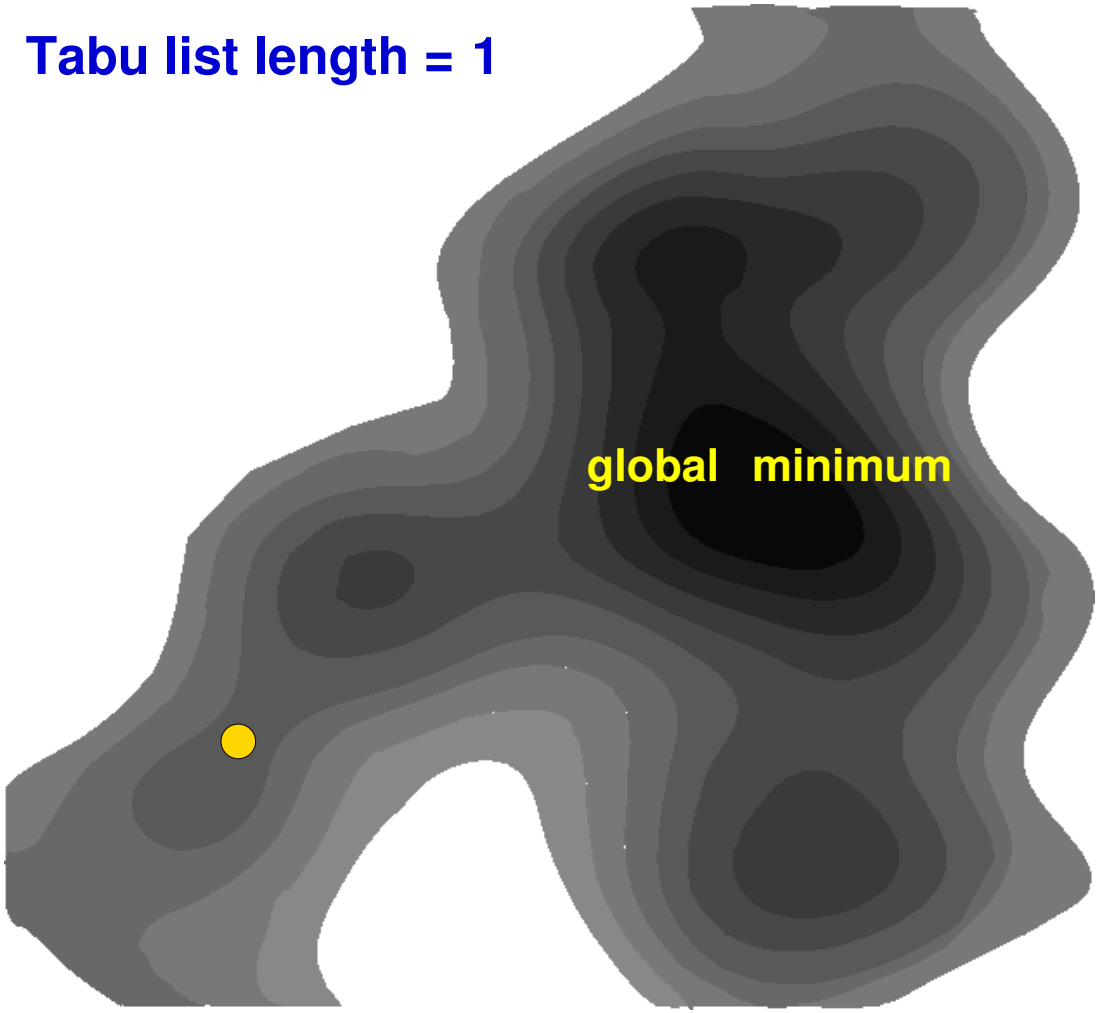
Tabu search example



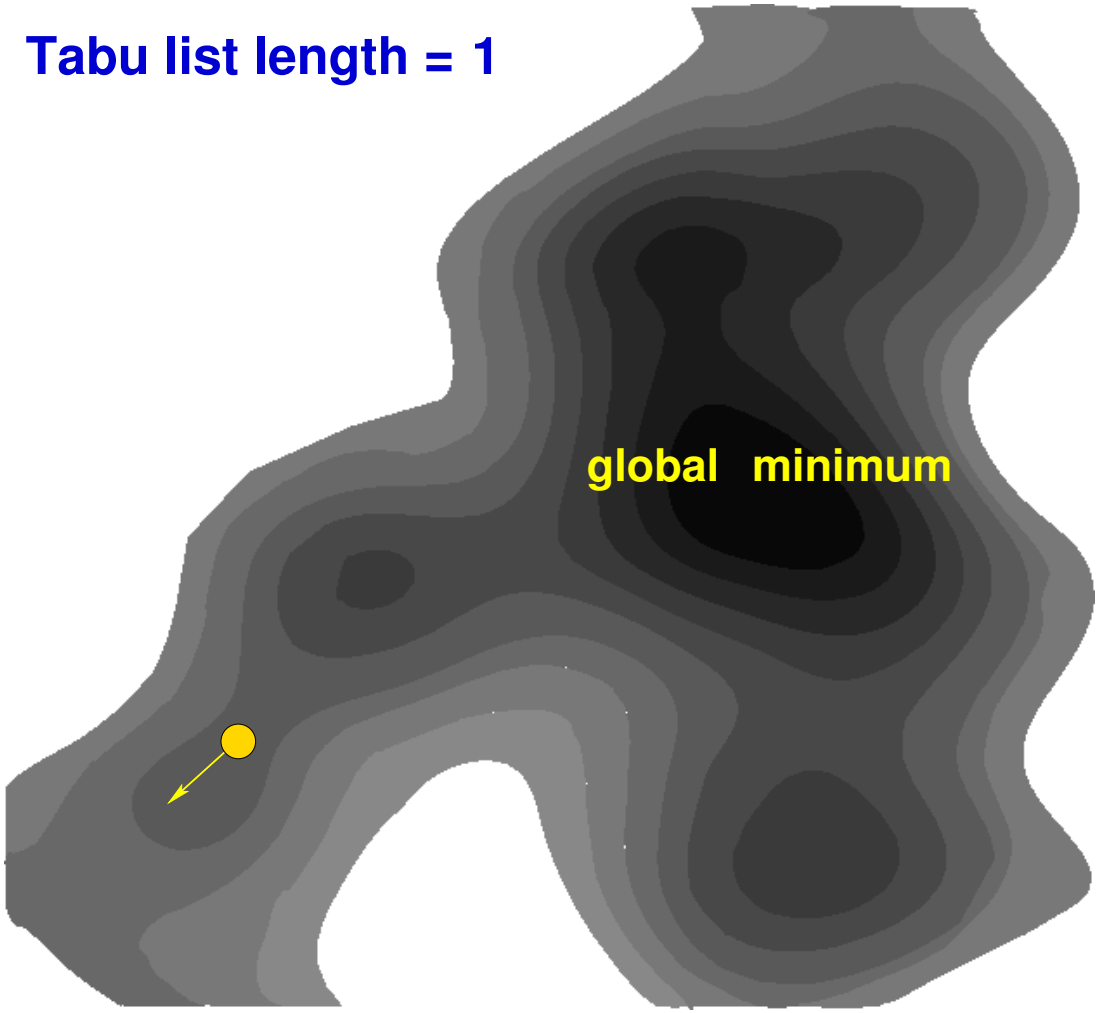
Tabu search example



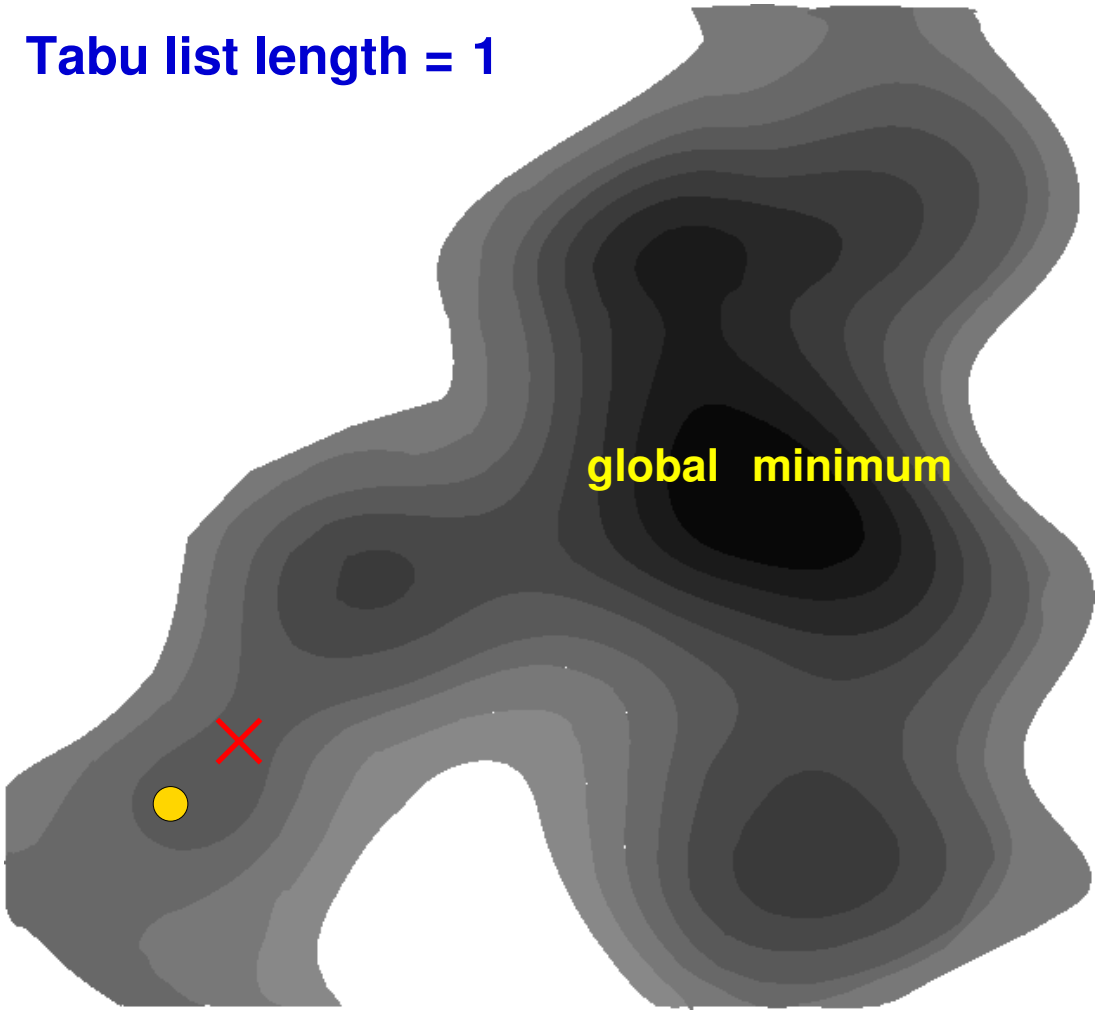
Tabu search example



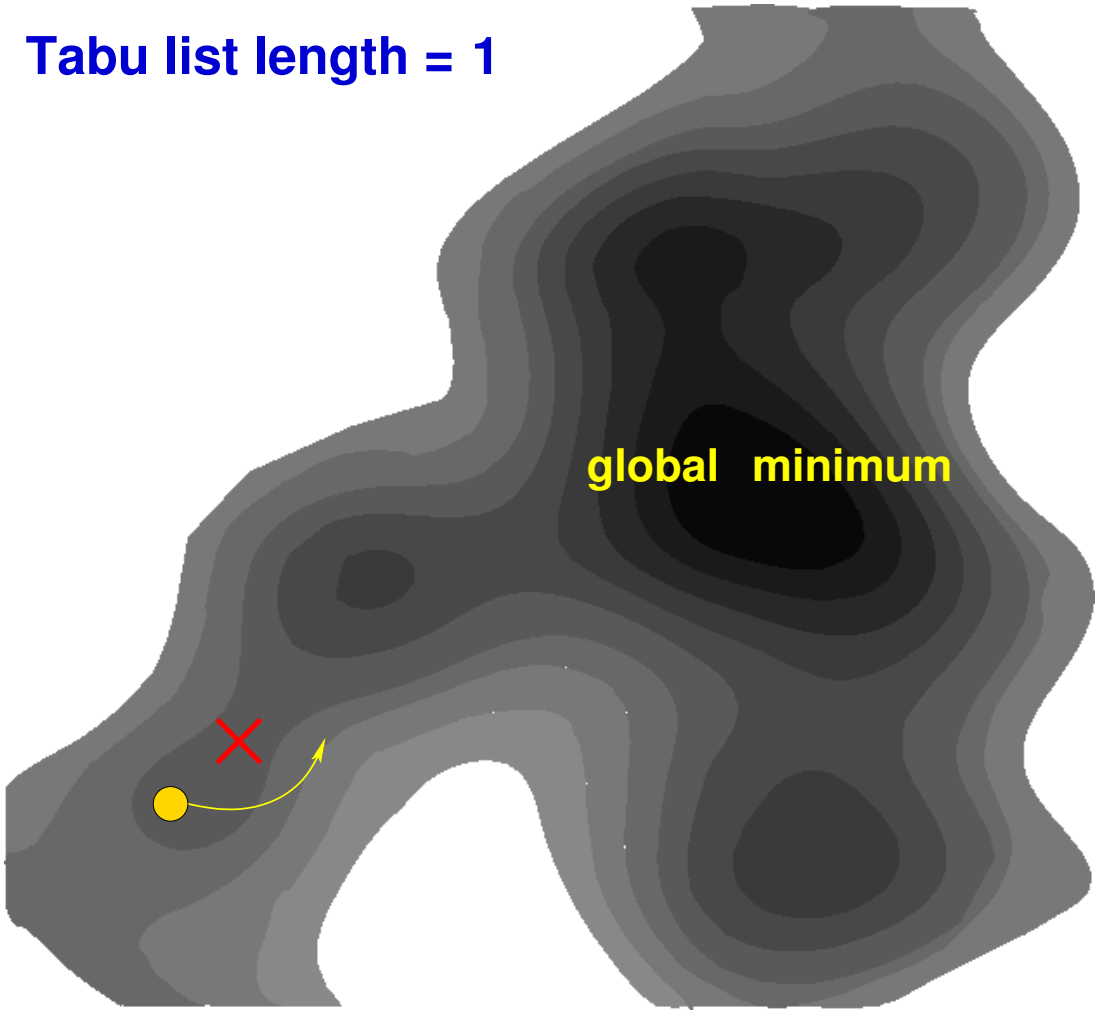
Tabu search example



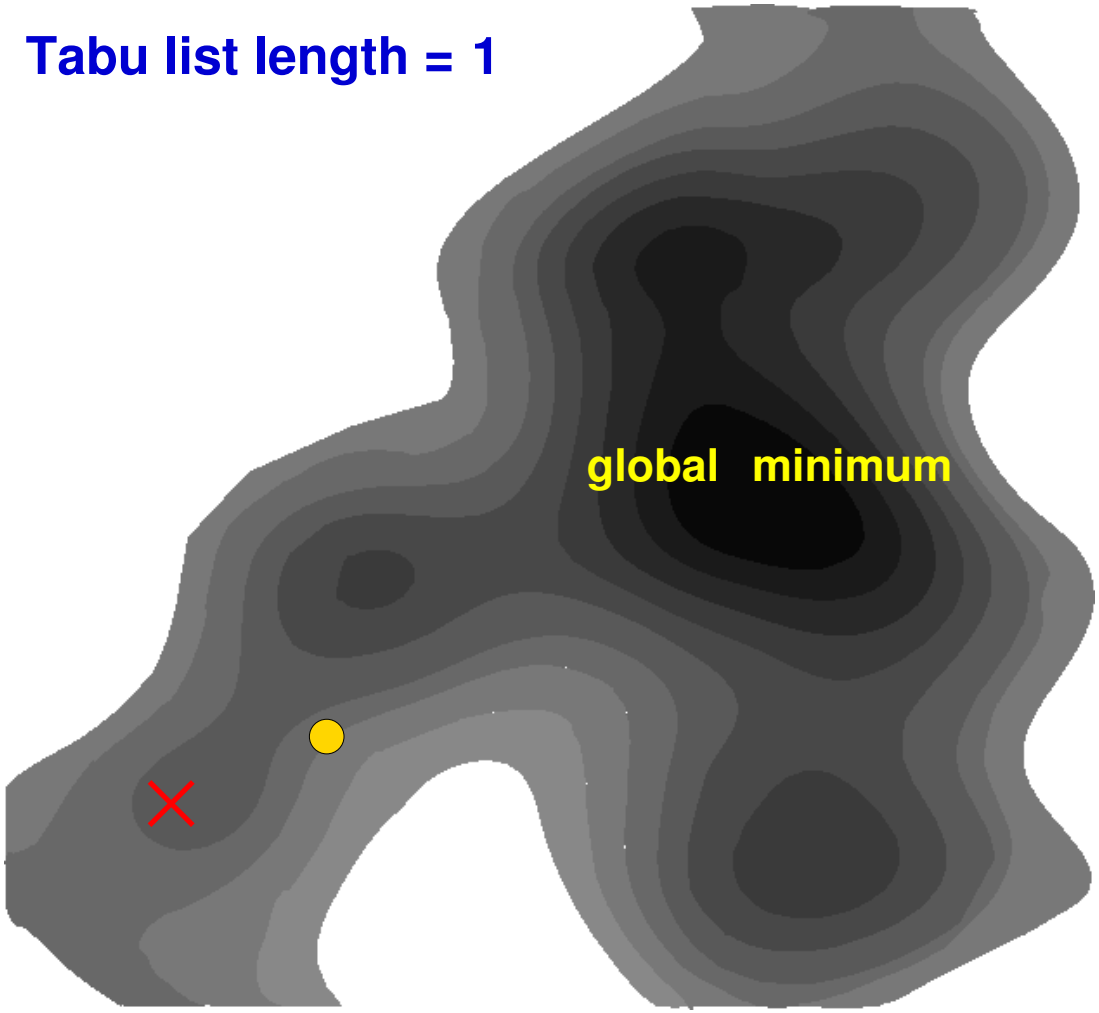
Tabu search example



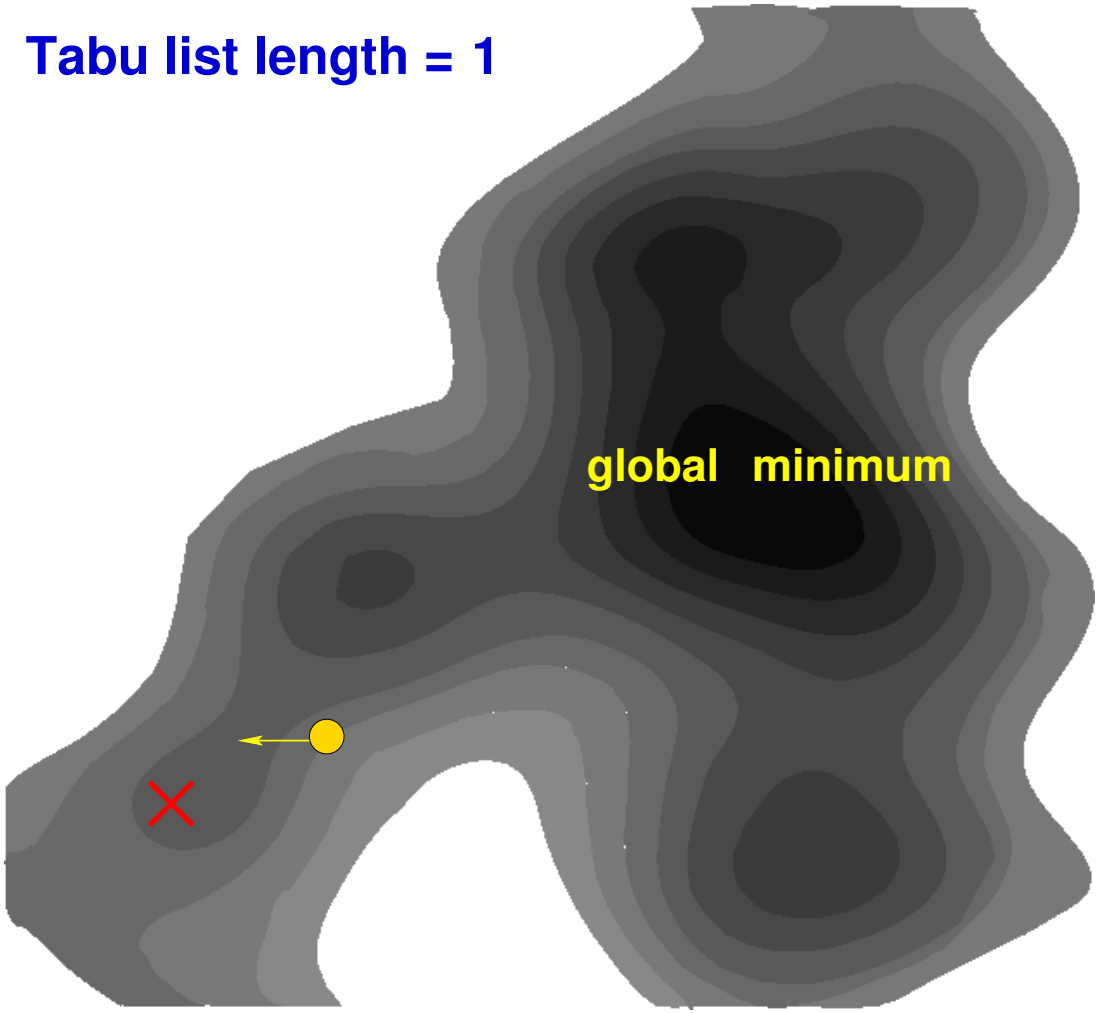
Tabu search example



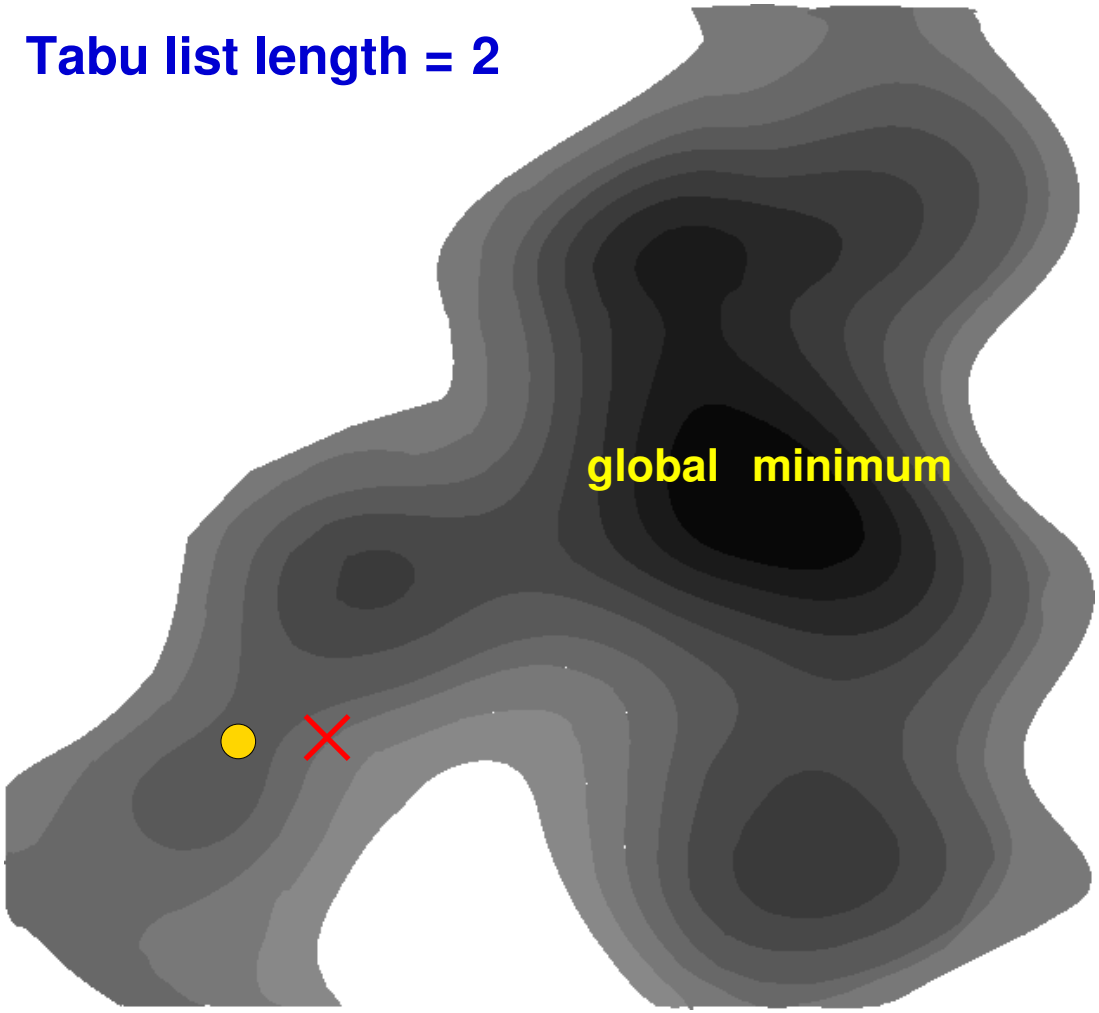
Tabu search example



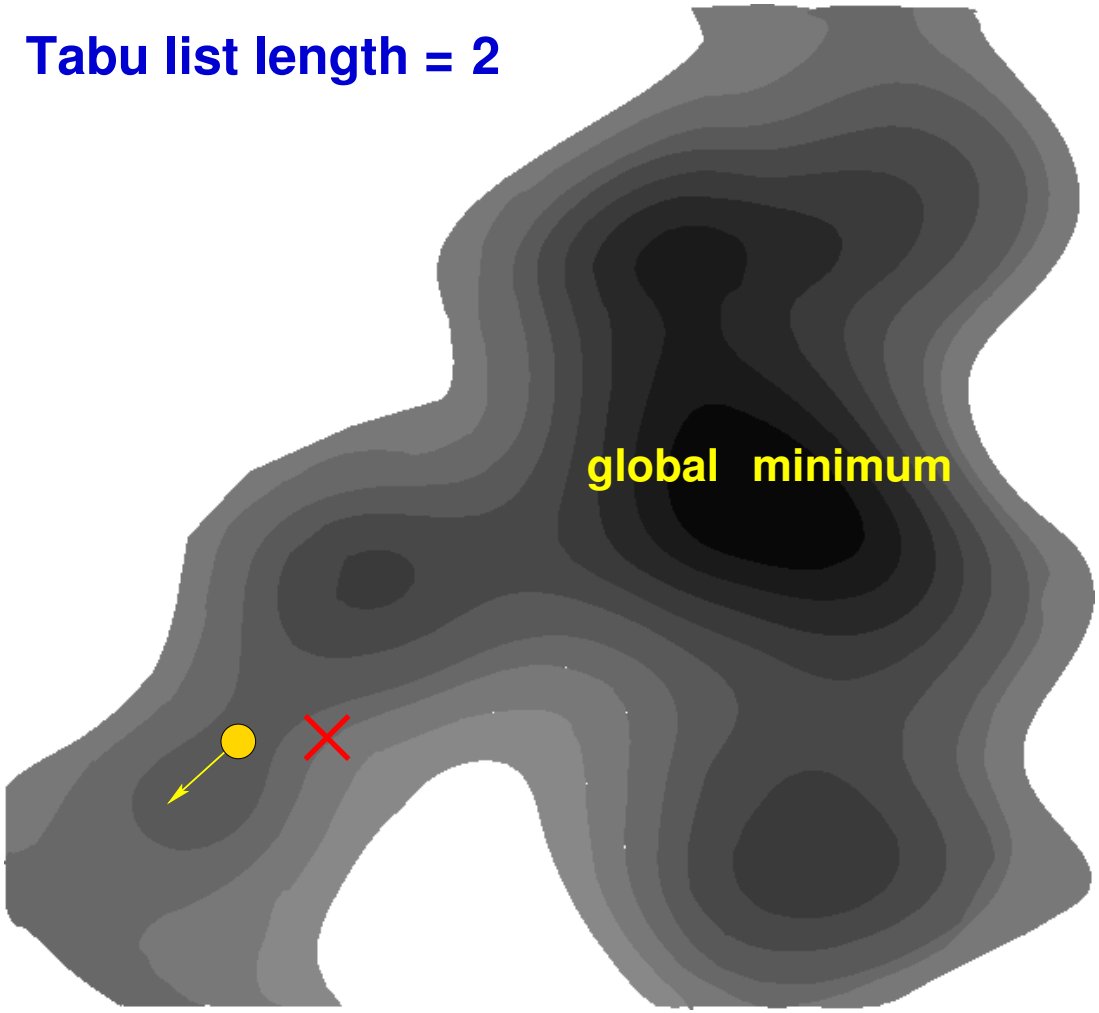
Tabu search example



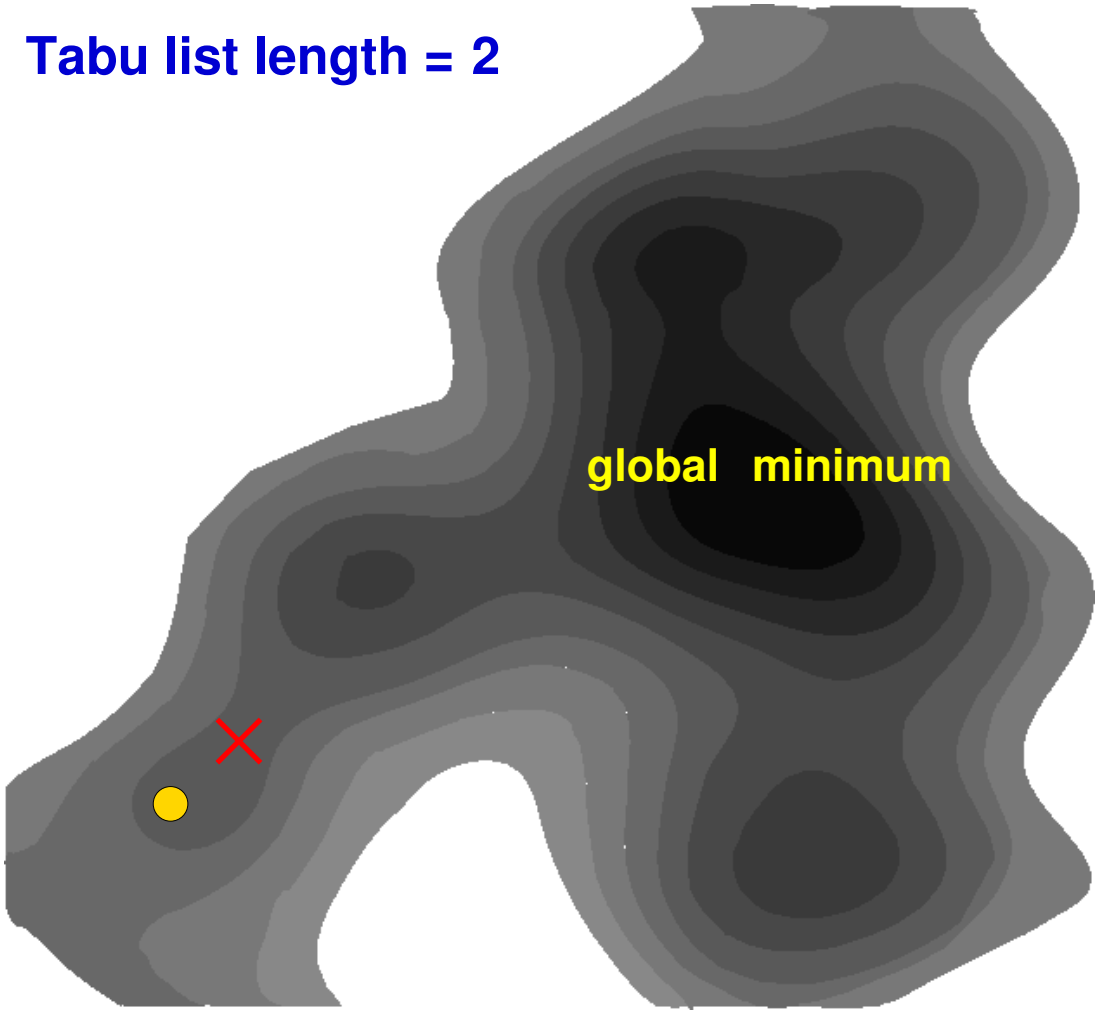
Tabu search example



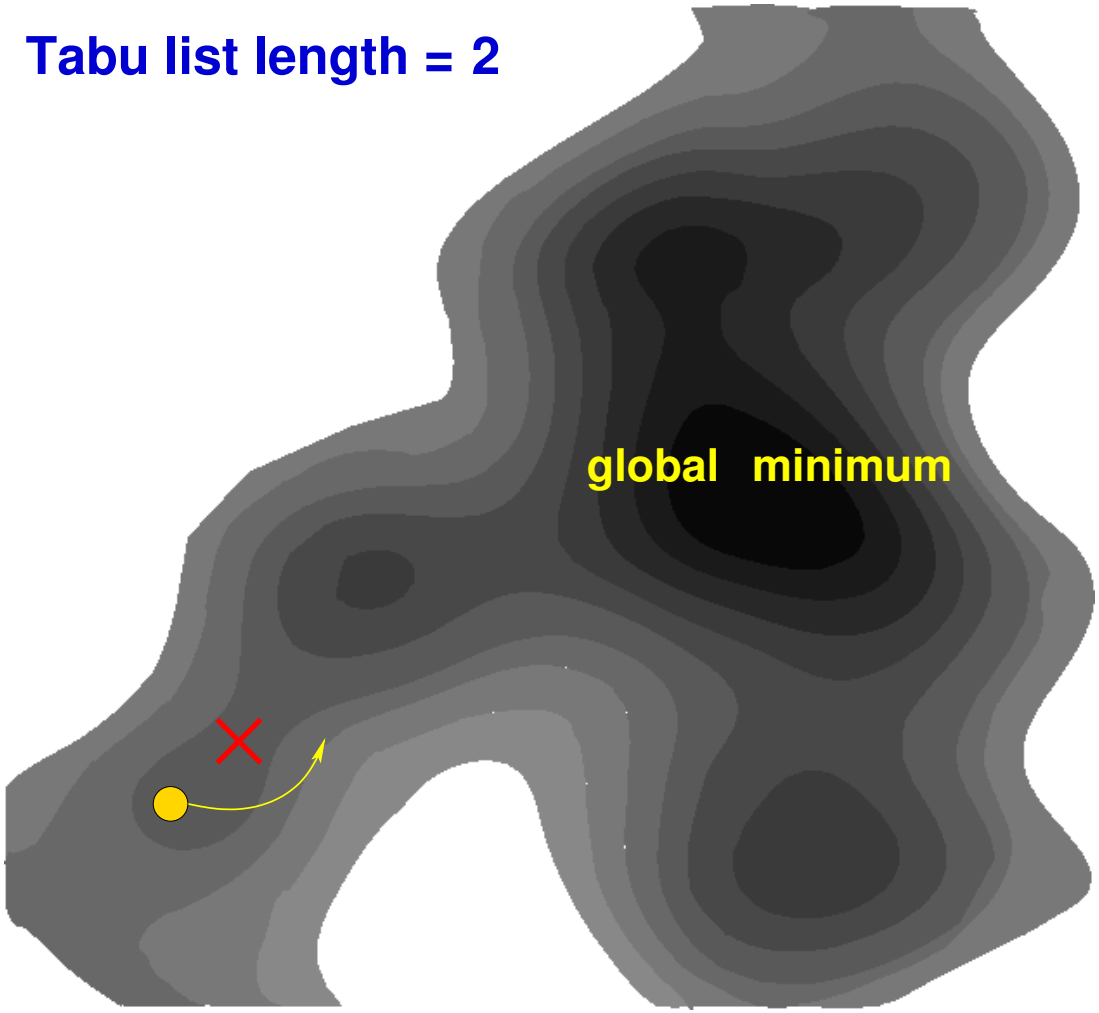
Tabu search example



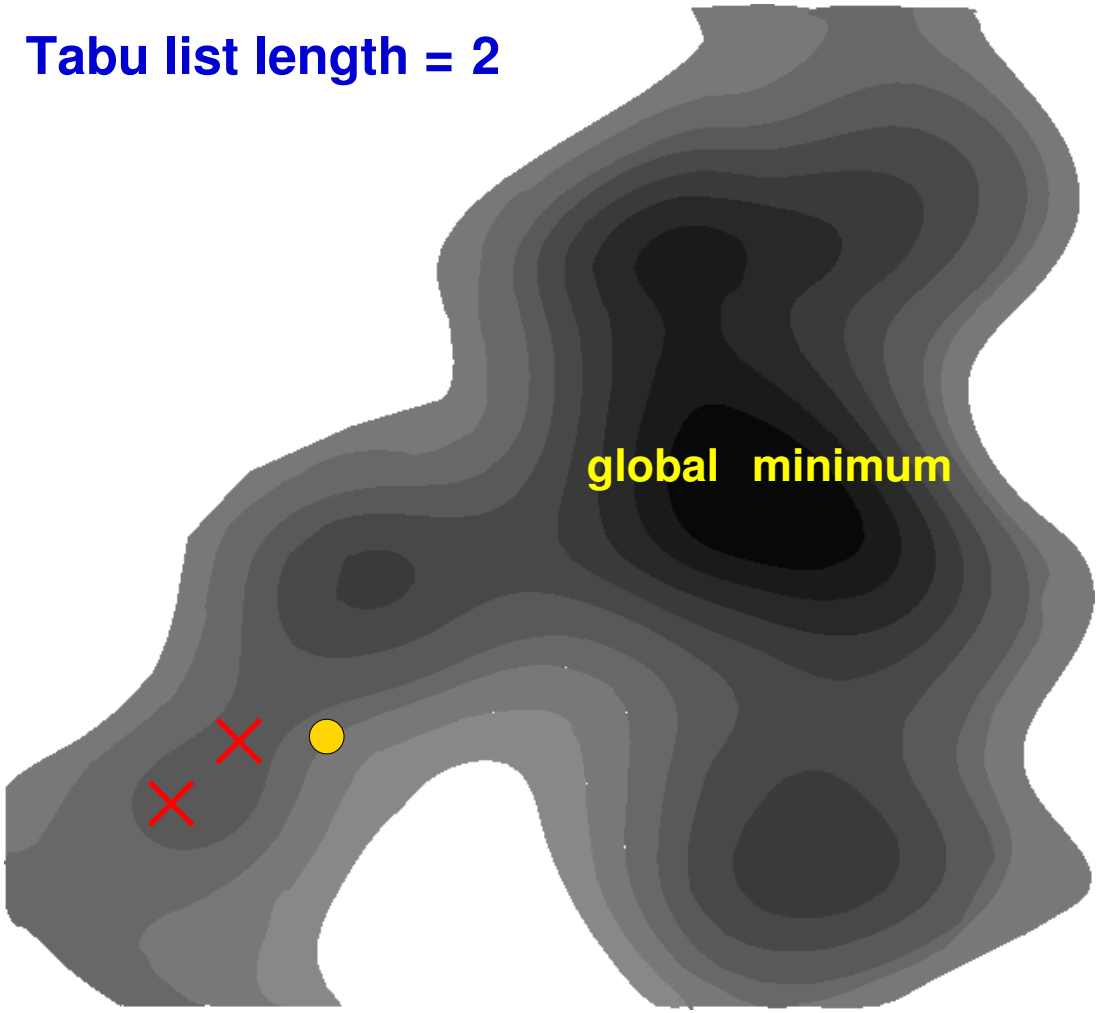
Tabu search example



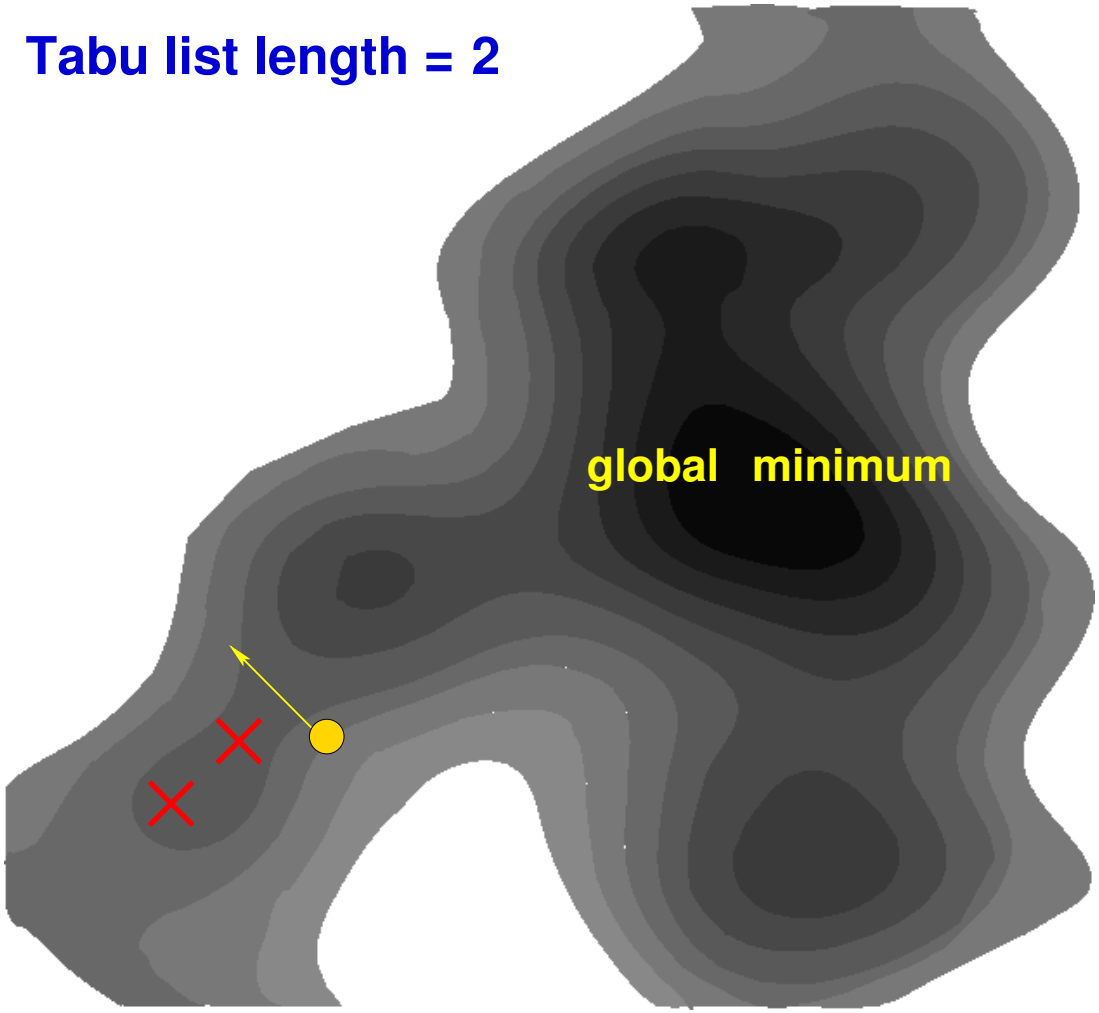
Tabu search example



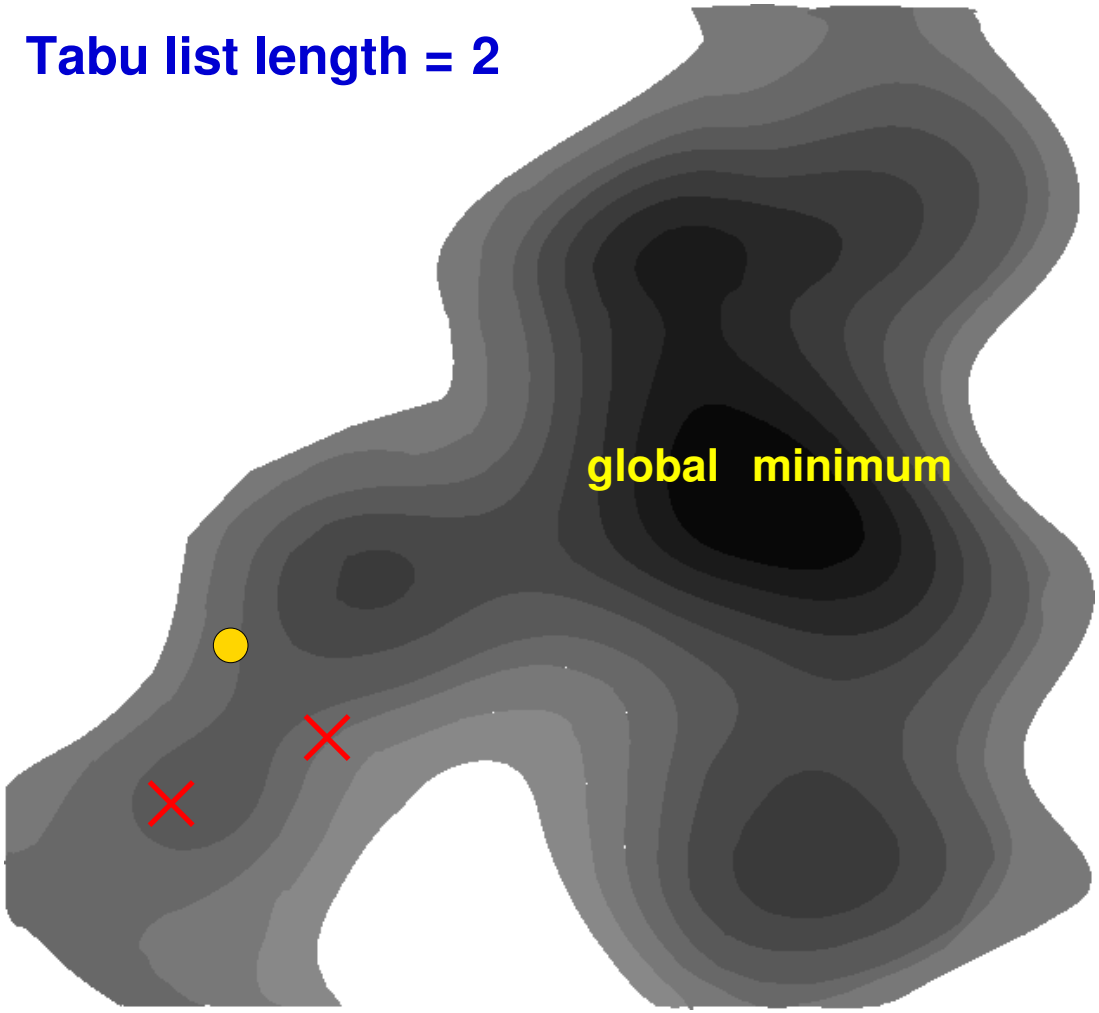
Tabu search example



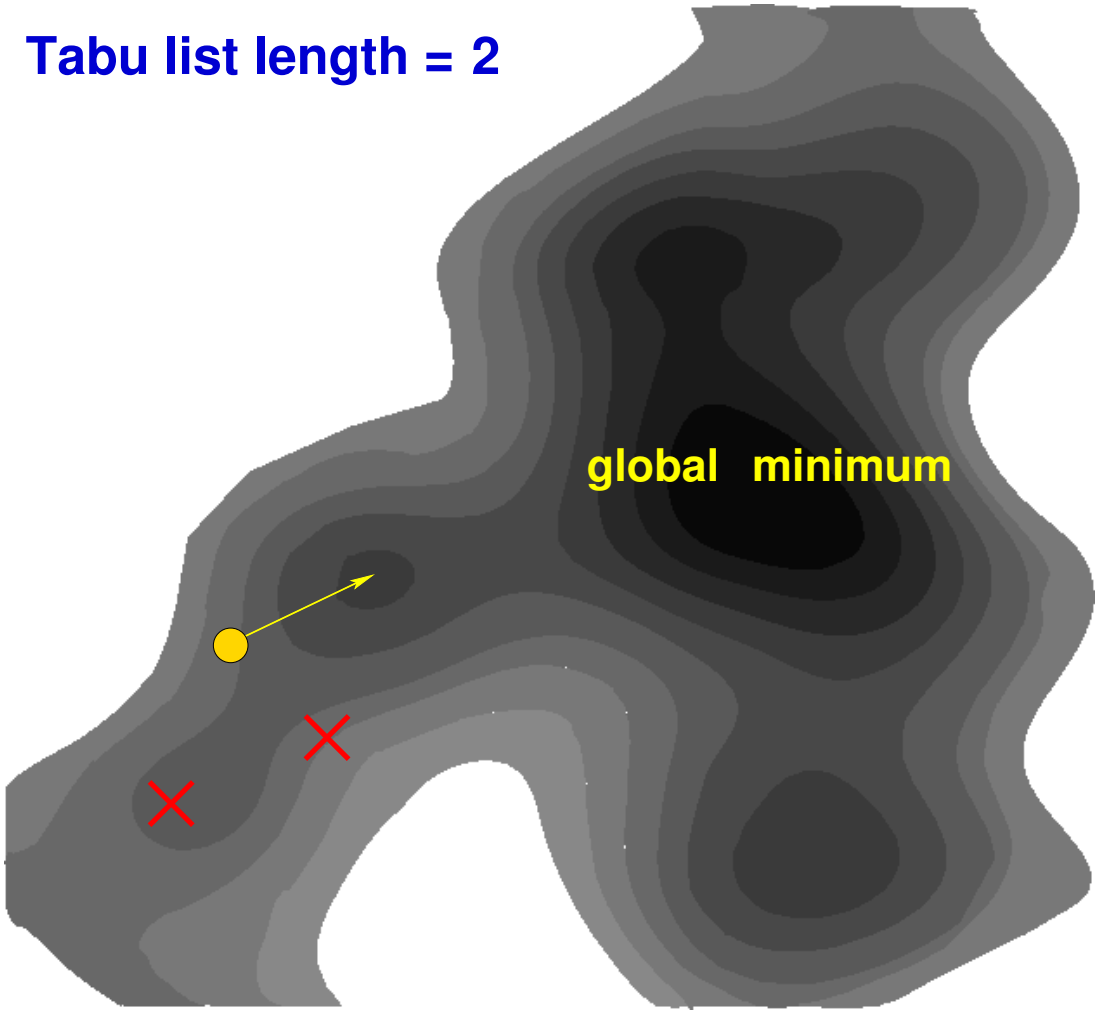
Tabu search example



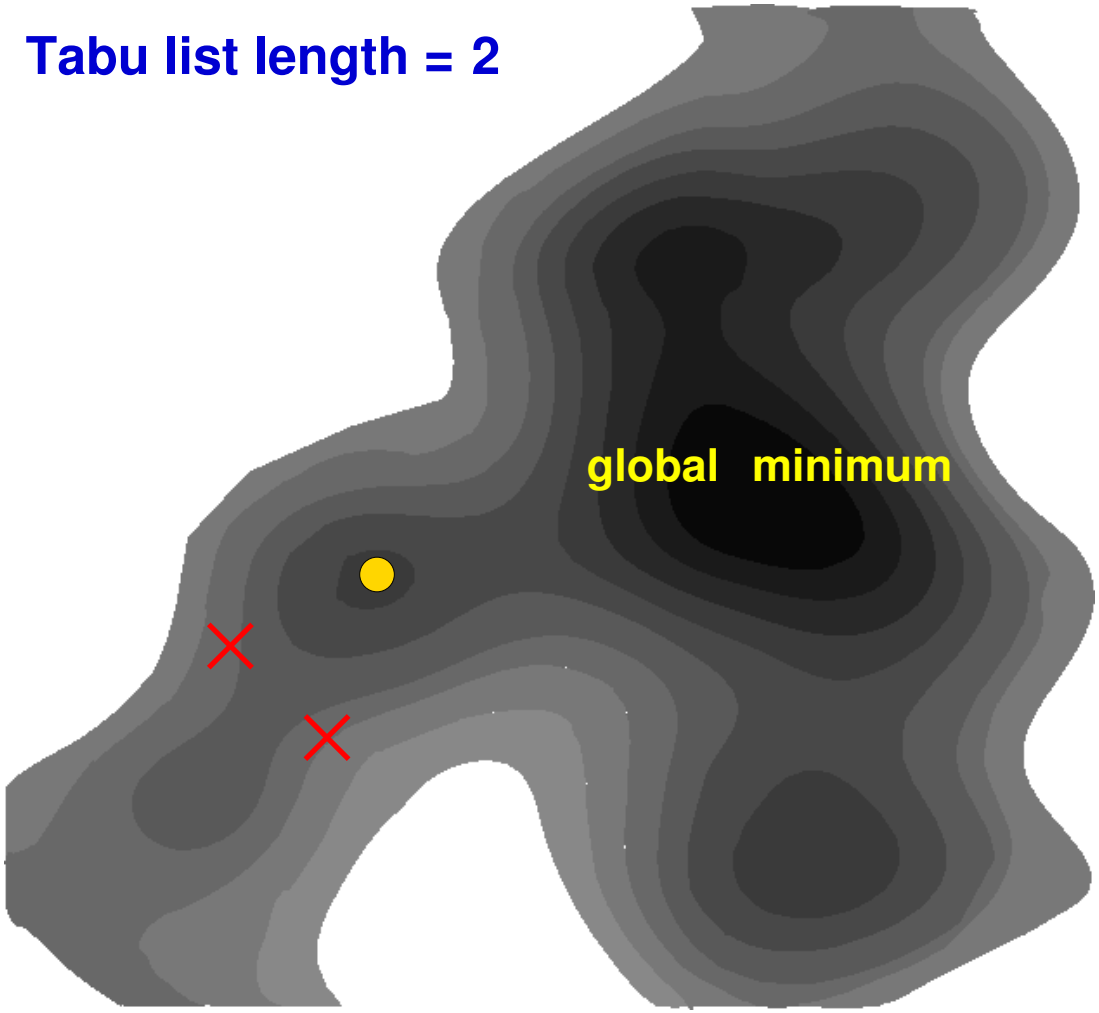
Tabu search example



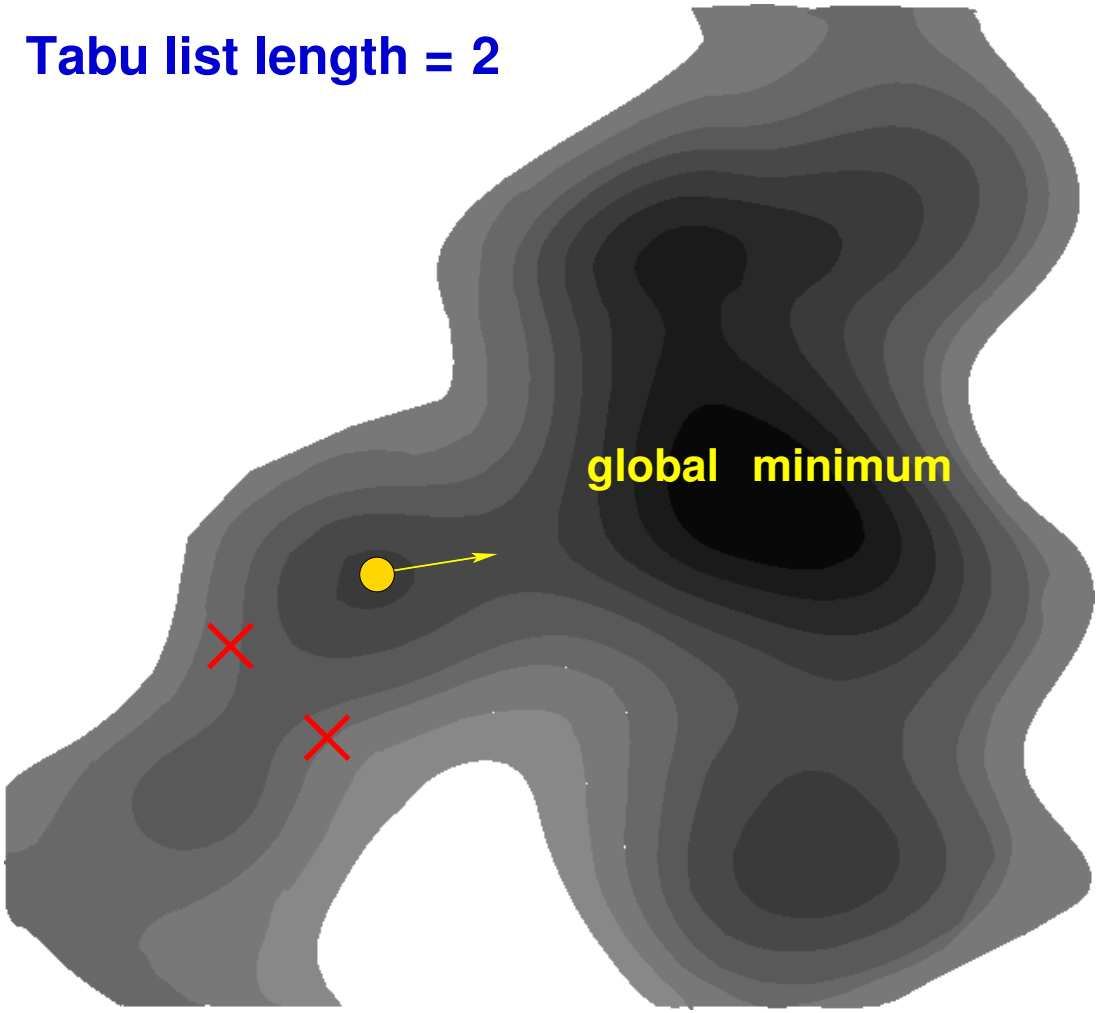
Tabu search example



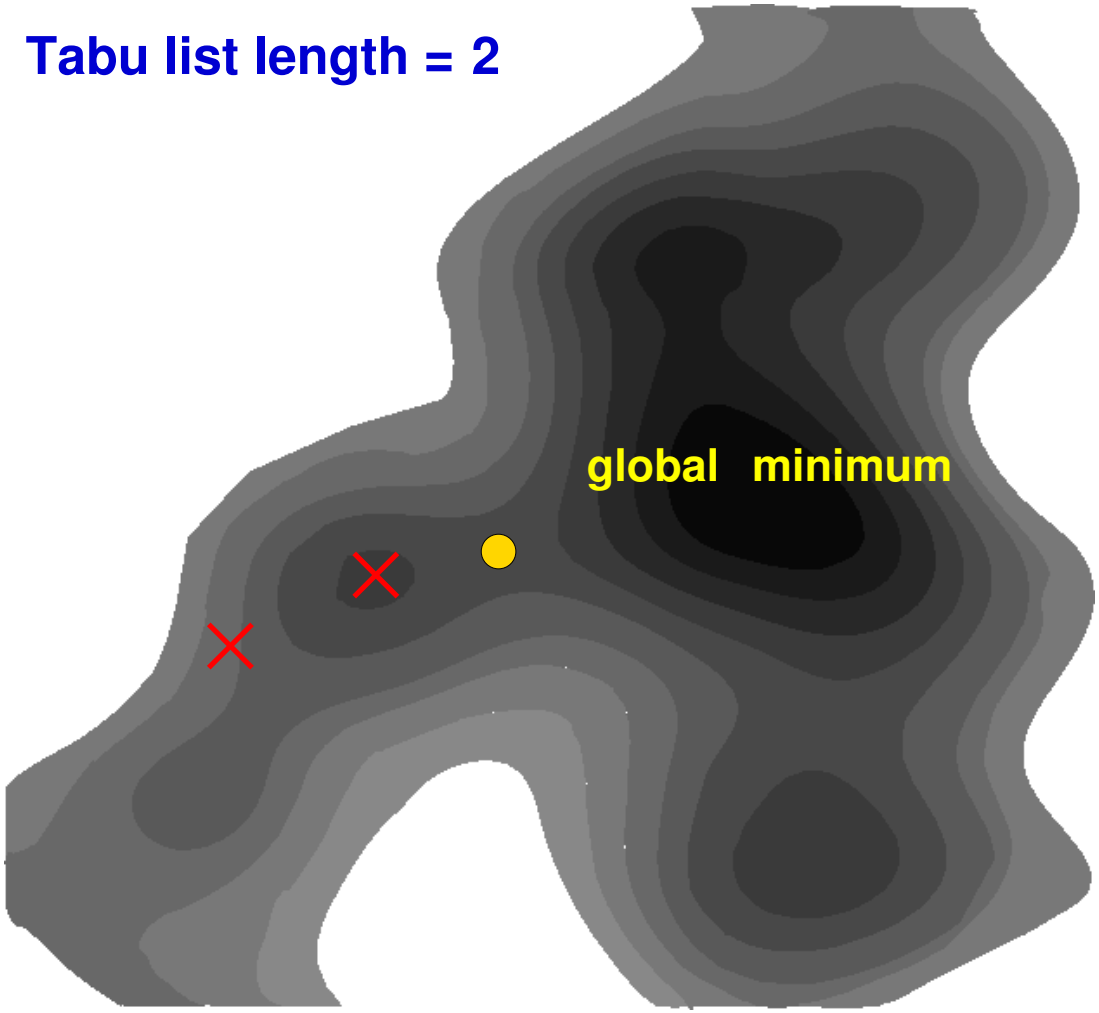
Tabu search example



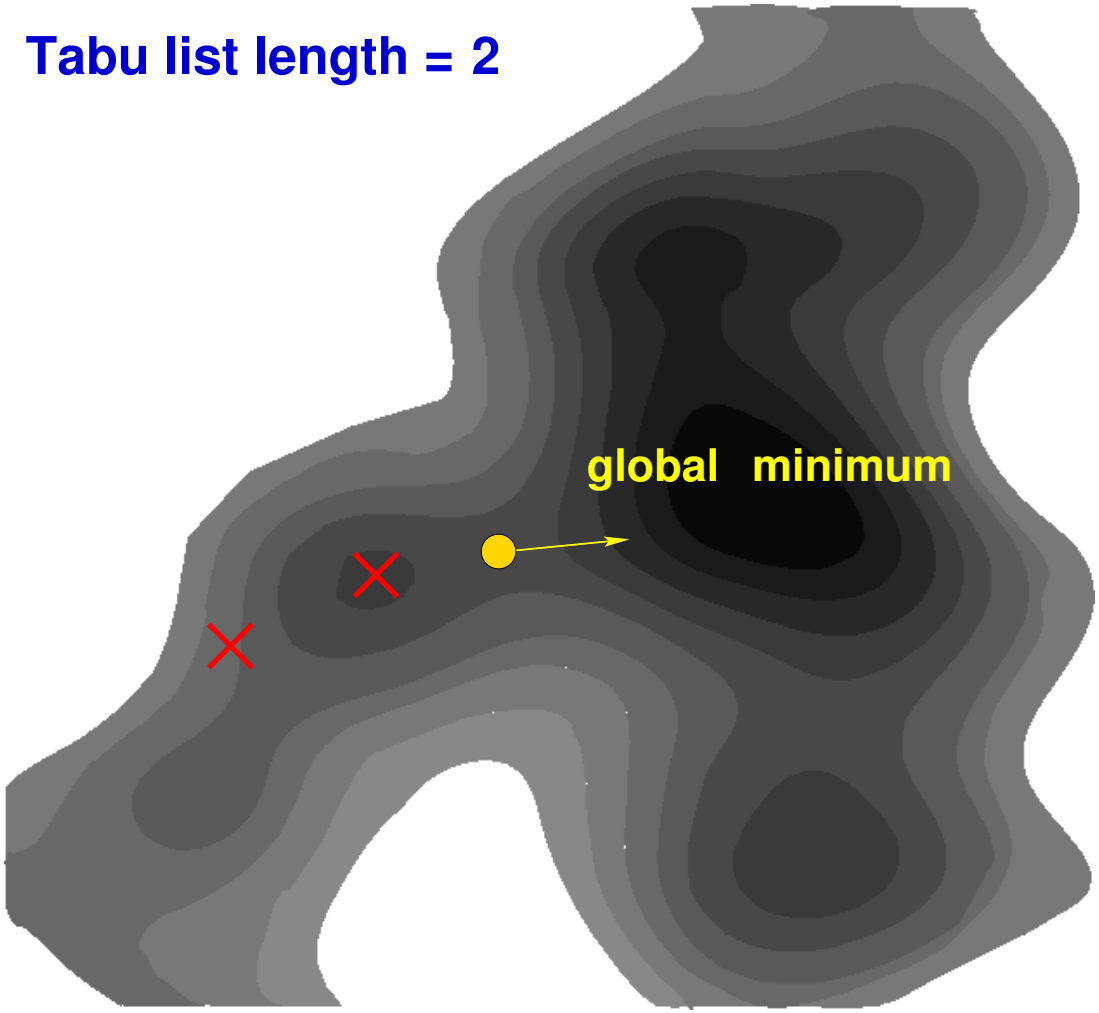
Tabu search example



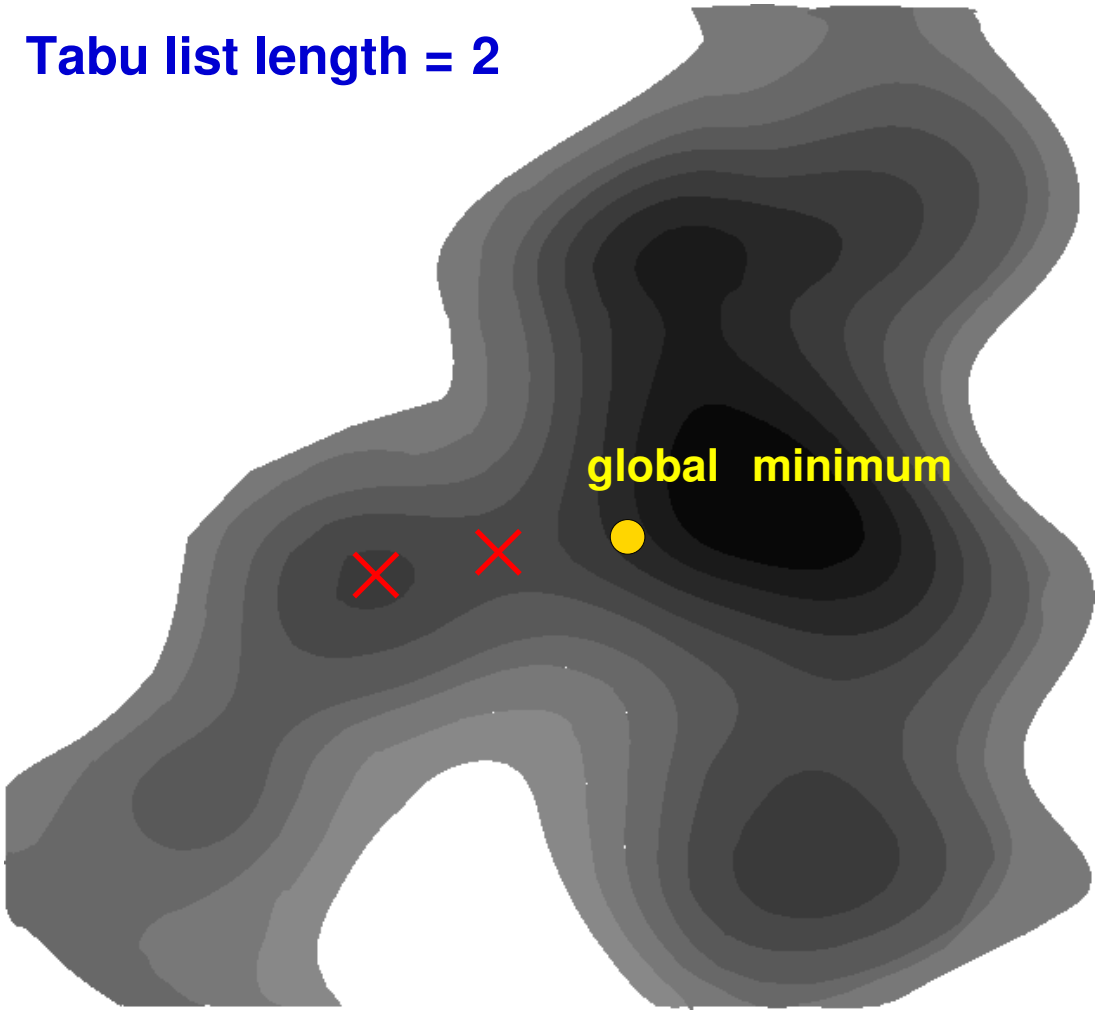
Tabu search example



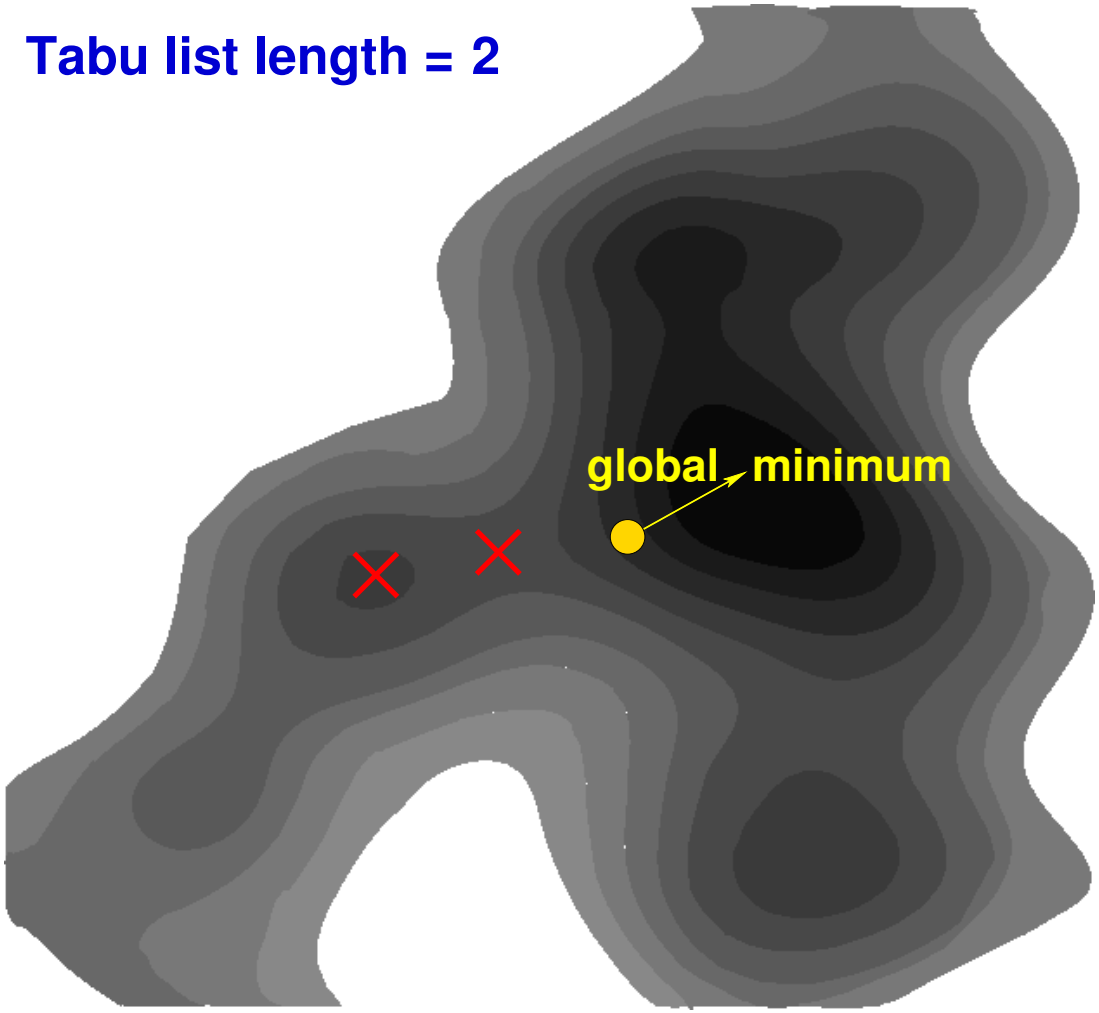
Tabu search example



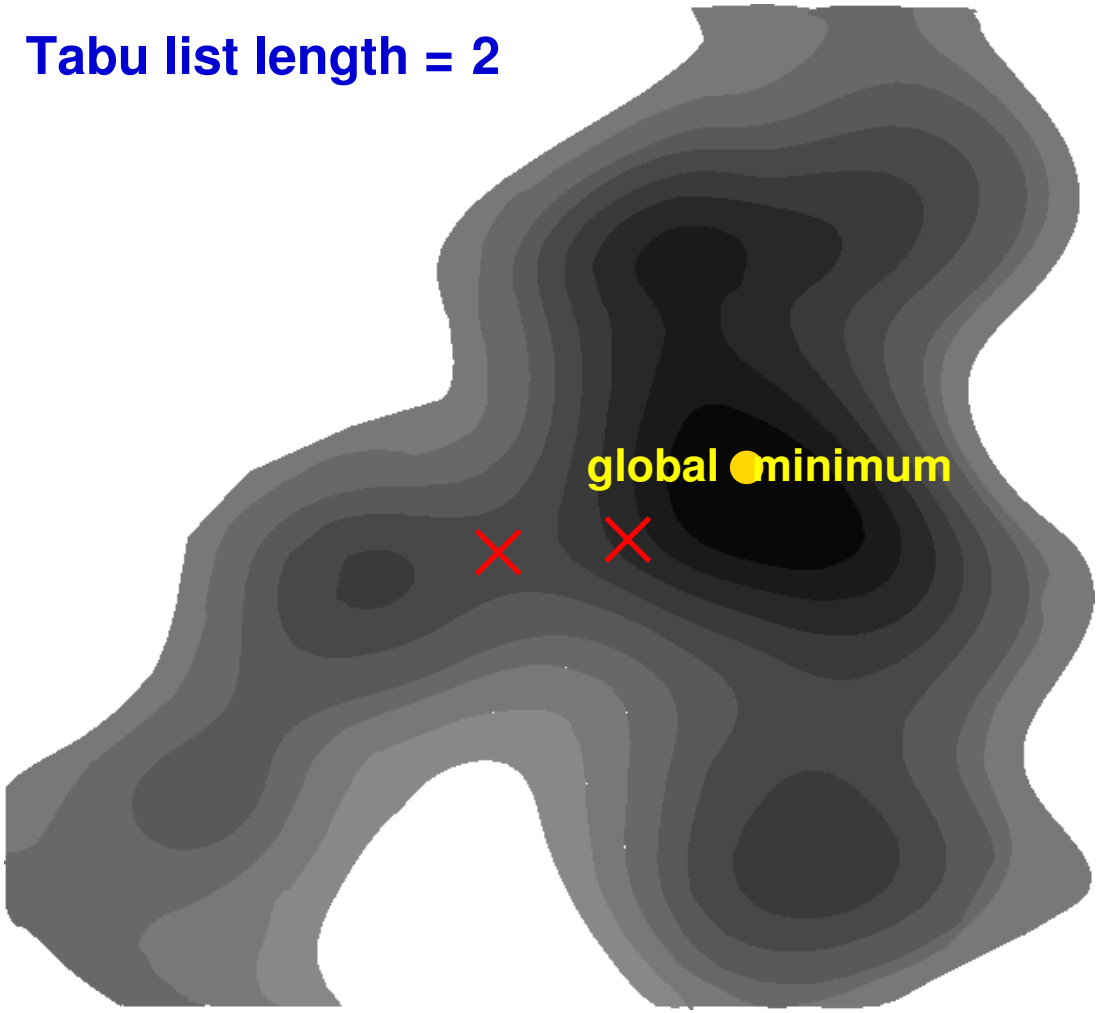
Tabu search example



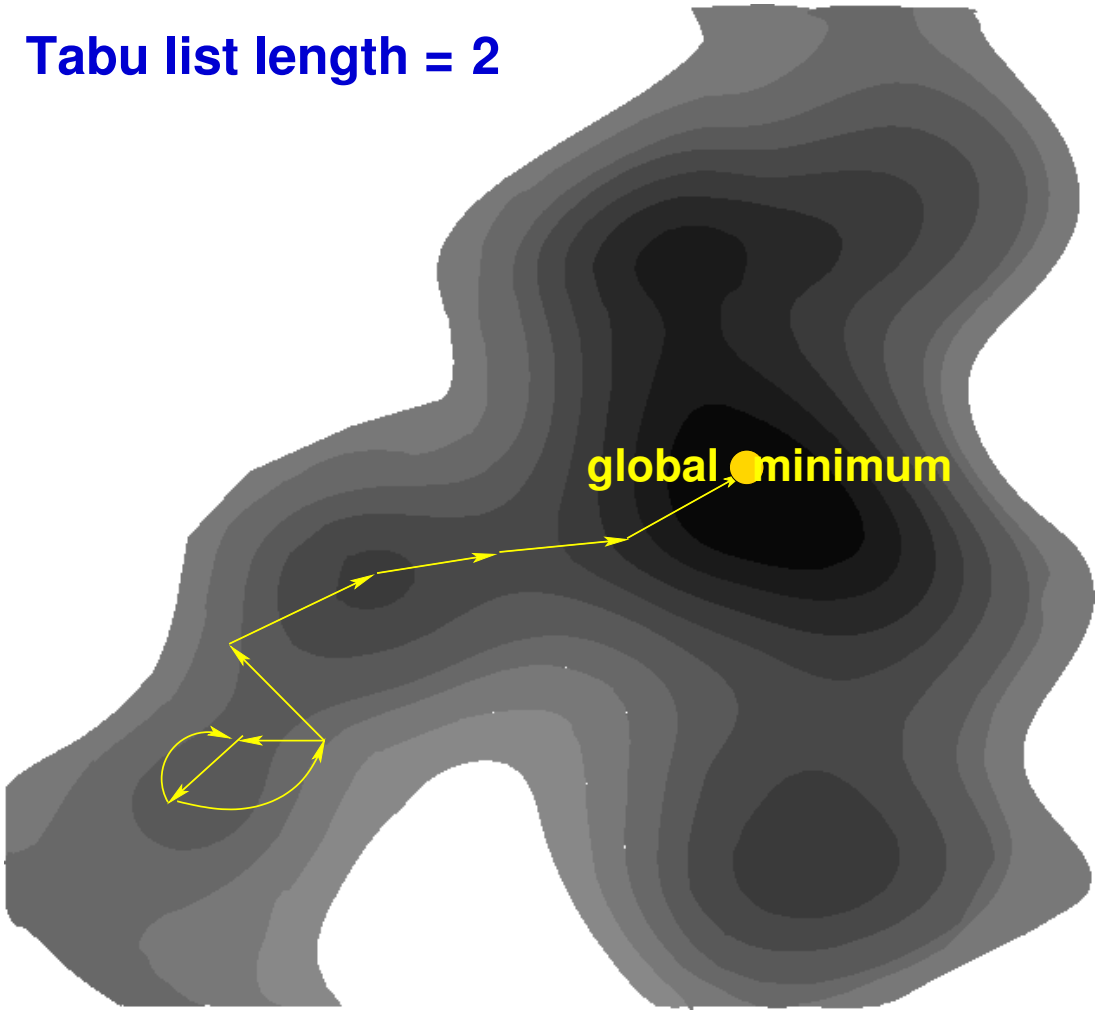
Tabu search example



Tabu search example



Tabu search example



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

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

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

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

Boltzmann trials

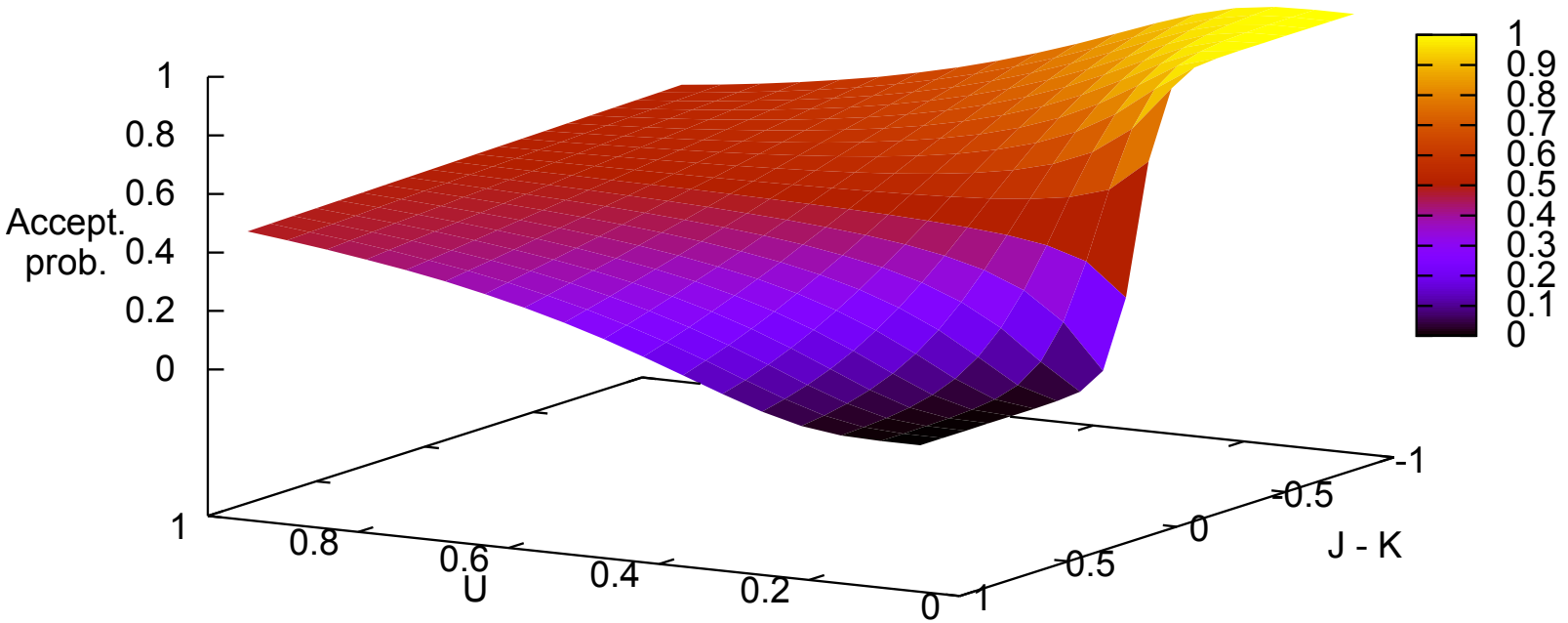
Introduce convenience variable U.

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

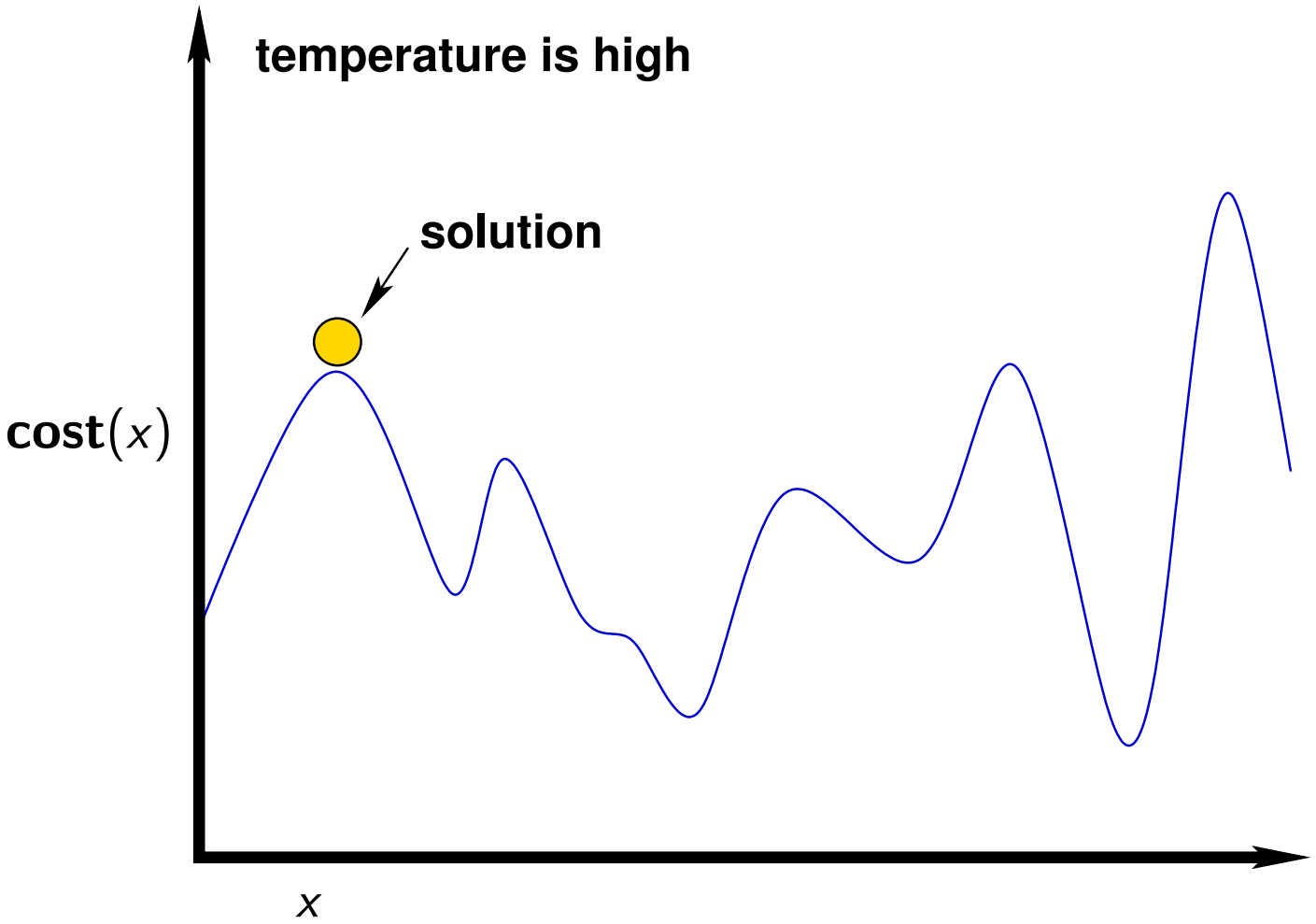
$$U(0) = 0$$

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

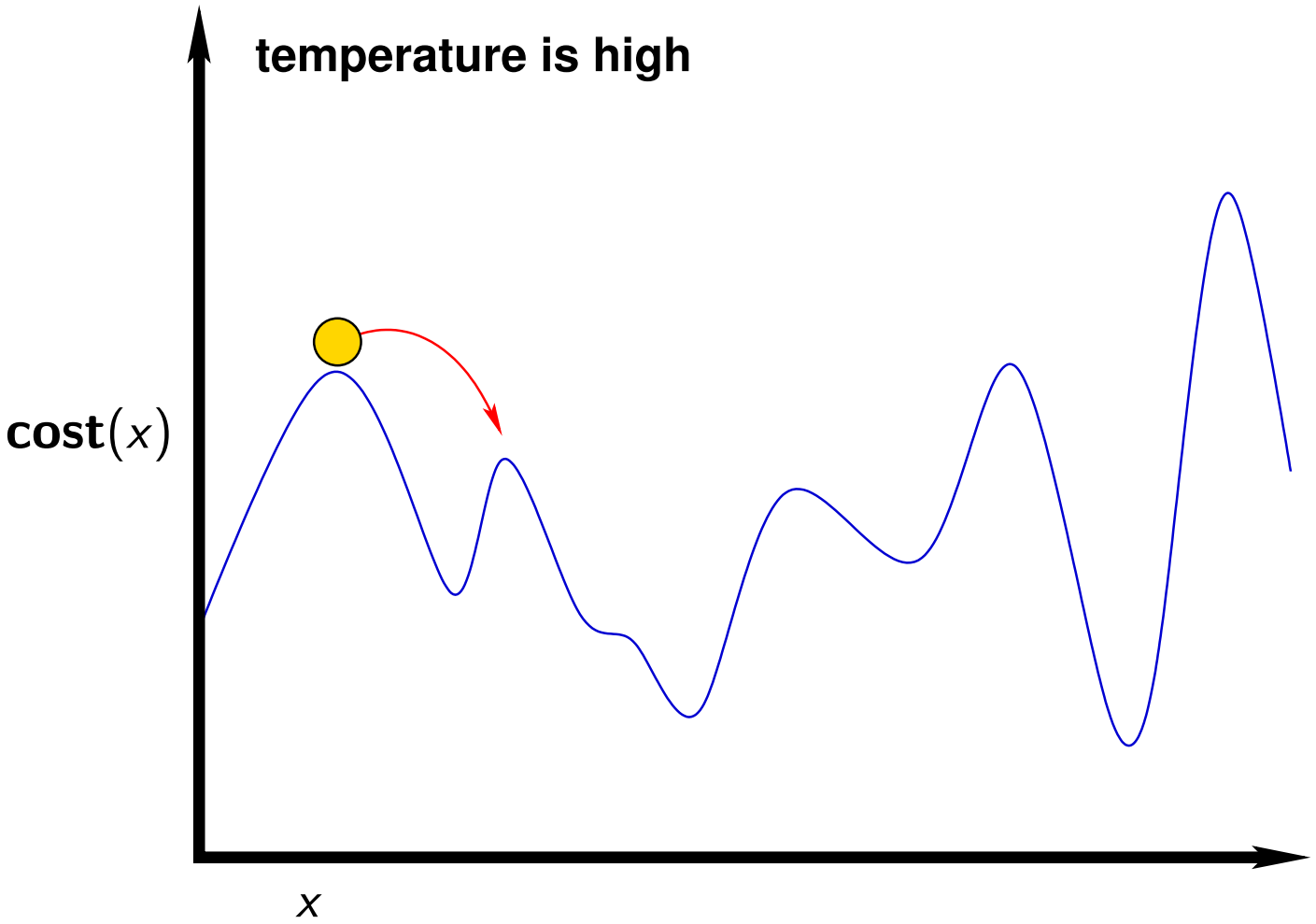
Boltzmann trials



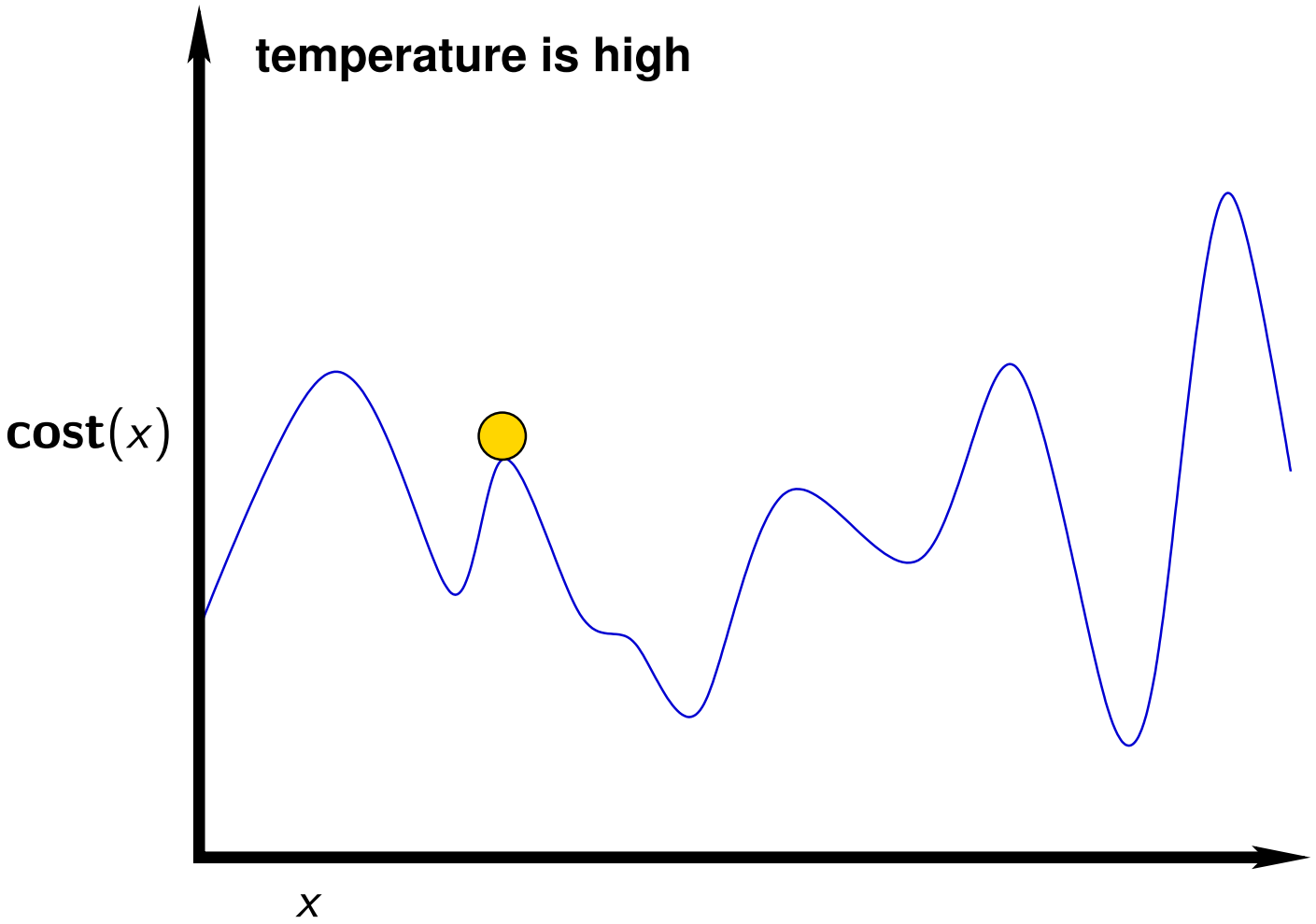
Simulated annealing example



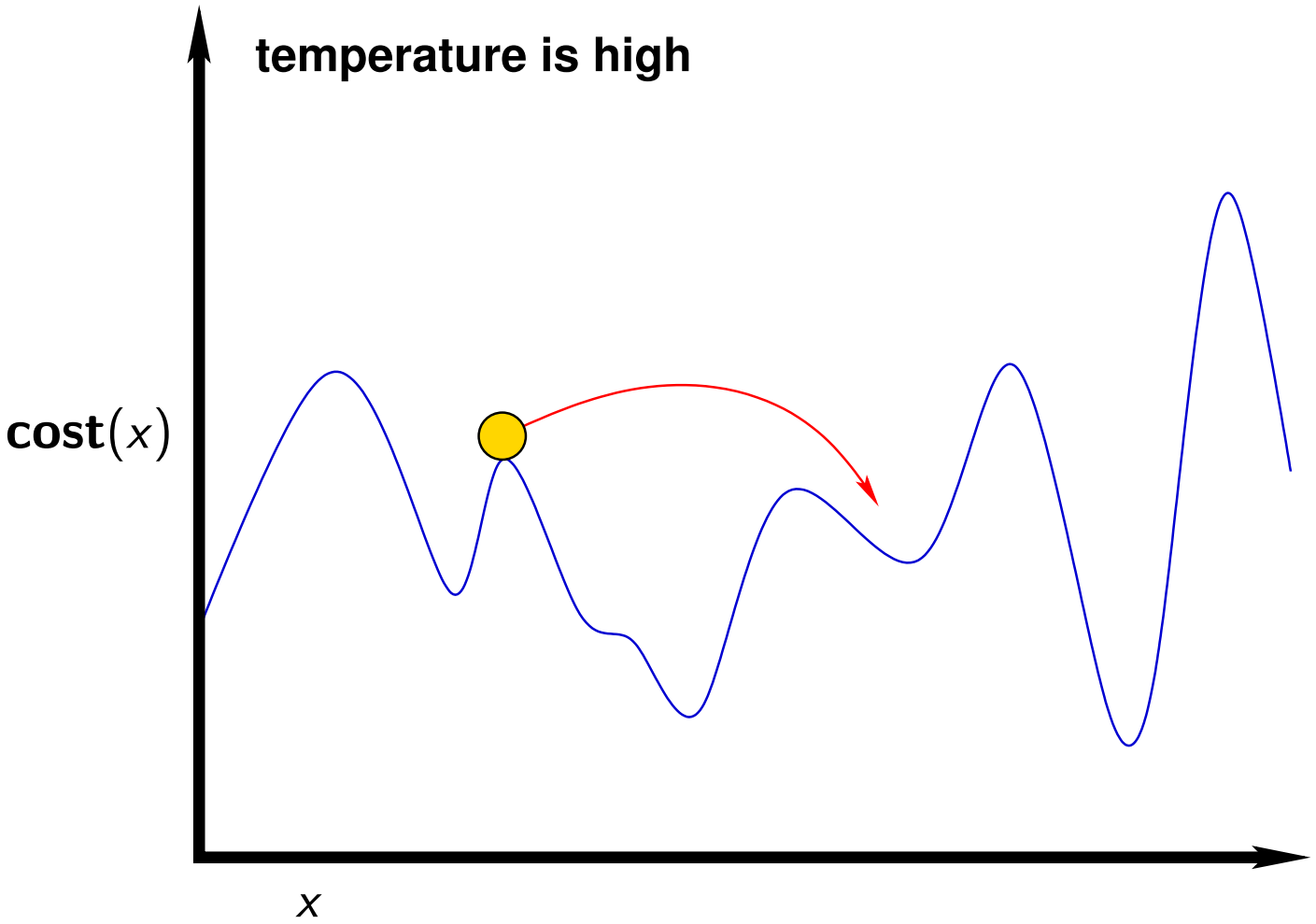
Simulated annealing example



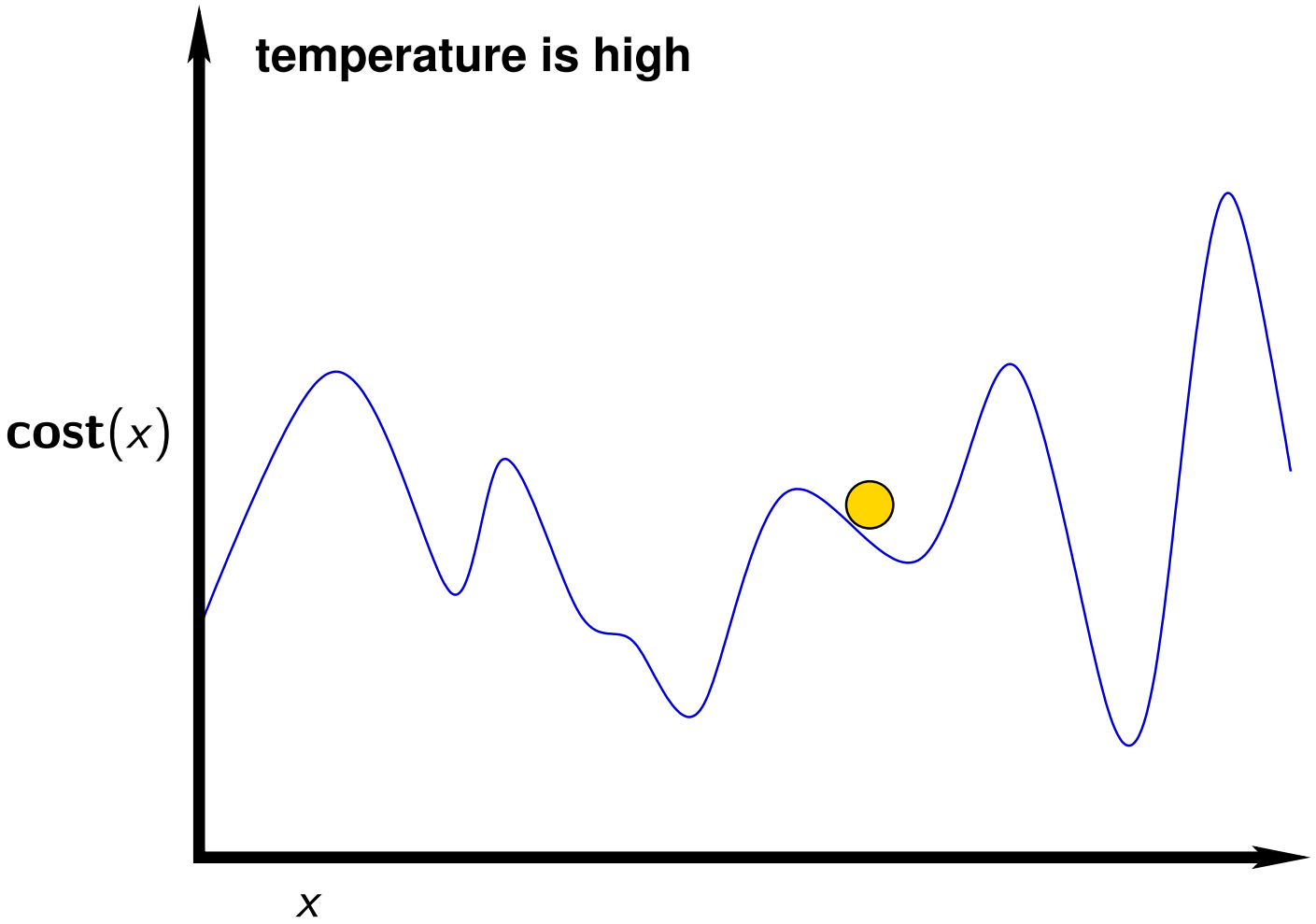
Simulated annealing example



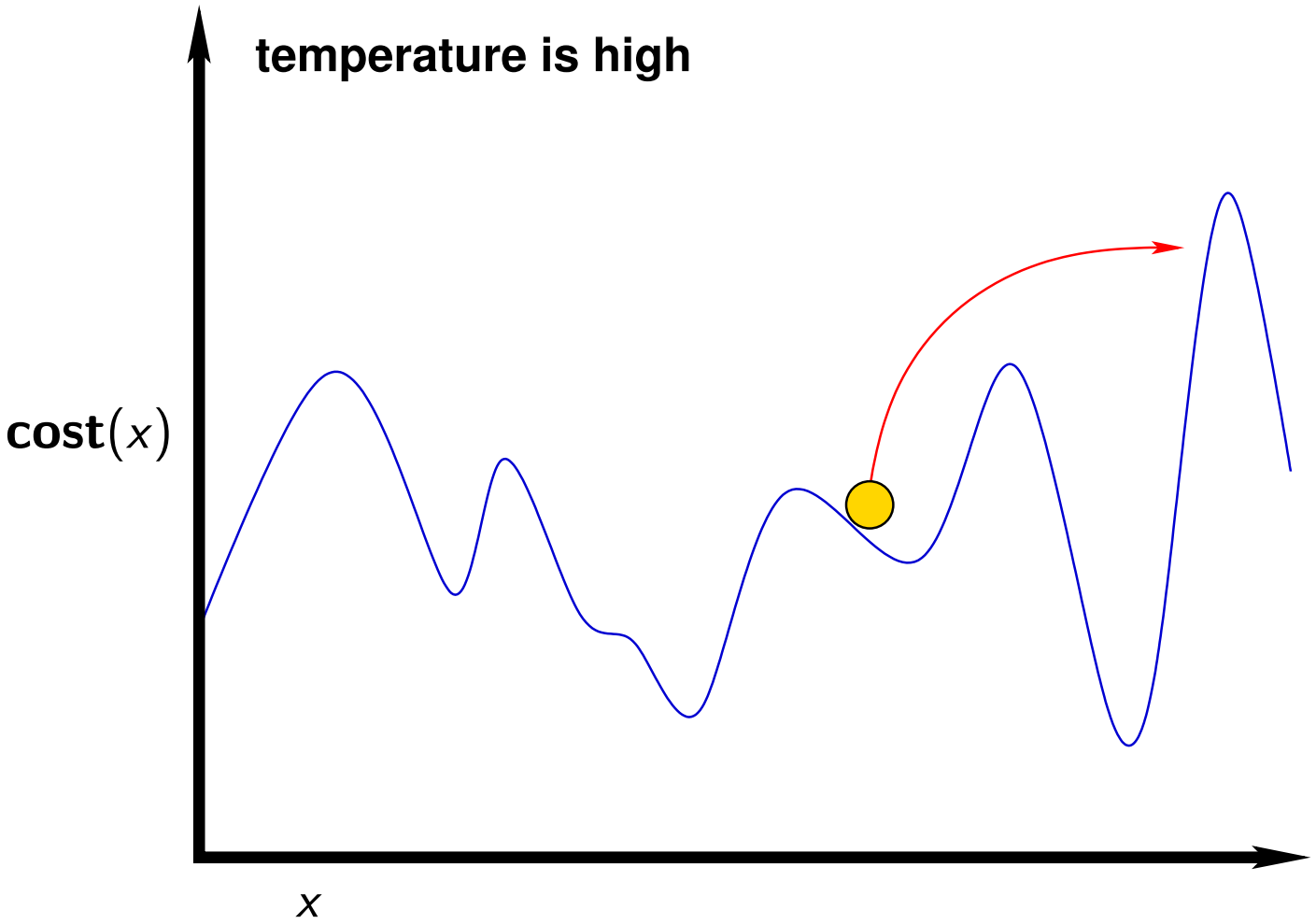
Simulated annealing example



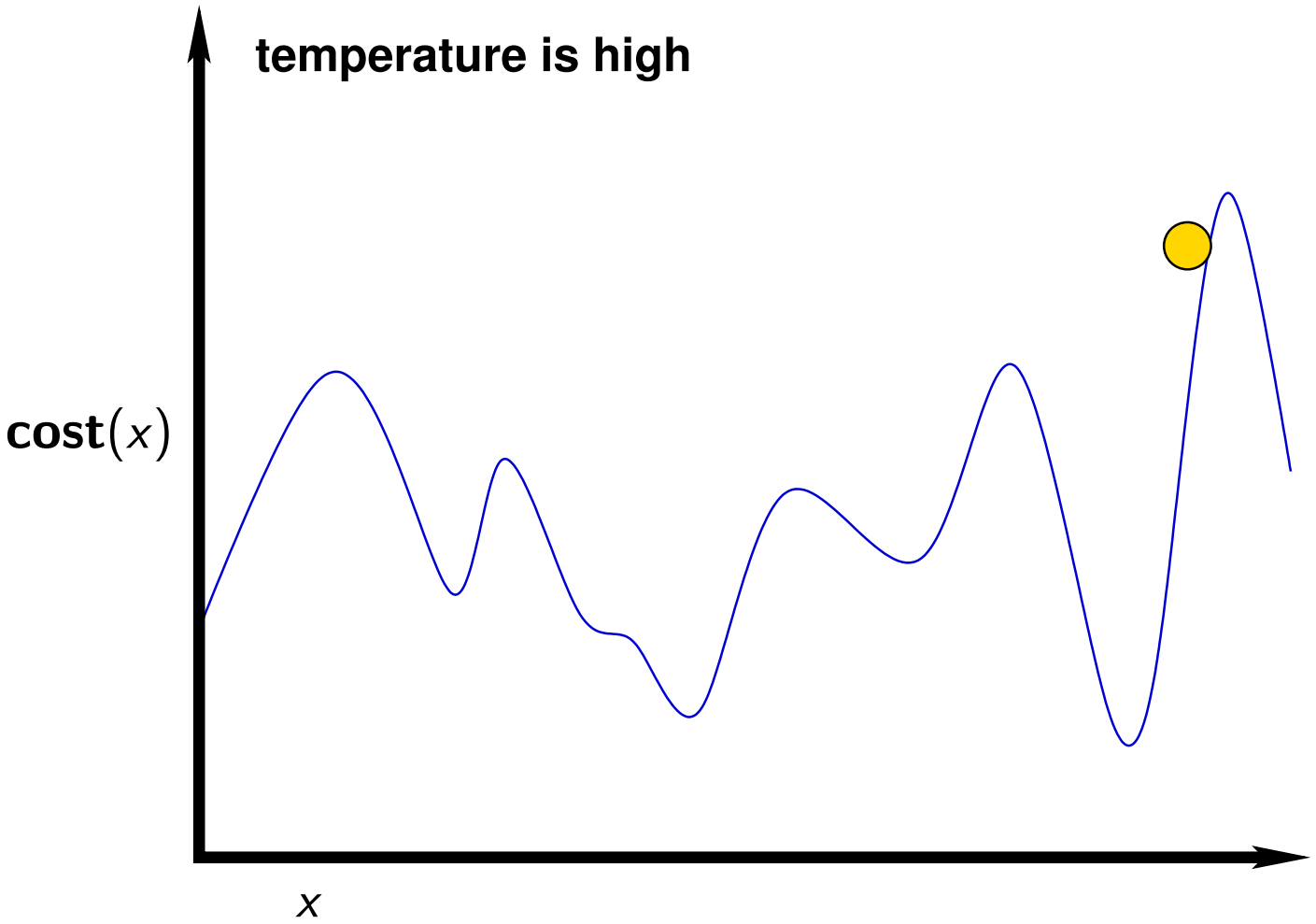
Simulated annealing example



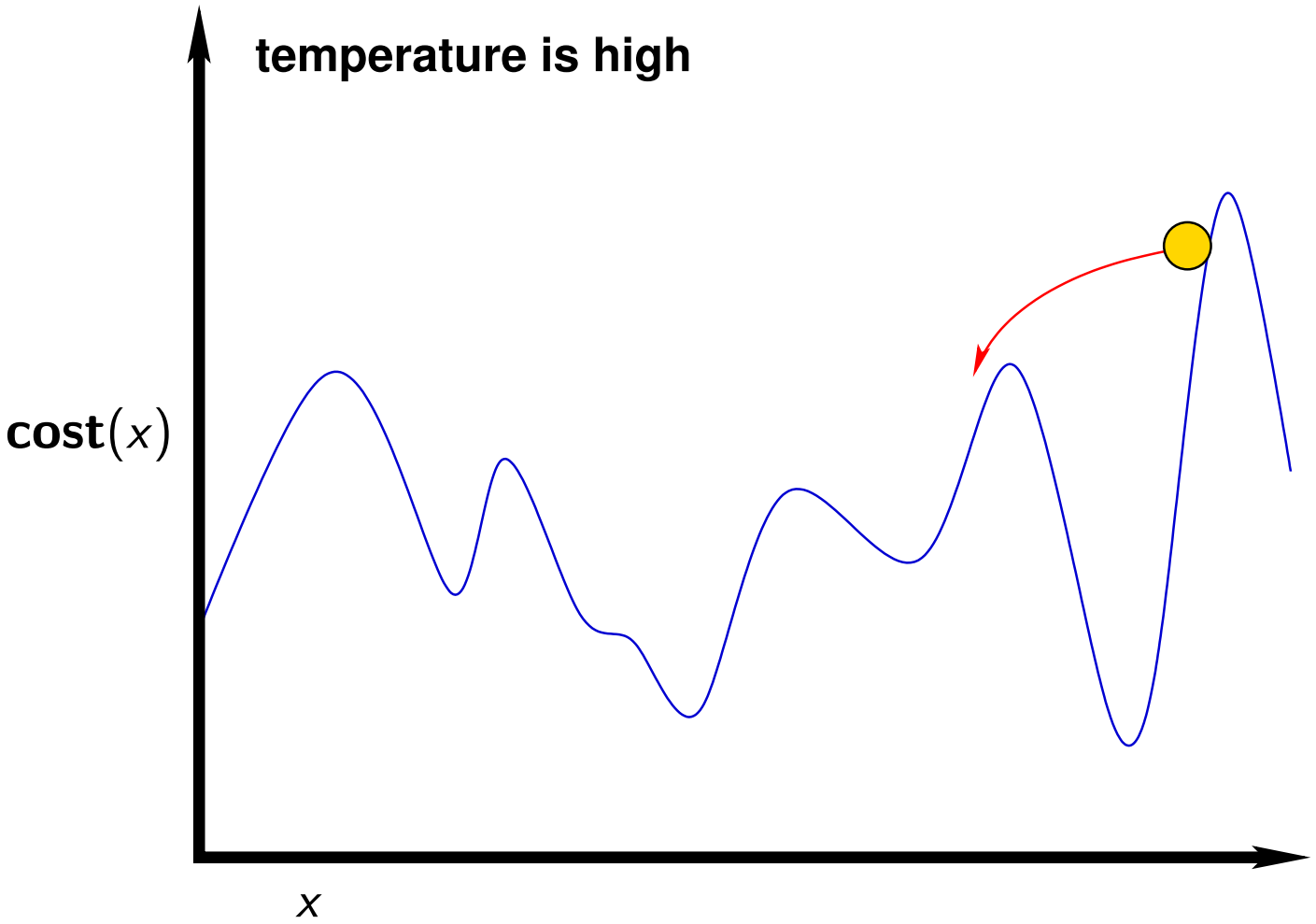
Simulated annealing example



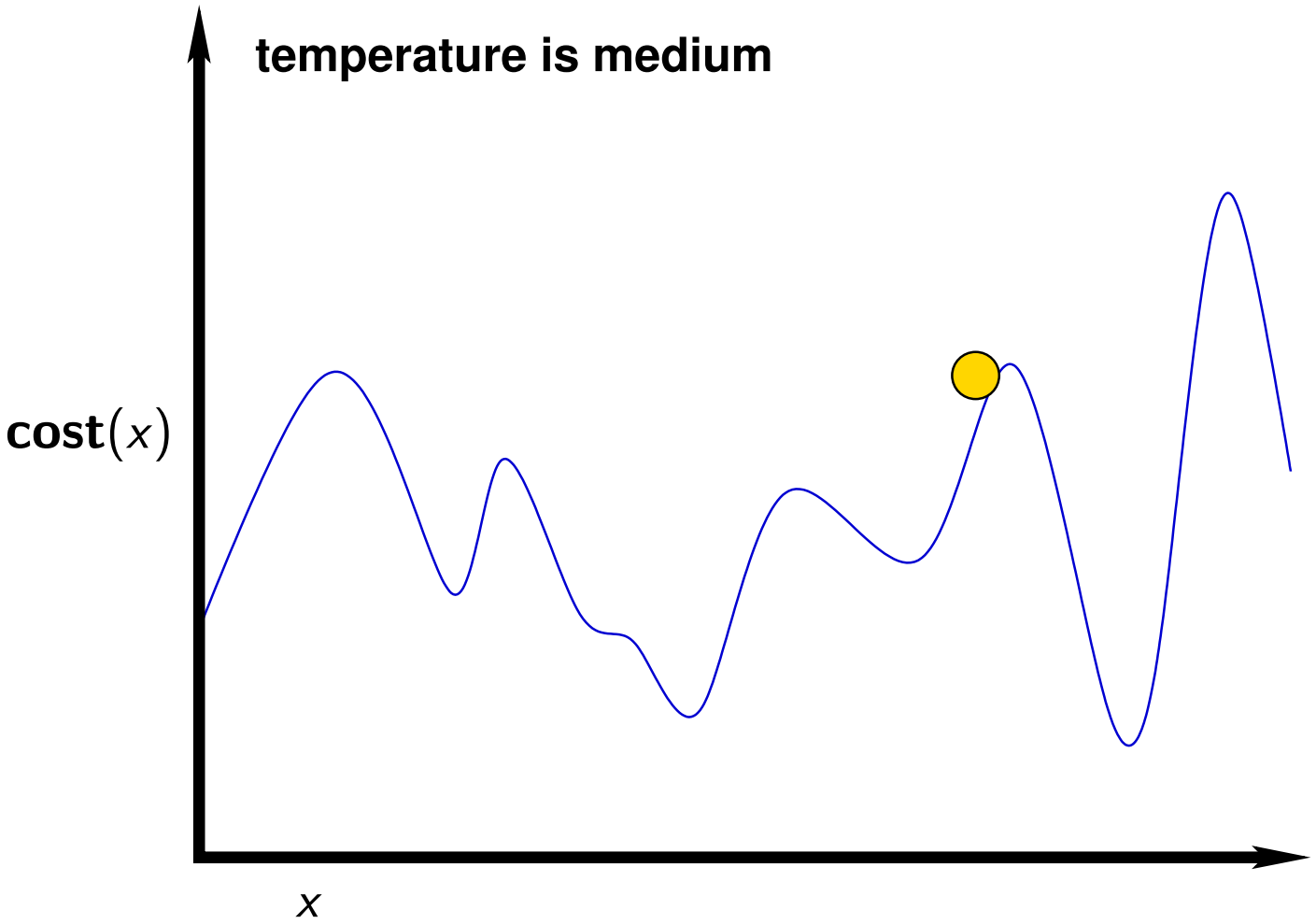
Simulated annealing example



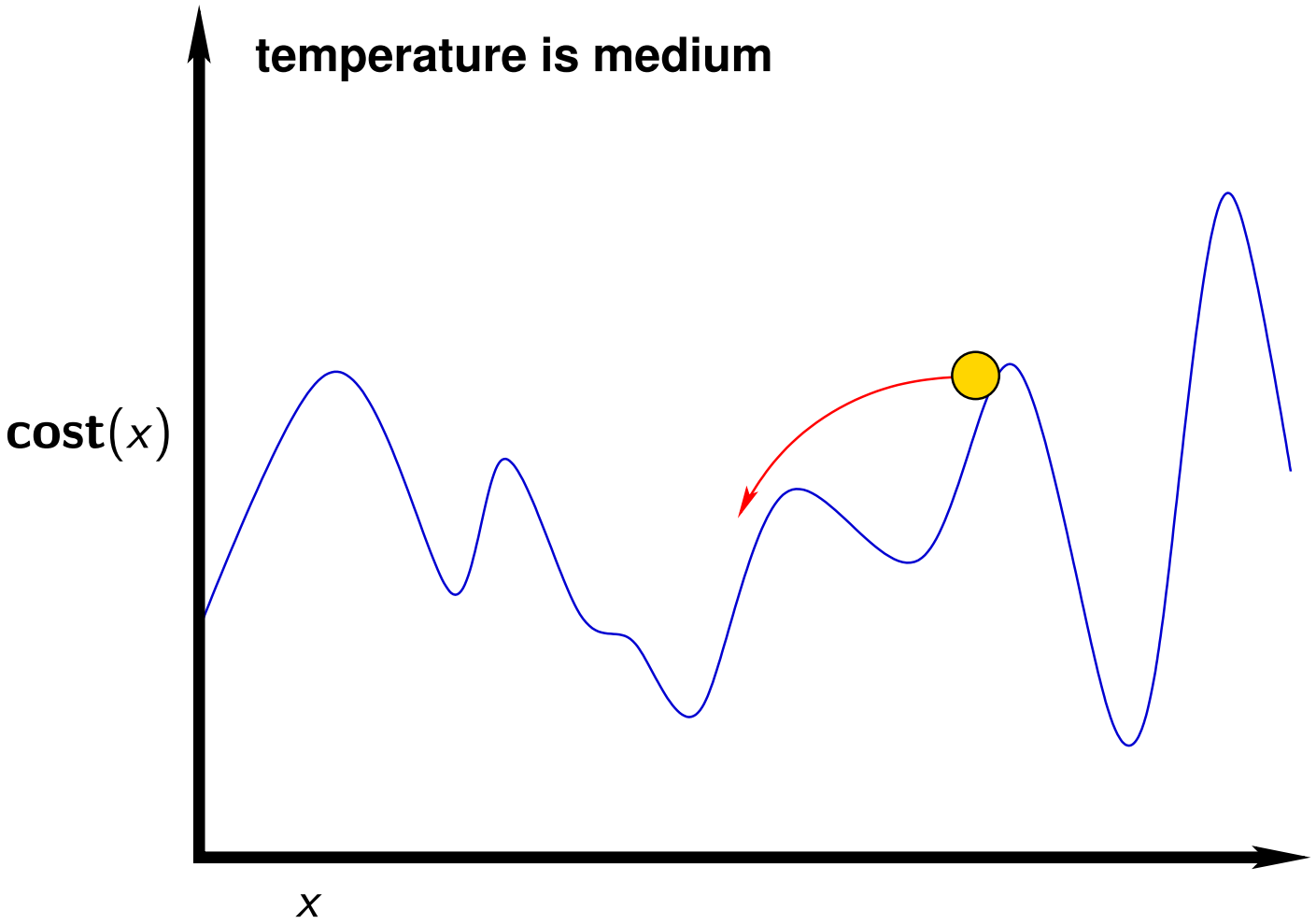
Simulated annealing example



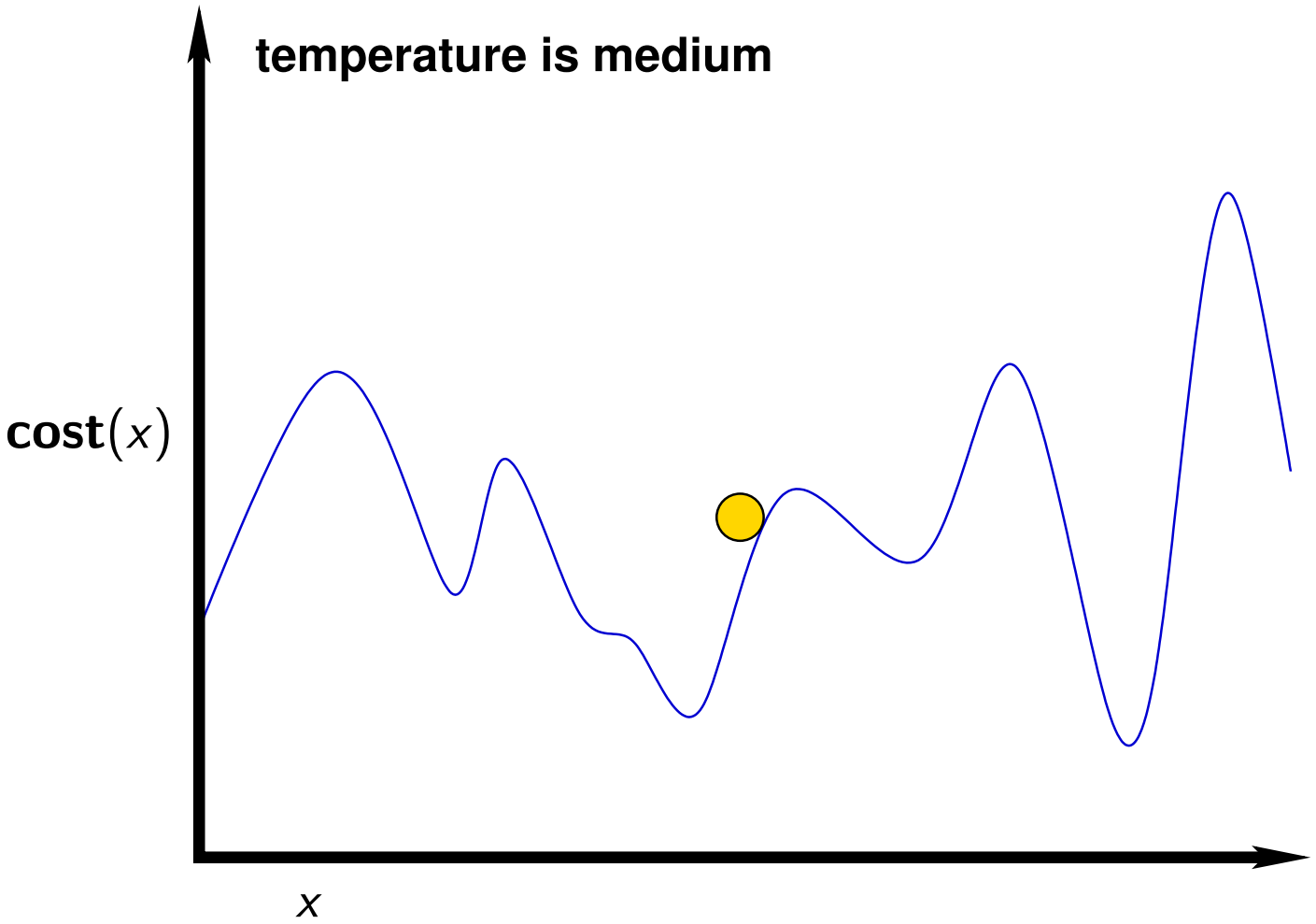
Simulated annealing example



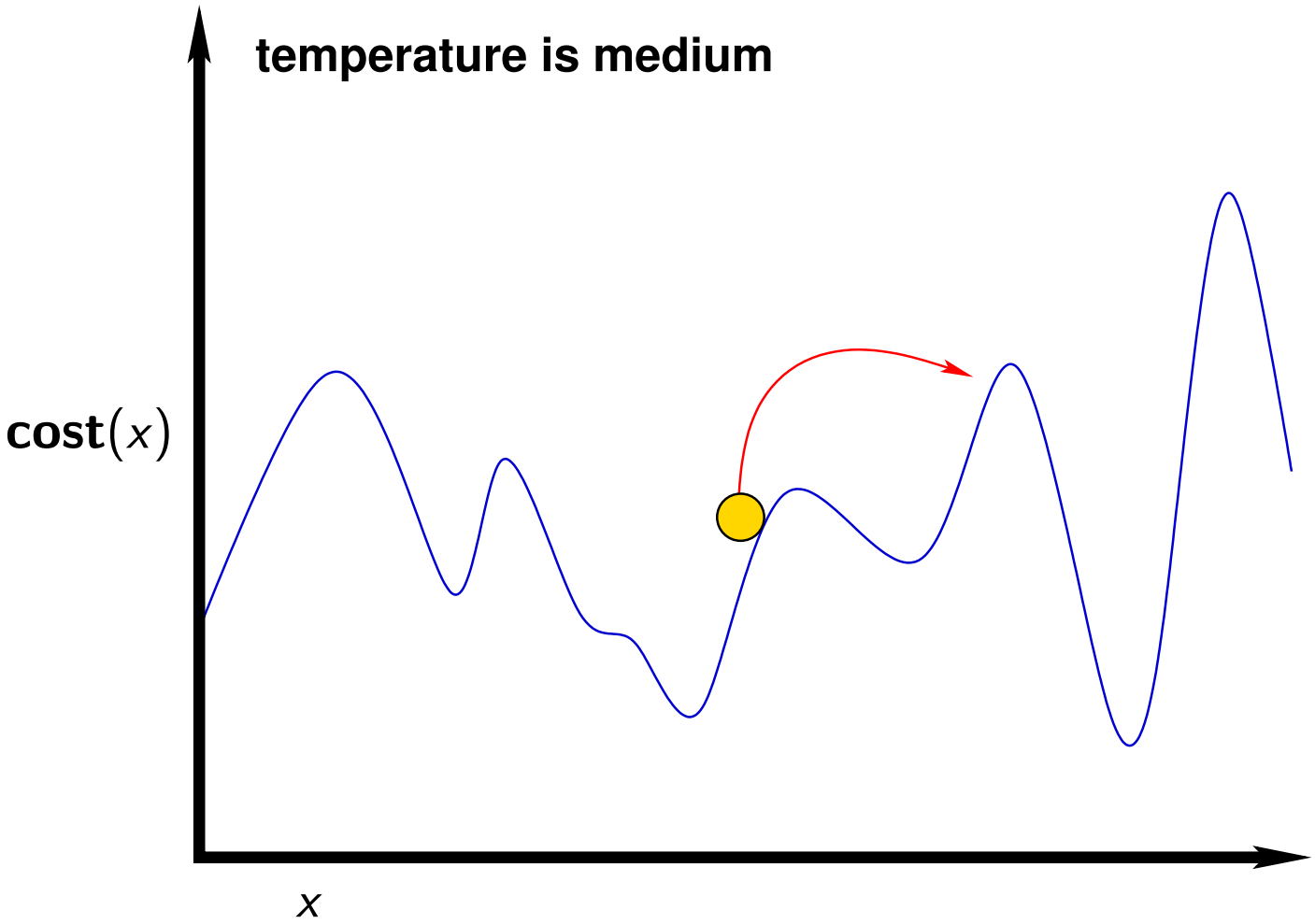
Simulated annealing example



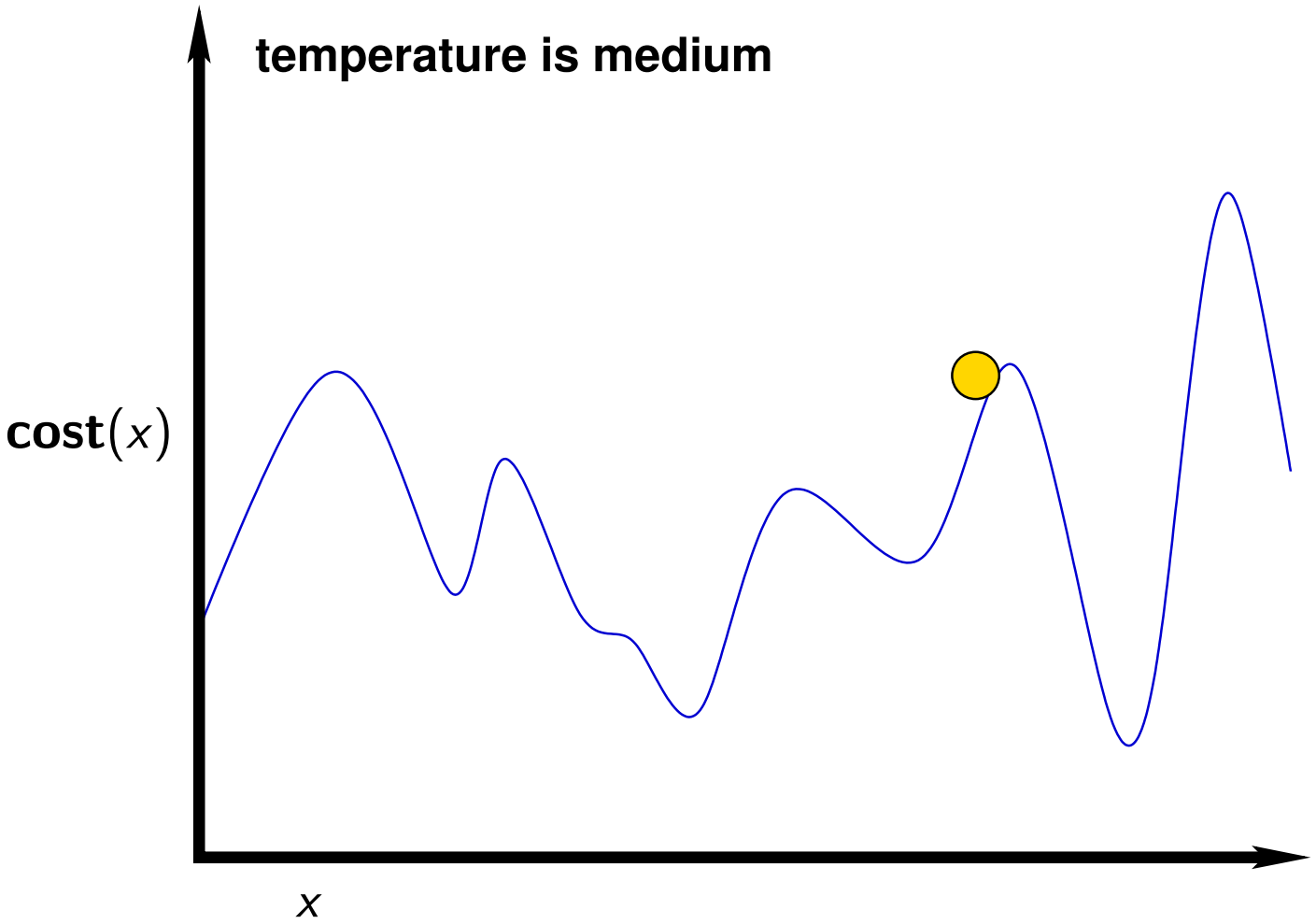
Simulated annealing example



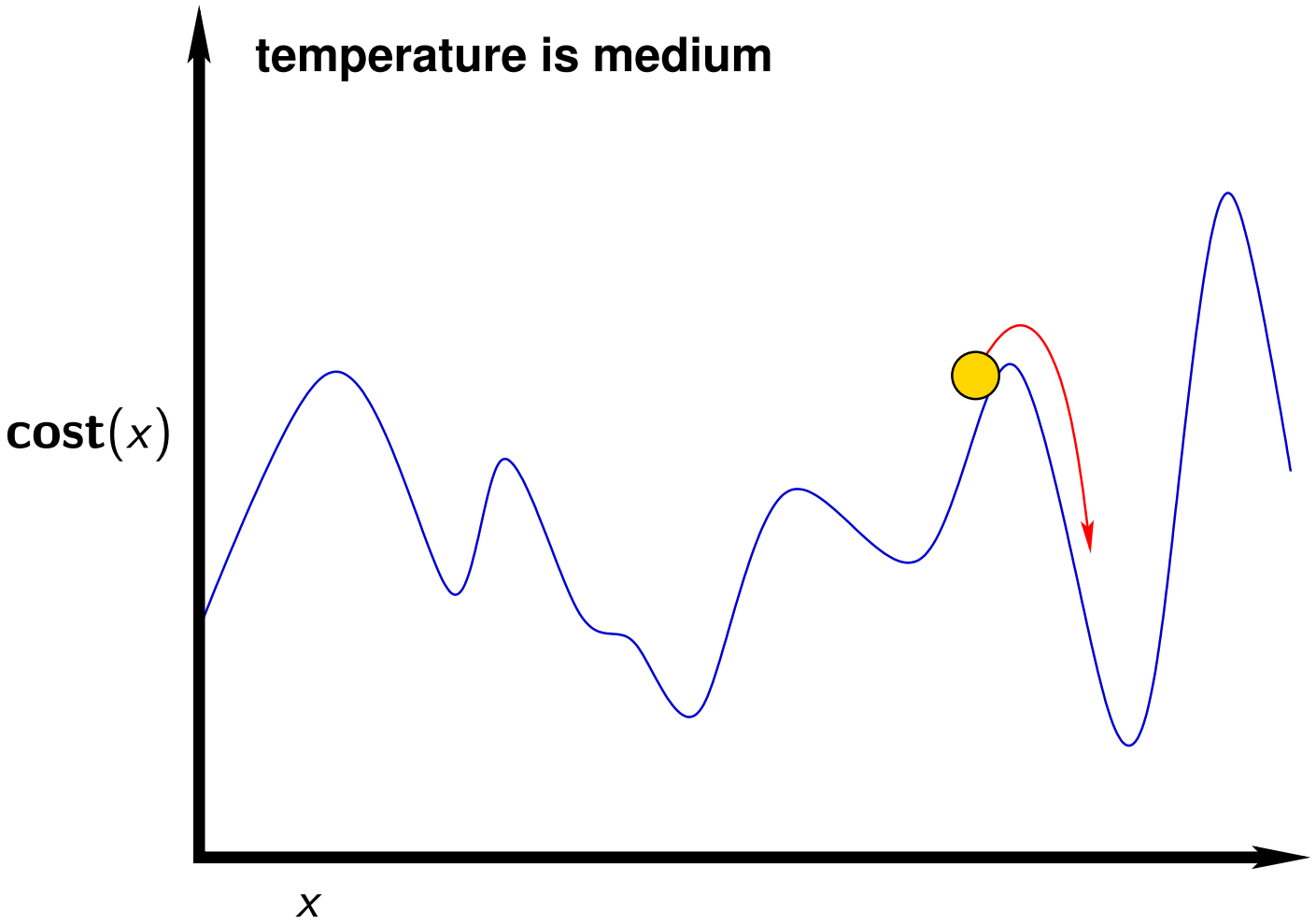
Simulated annealing example



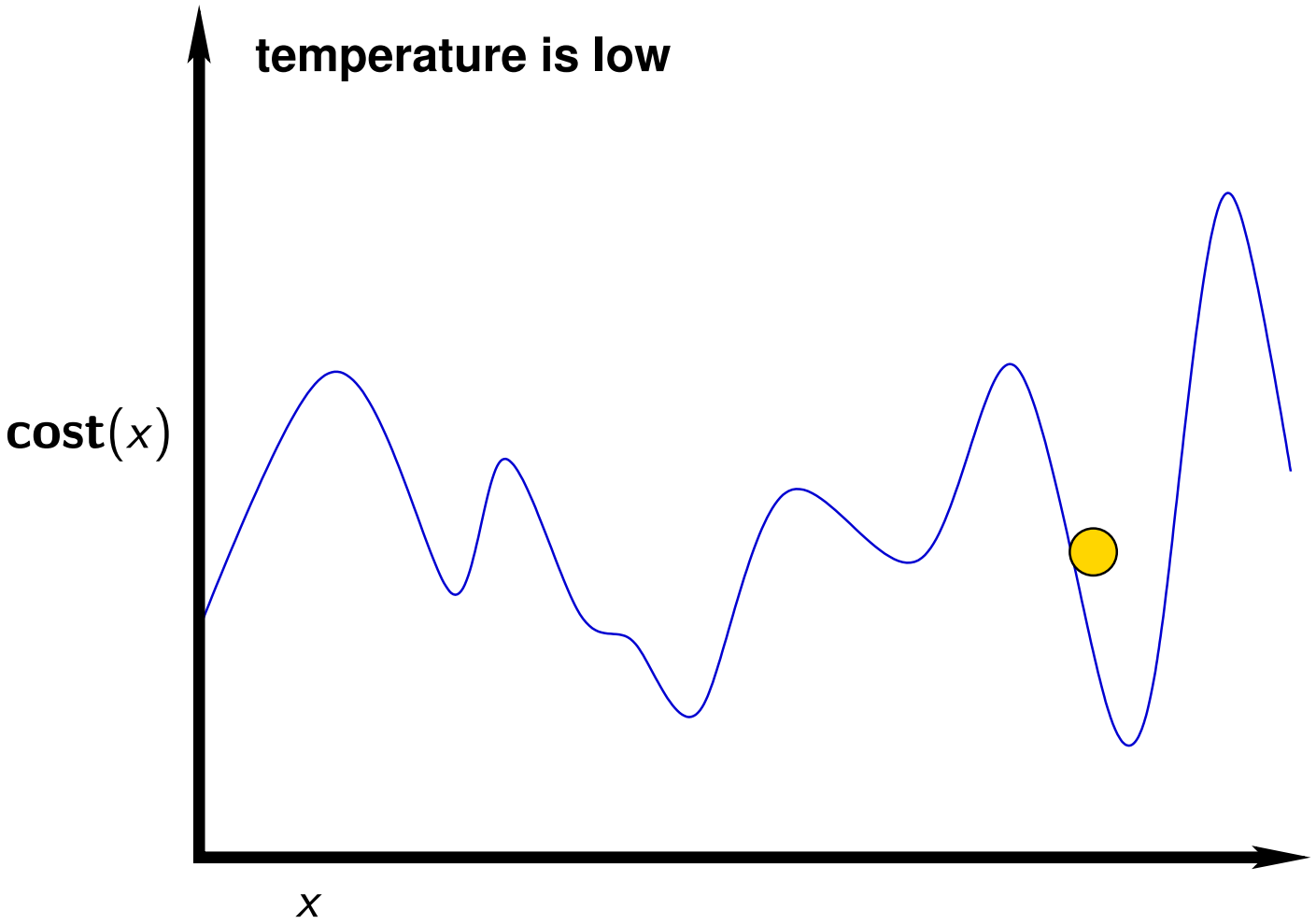
Simulated annealing example



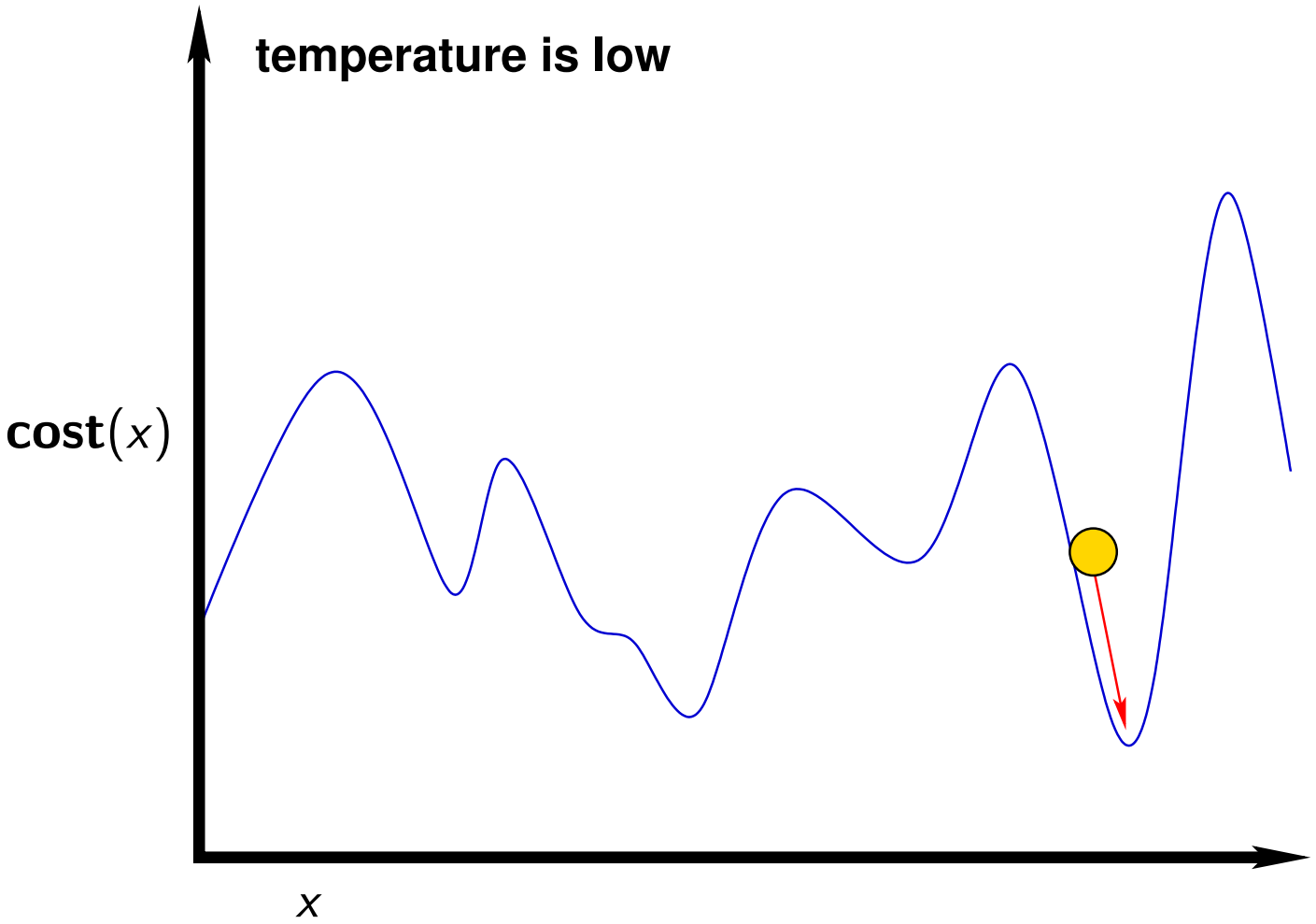
Simulated annealing example



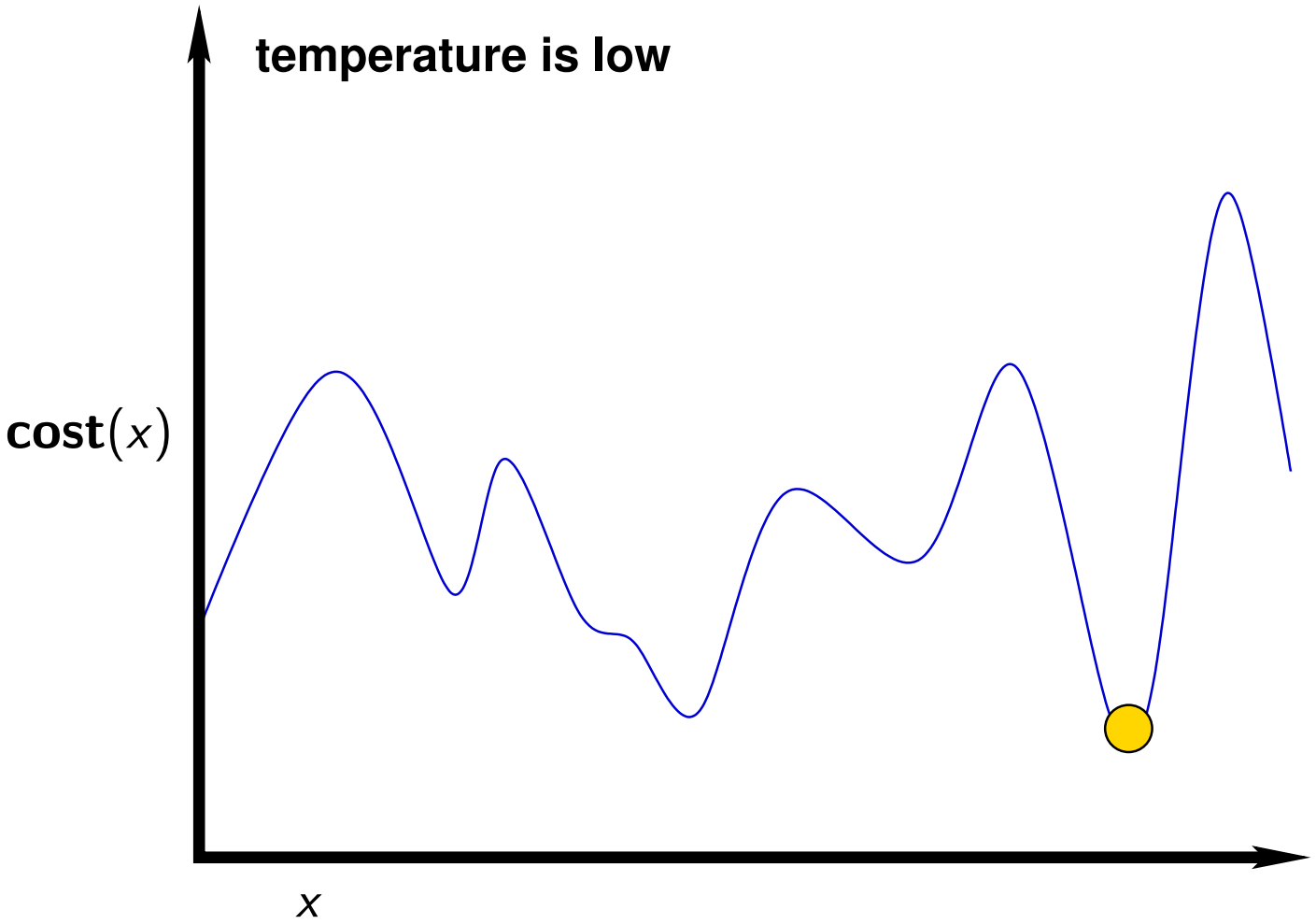
Simulated annealing example



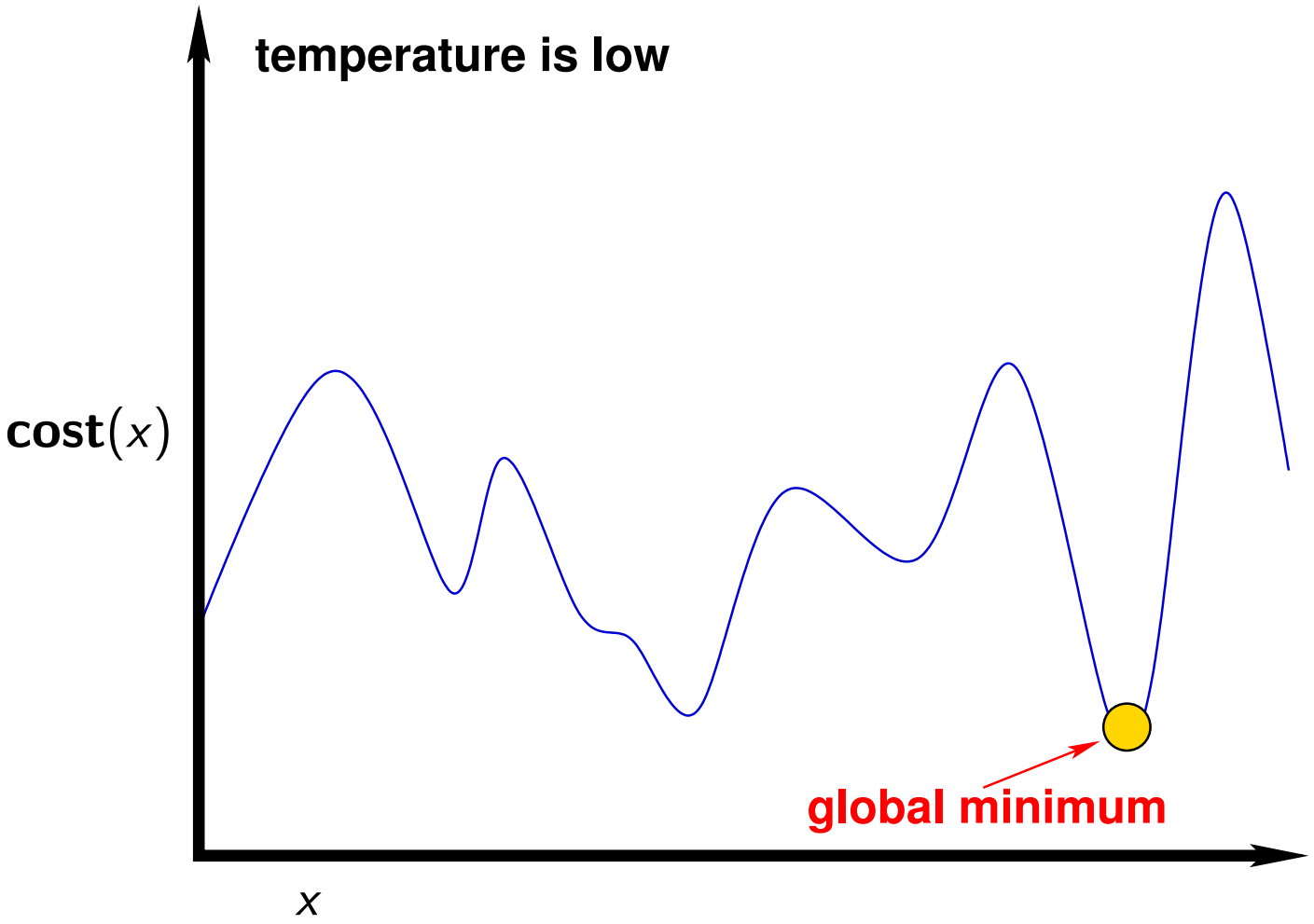
Simulated annealing example



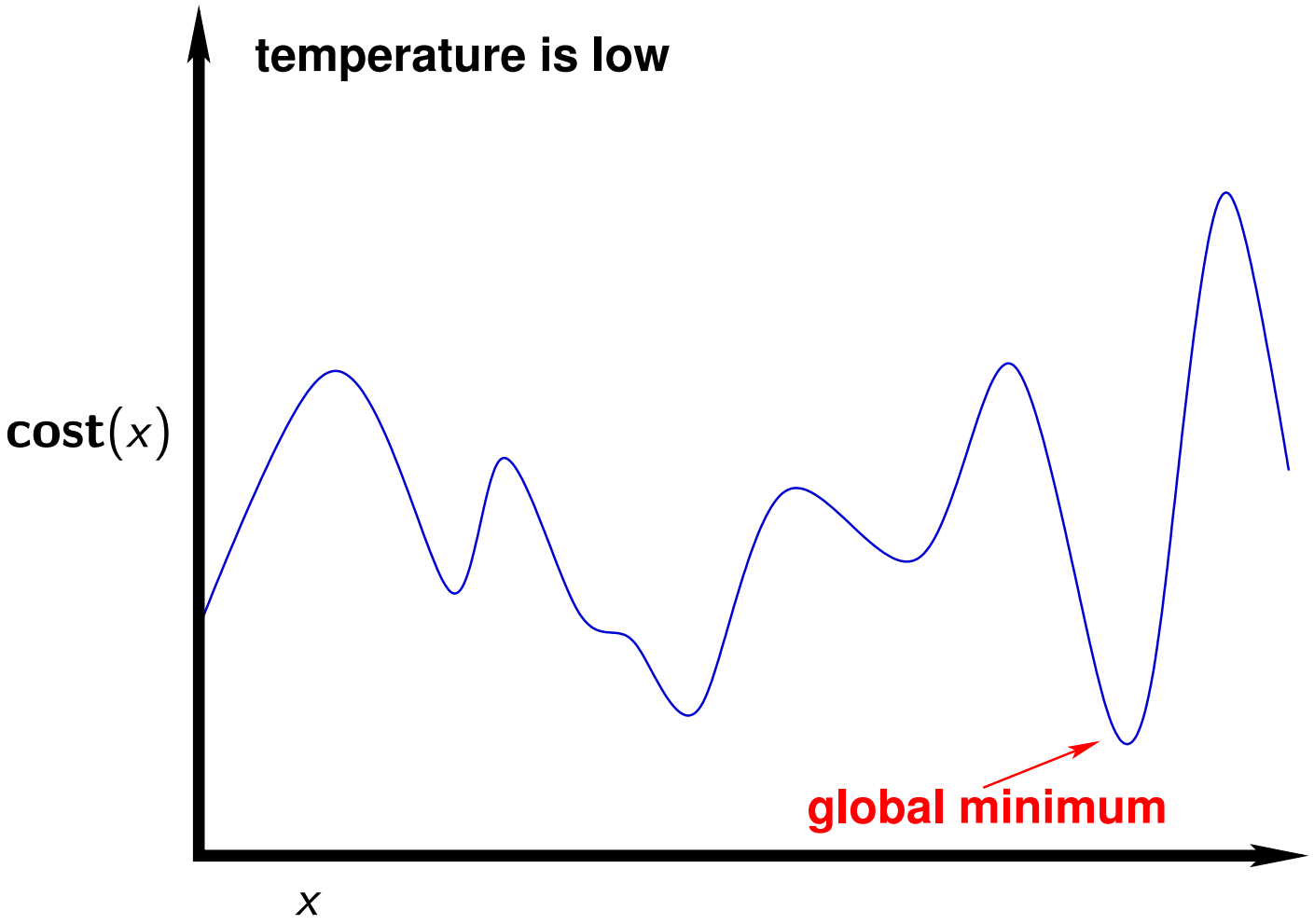
Simulated annealing example



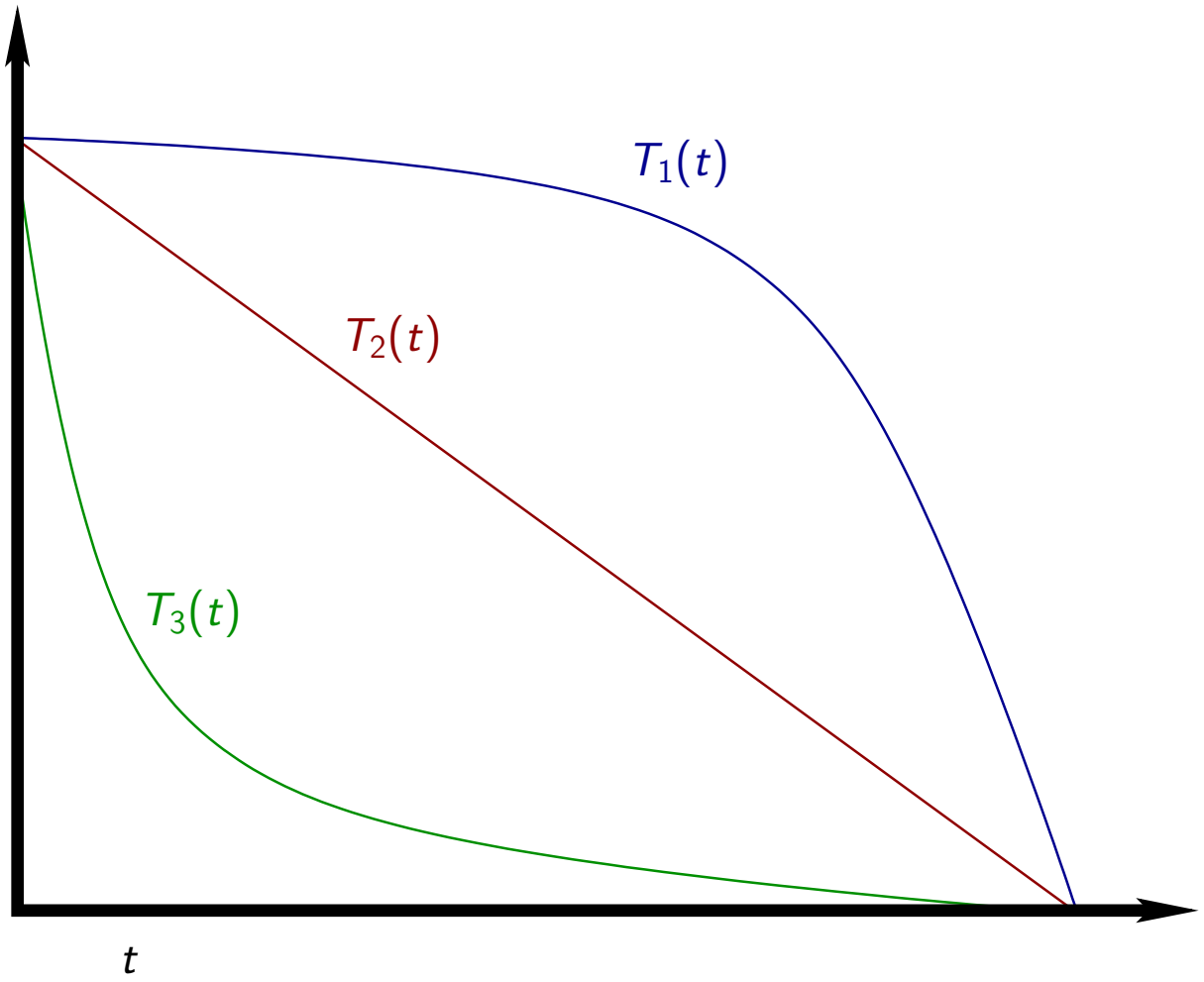
Simulated annealing example



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 anealing 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.

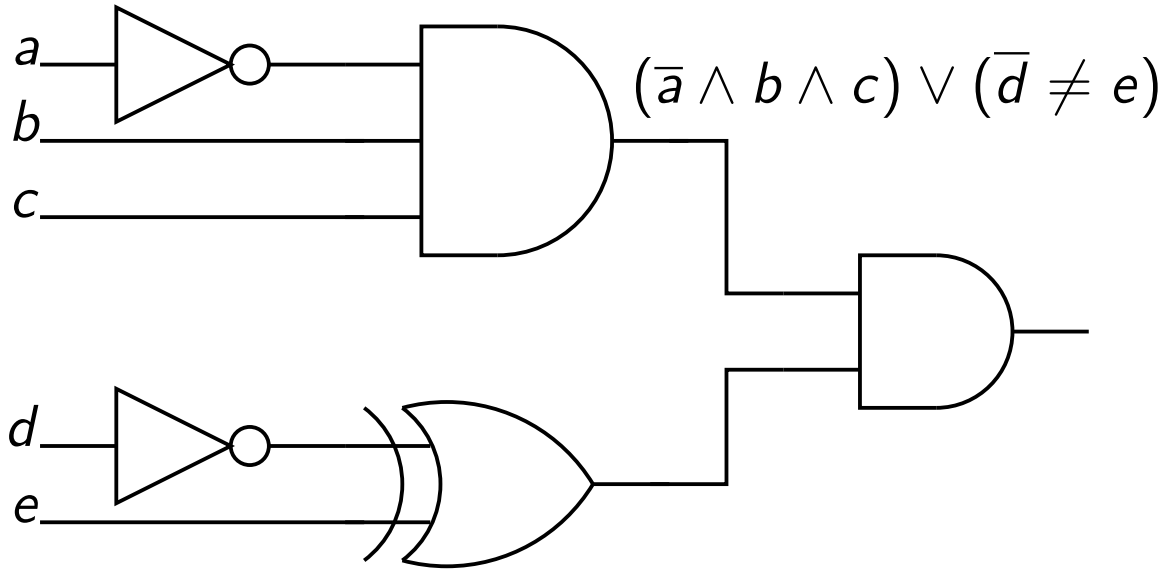
Good at escaping sub-optimal local minima.

Greedy iterative improvement if no information sharing.

Difficult to implement and analyze.

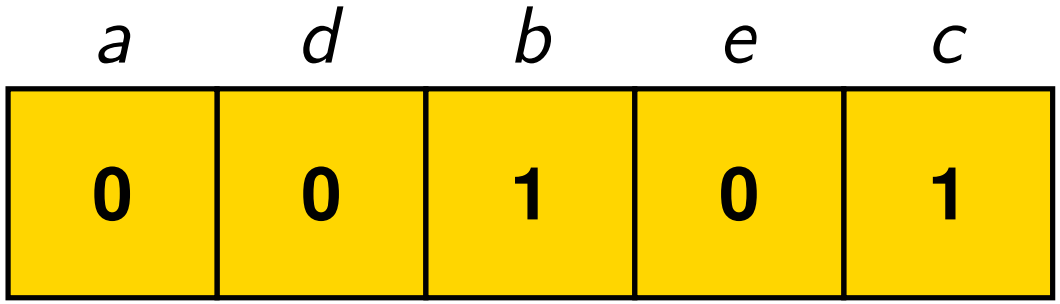
Researchers have applied in testing, system synthesis.

Solution representation



a	d	b	e	c
0	0	1	0	1

Solution representation



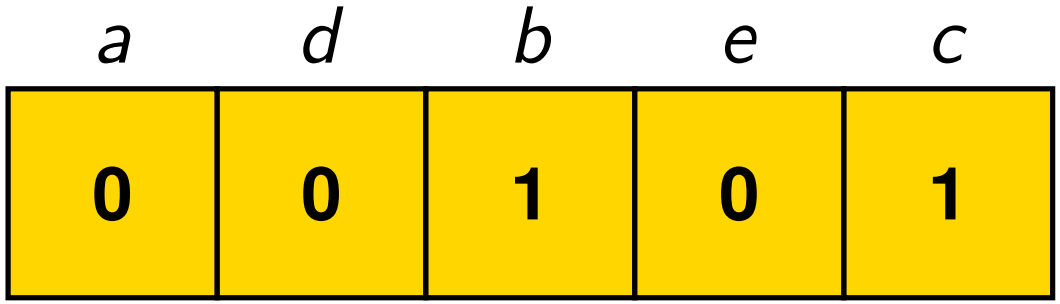
Mutation

Choose an element of the solution.

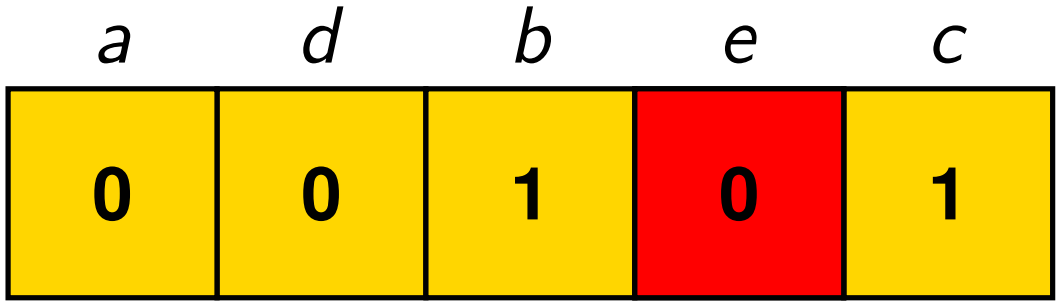
Change it to another value.

Local modification, similar to that in iterative improvement.

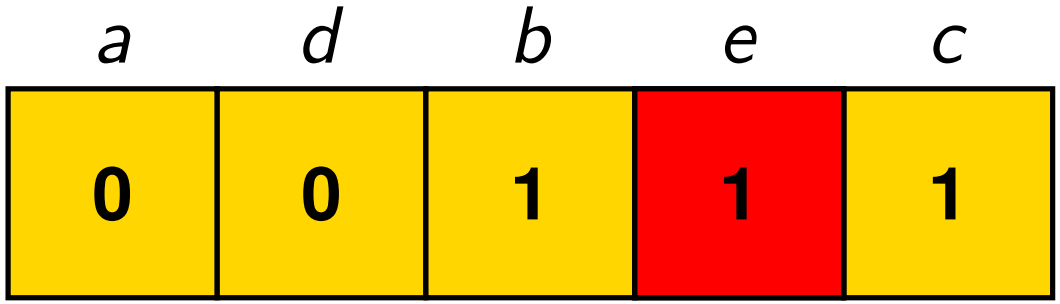
Mutation



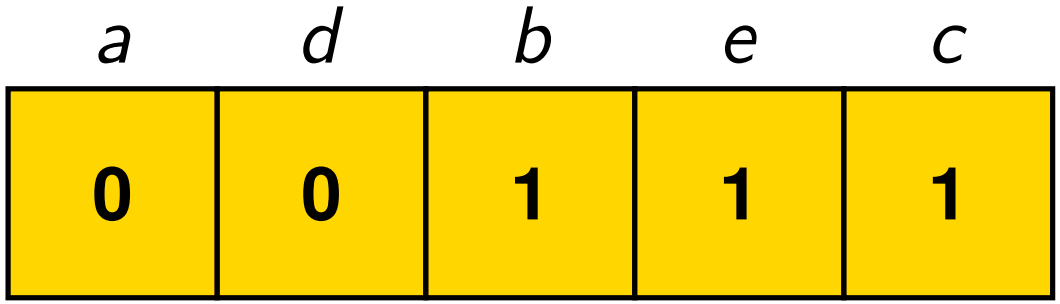
Mutation



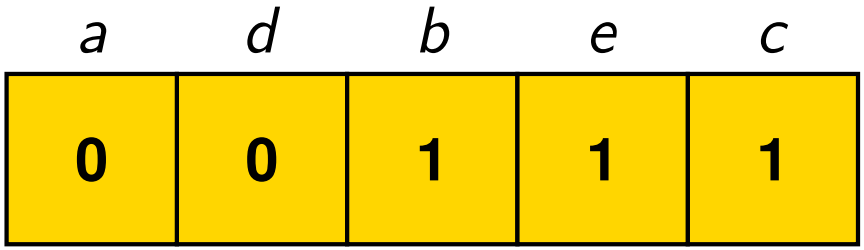
Mutation



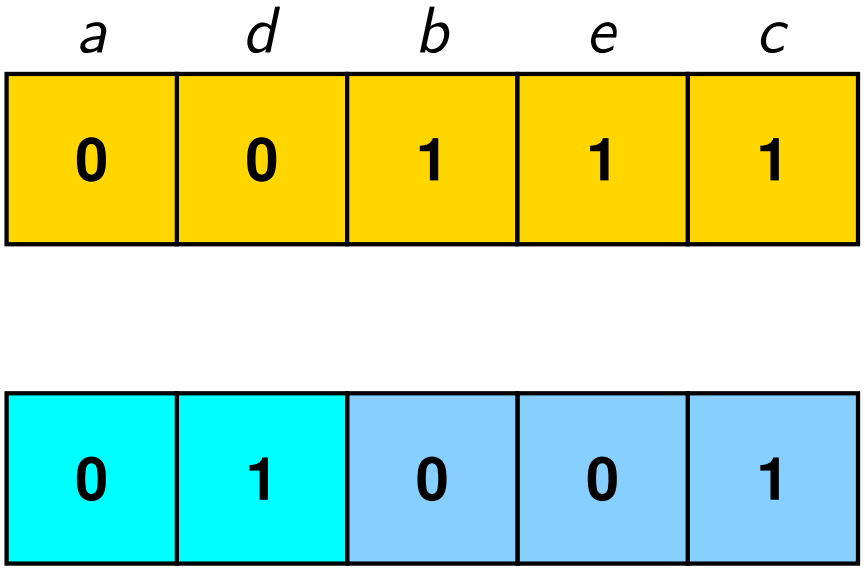
Mutation



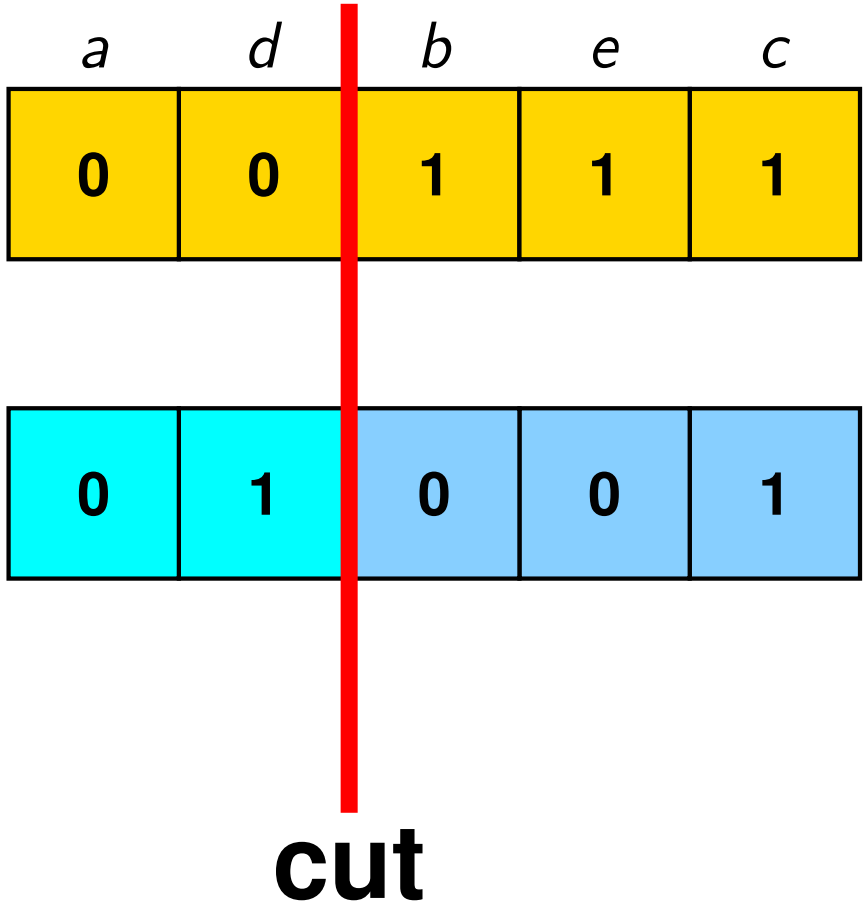
Crossover



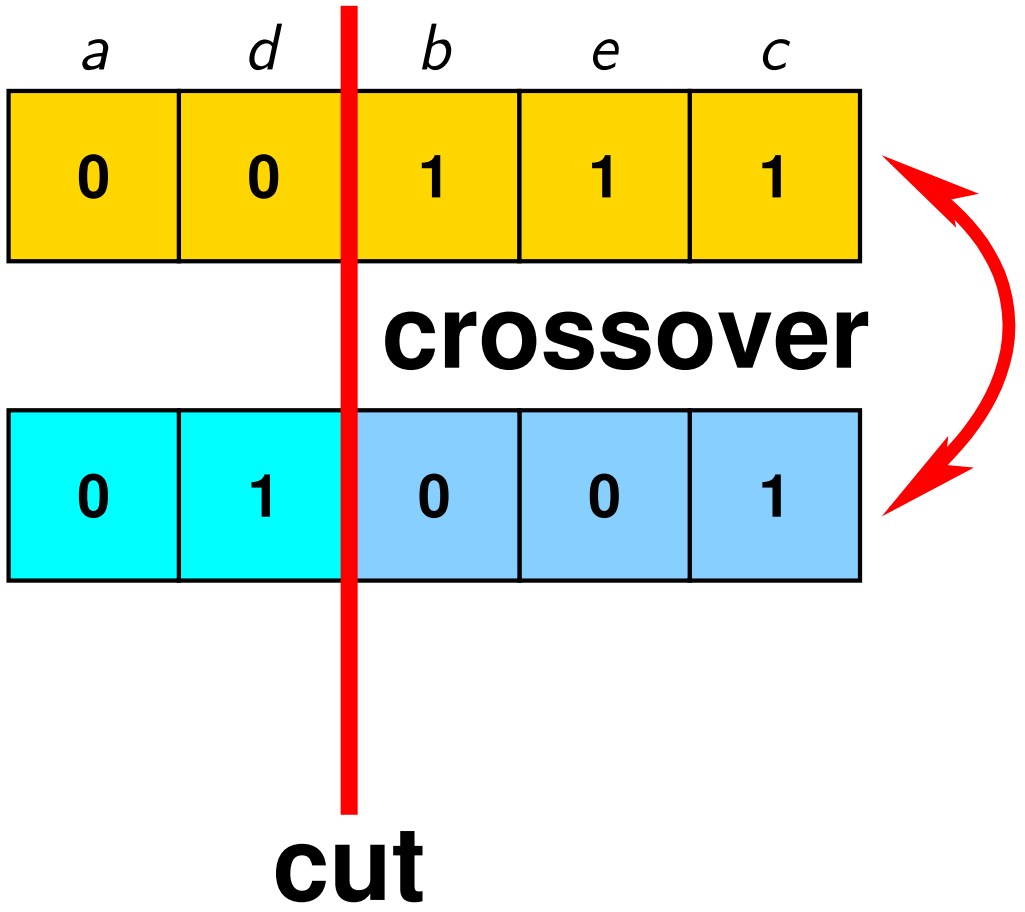
Crossover



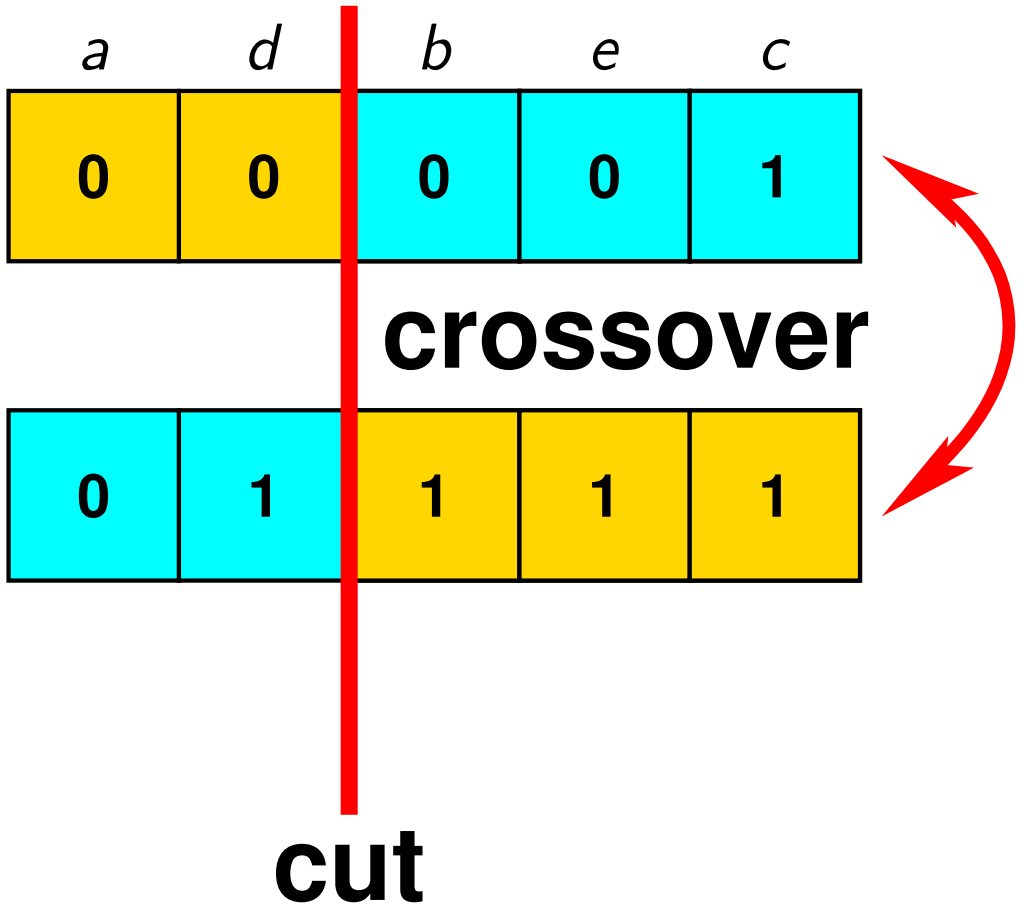
Crossover



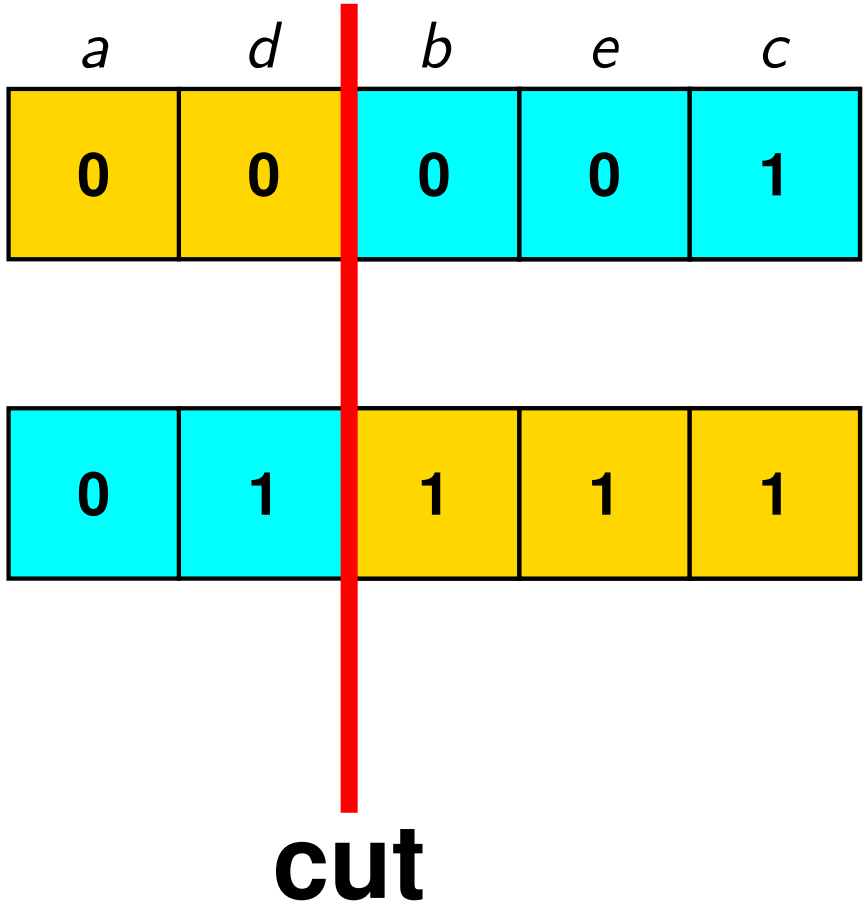
Crossover



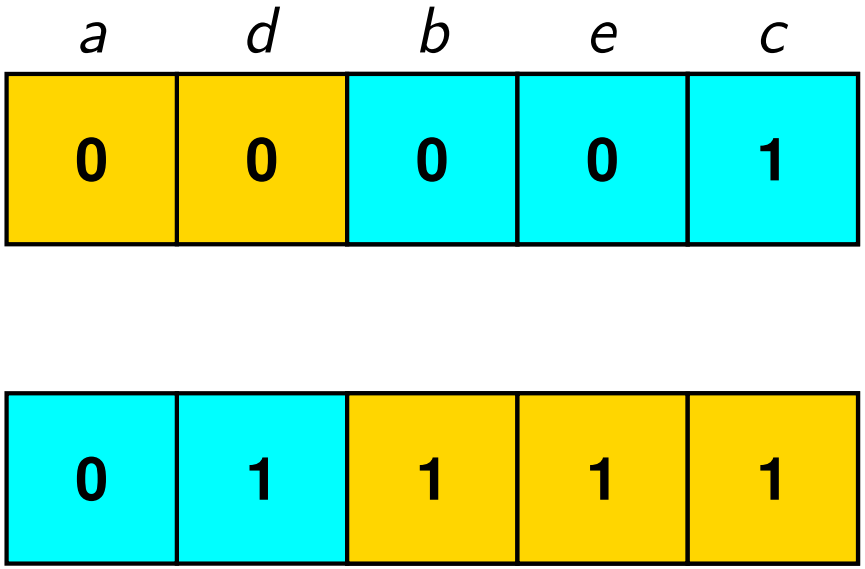
Crossover



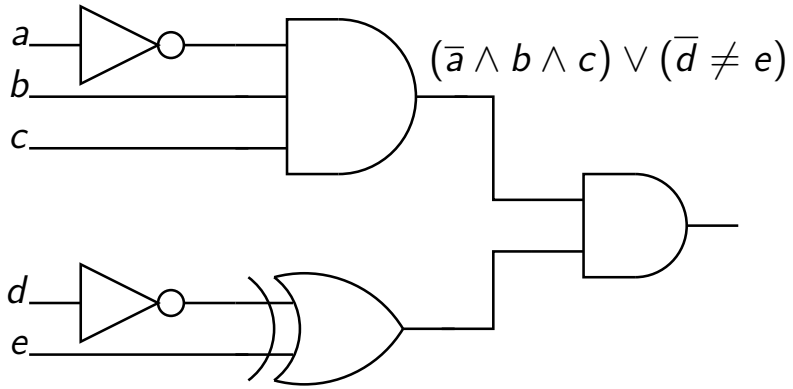
Crossover



Crossover



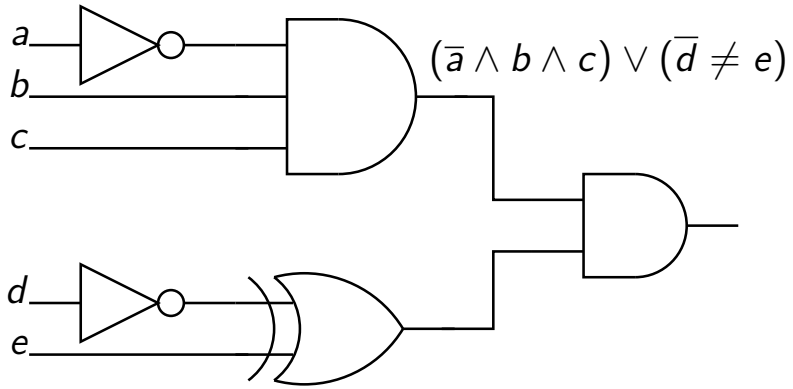
Locality not preserved



a	d	b	e	c
0	0	0	0	1

0	1	1	1	1

Locality not preserved



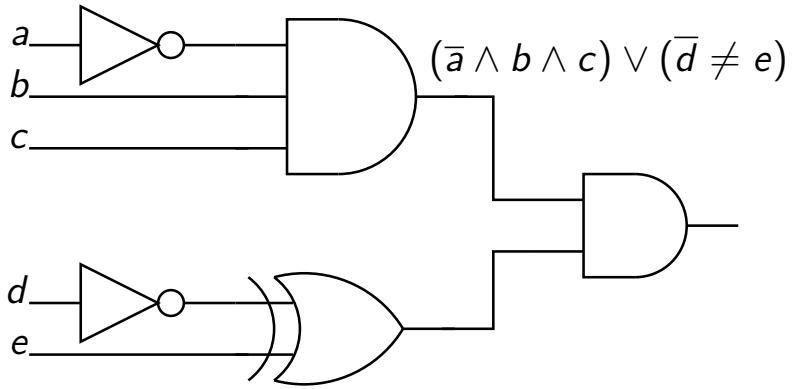
a	d	b	e	c
0	0	0	0	1

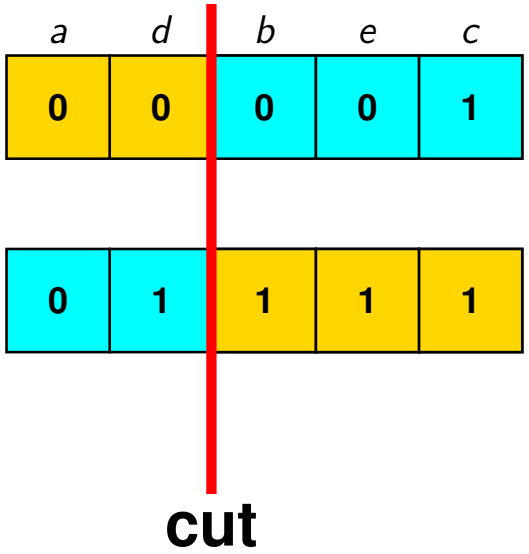
good soln to XOR2

0	1	1	1	1

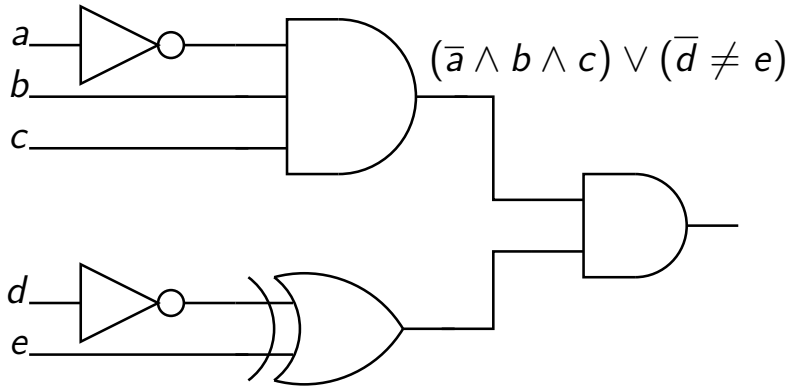
good soln to AND3

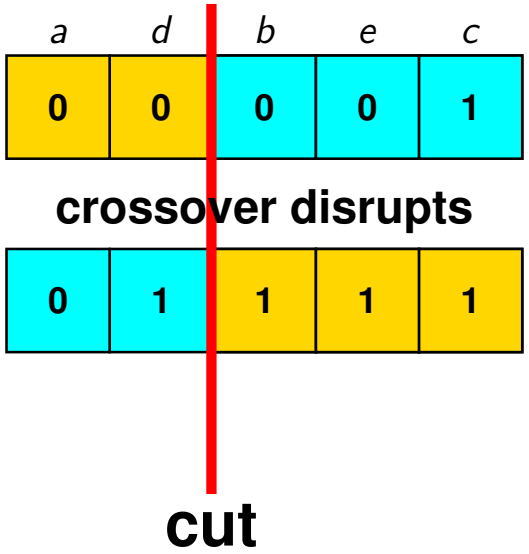
Locality not preserved



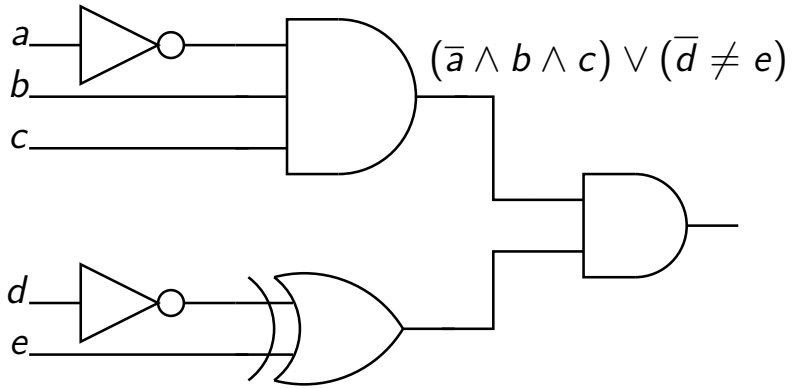


Locality not preserved





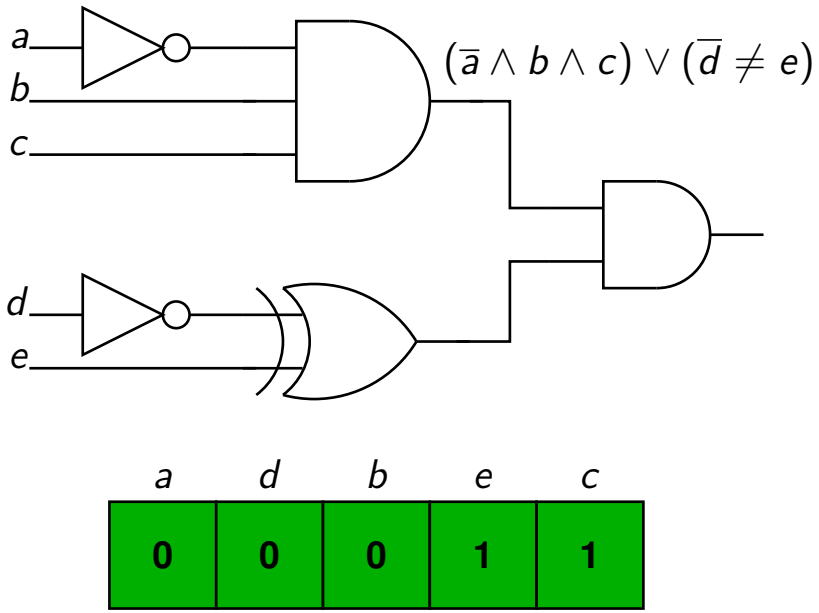
Locality not preserved



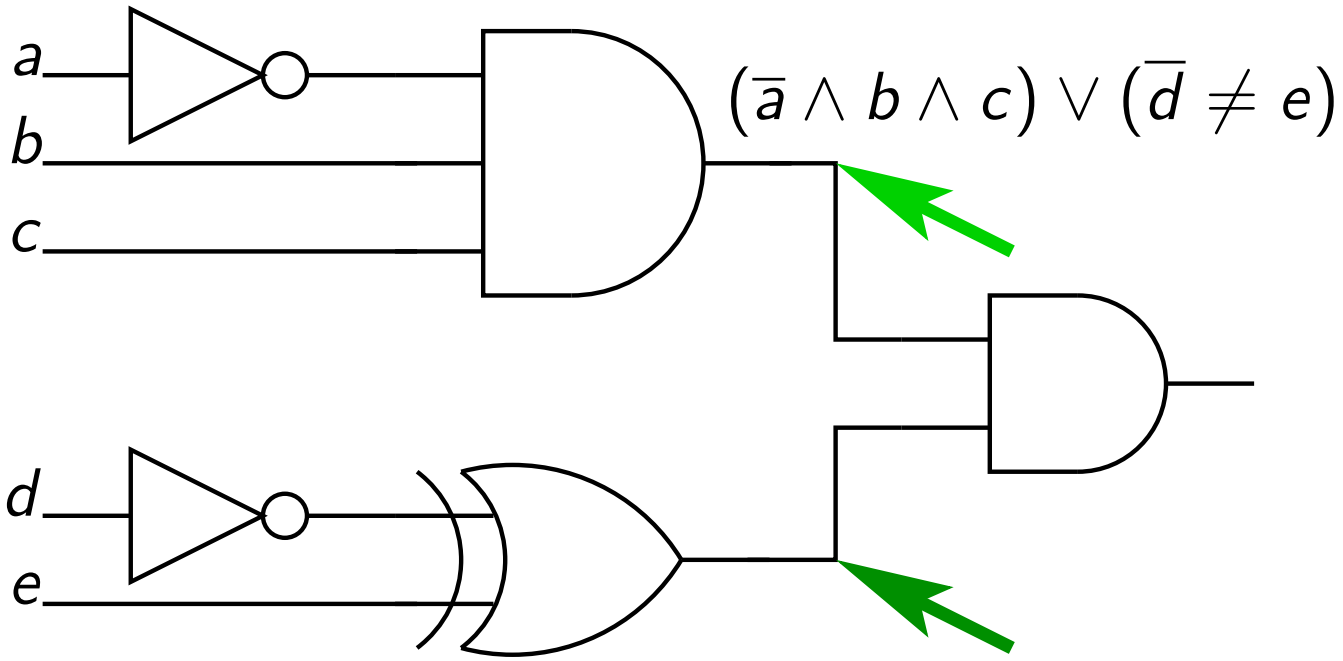
a	d	b	e	c
0	0	0	0	1

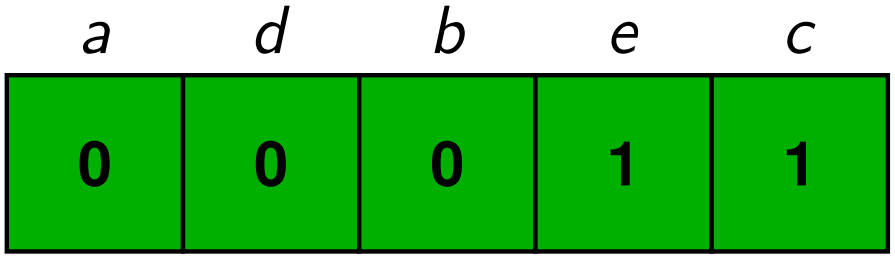
0	1	1	1	1

Locality preserved

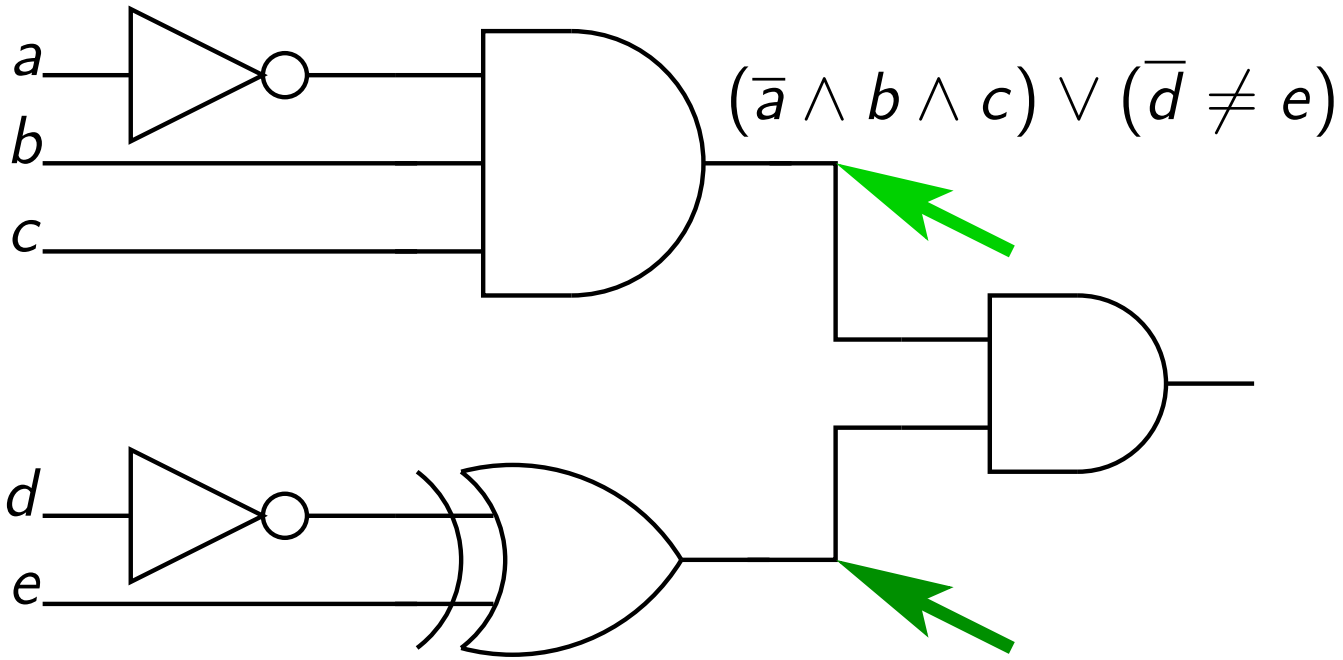


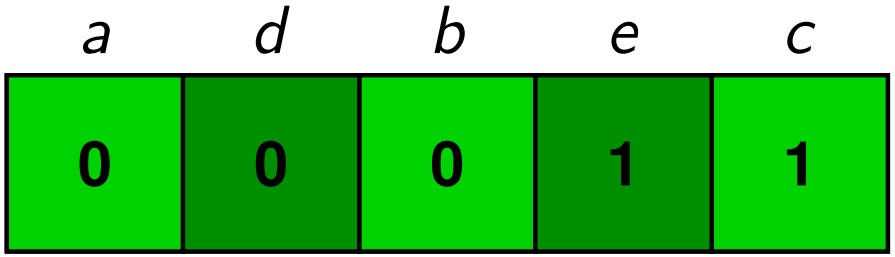
Locality preserved



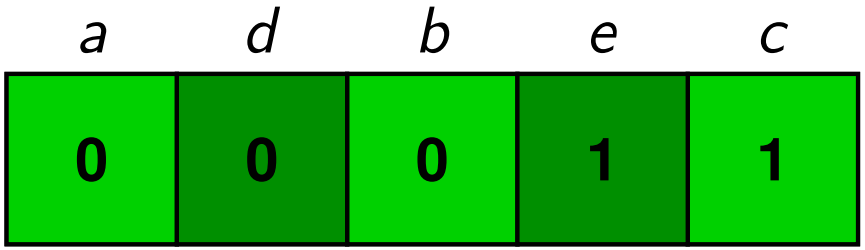


Locality preserved

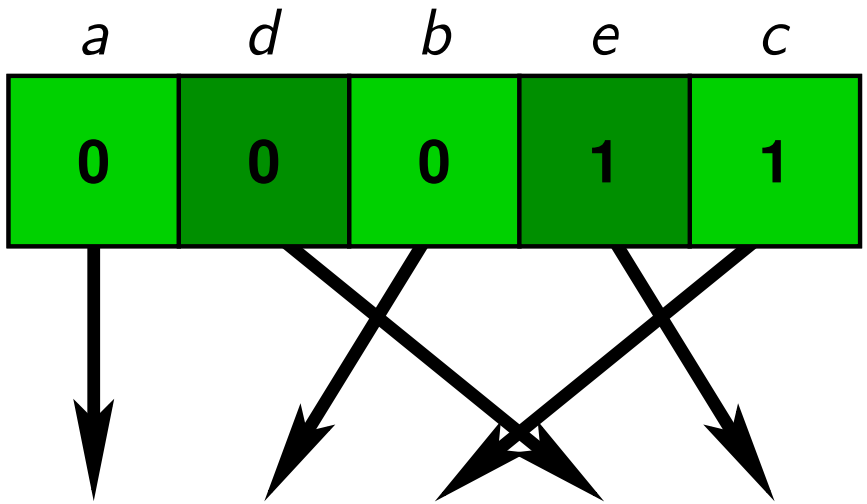




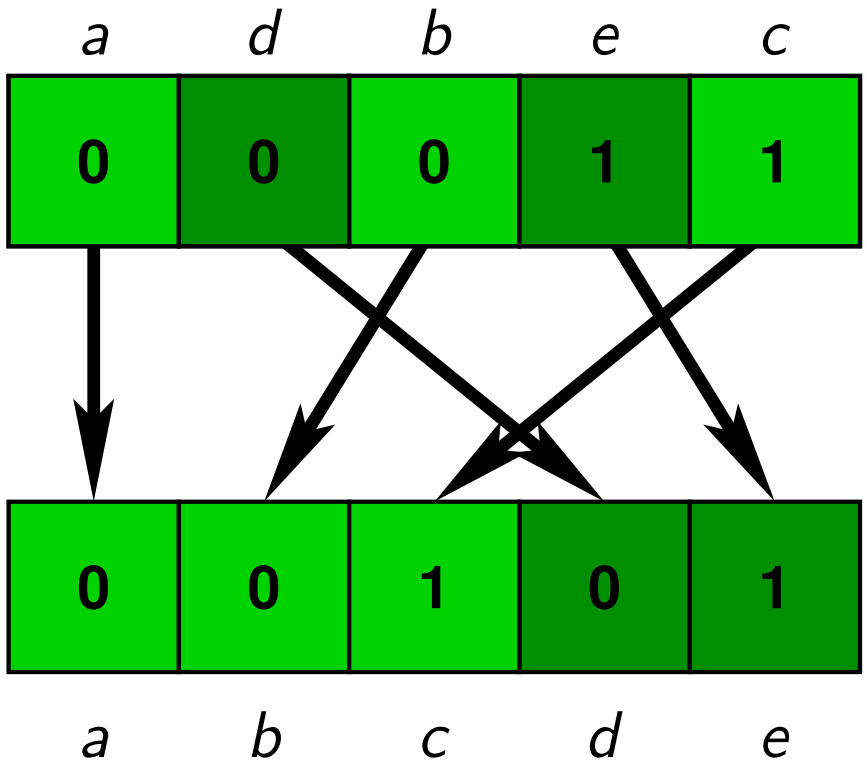
Locality preserved



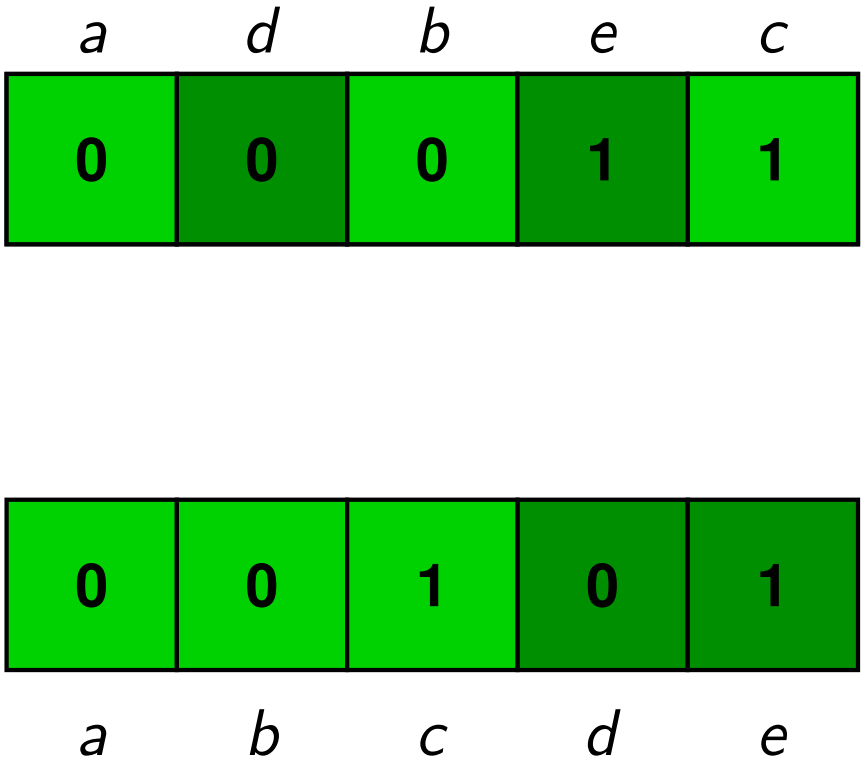
Locality preserved



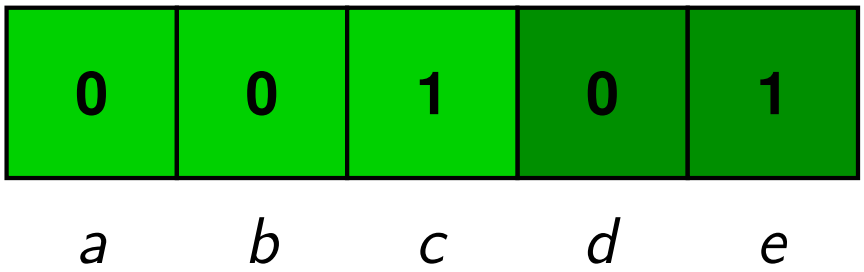
Locality preserved



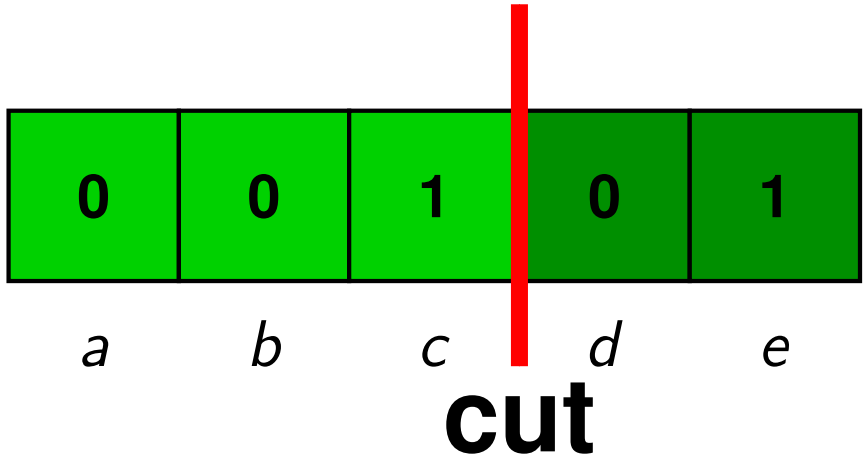
Locality preserved



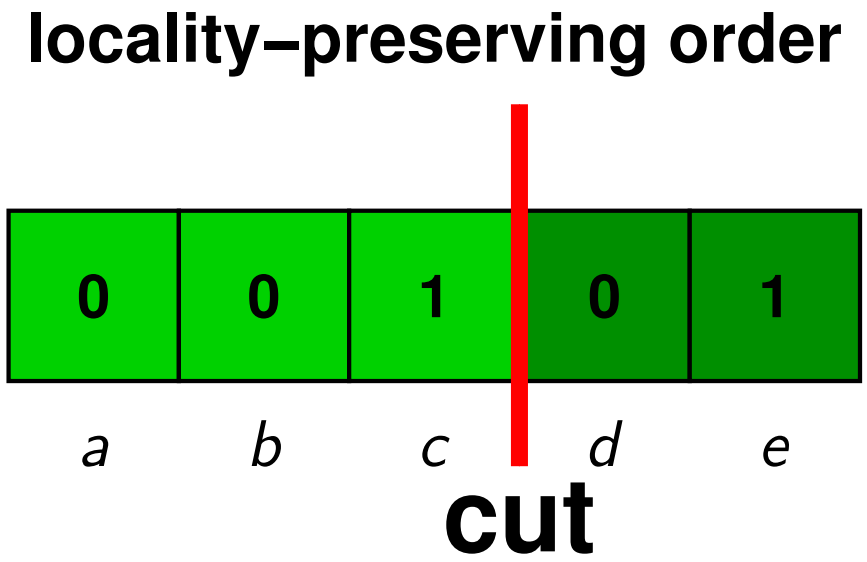
Locality preserved



Locality preserved

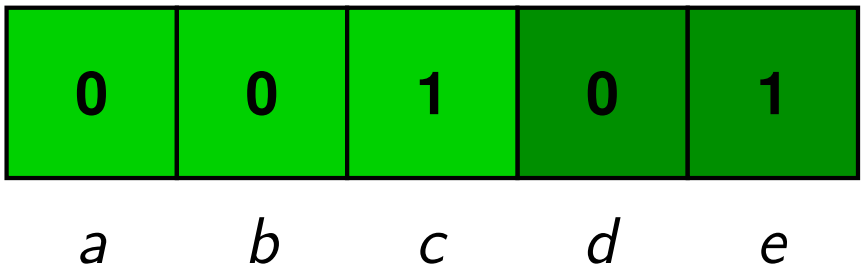


Locality preserved



Locality preserved

locality–preserving order
crossover doesn't disrupt sub–solutions



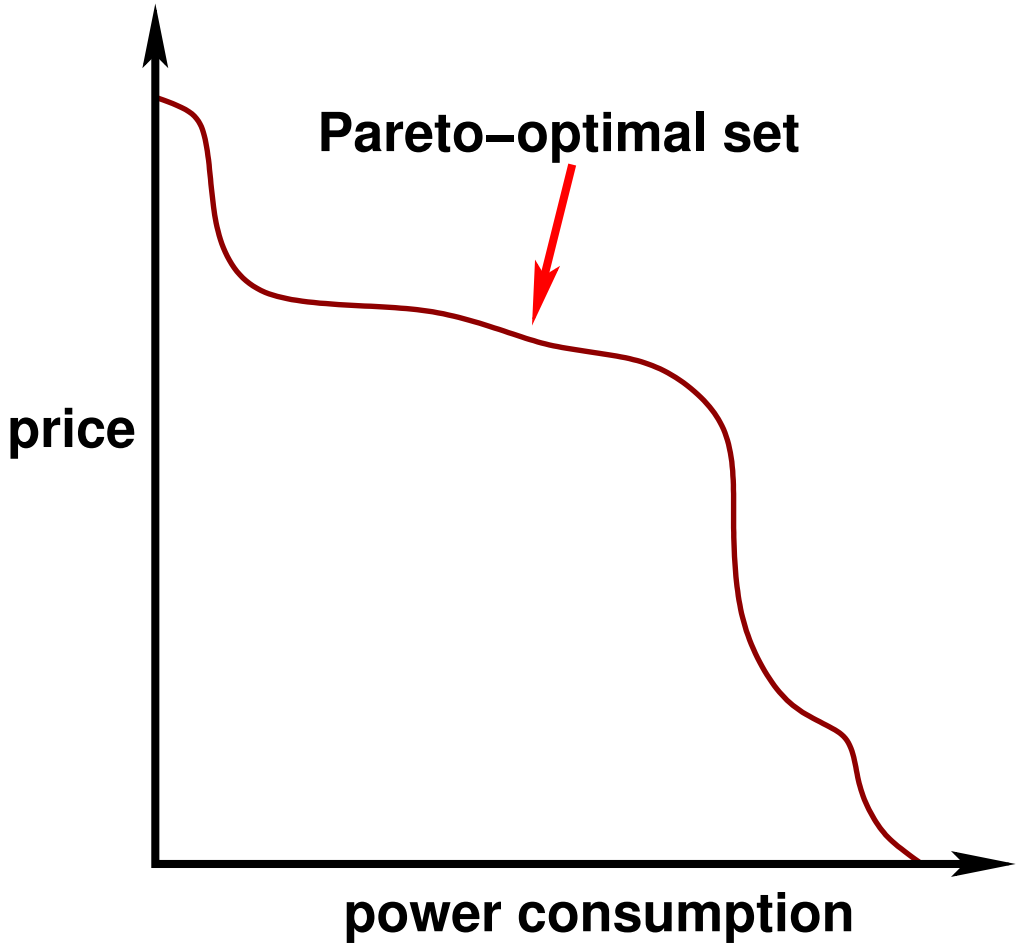
Multidimensional optimization

Real-world problems often have multiple costs.

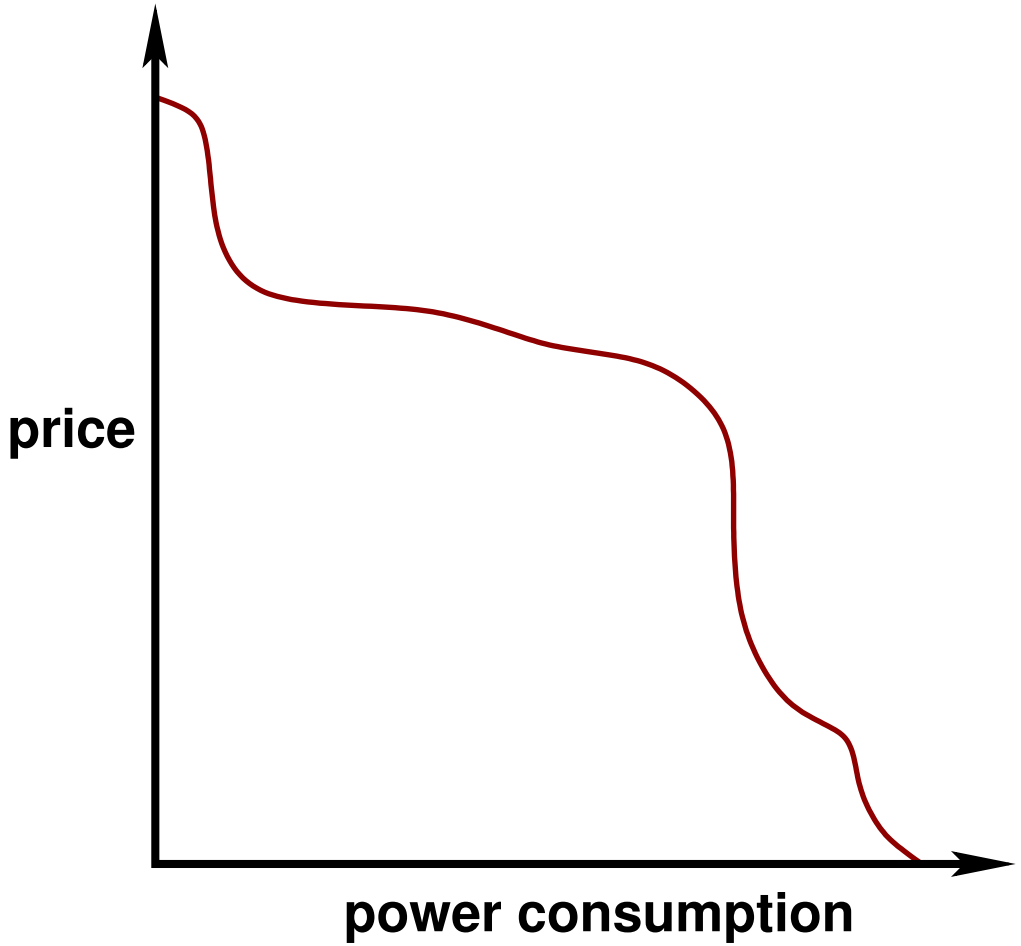
- Price.
- Power consumption.
- Speed.
- Temperature.
- Reliability.
- etc.

Necessary to simultaneously minimize all costs.

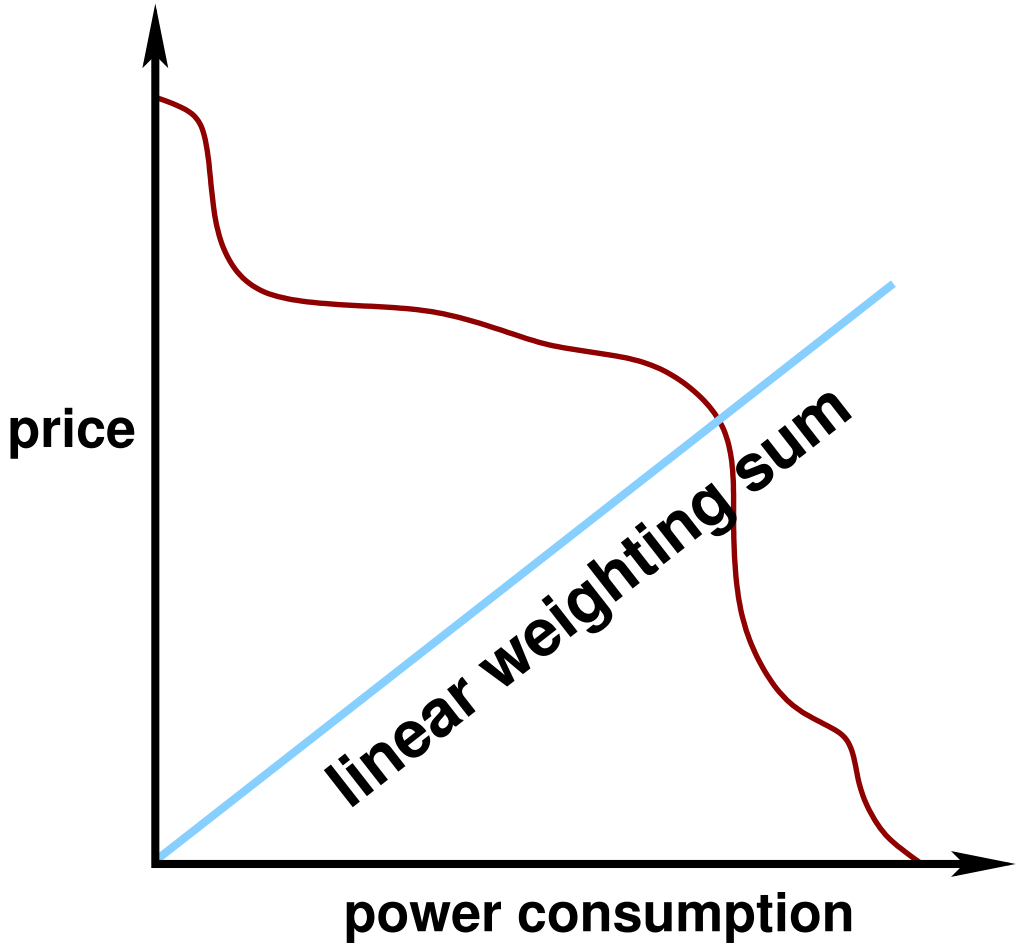
Linear weighting sum



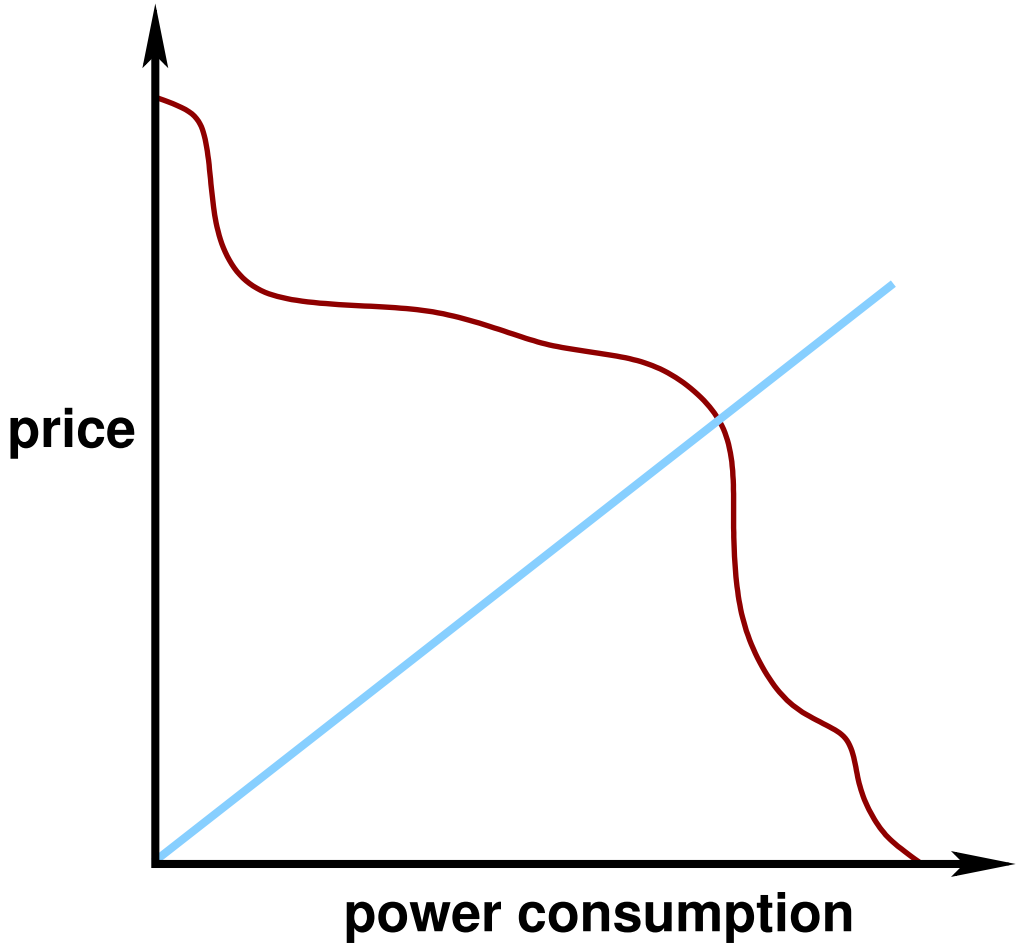
Linear weighting sum



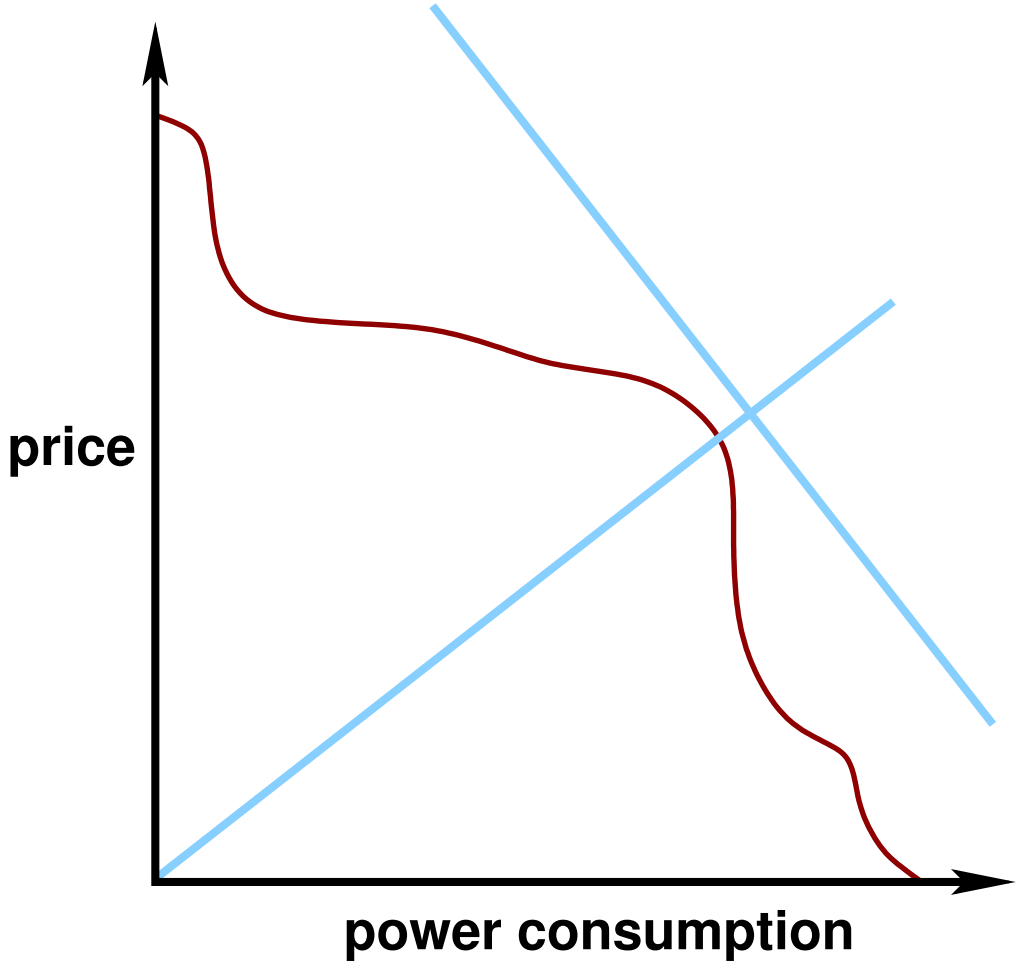
Linear weighting sum



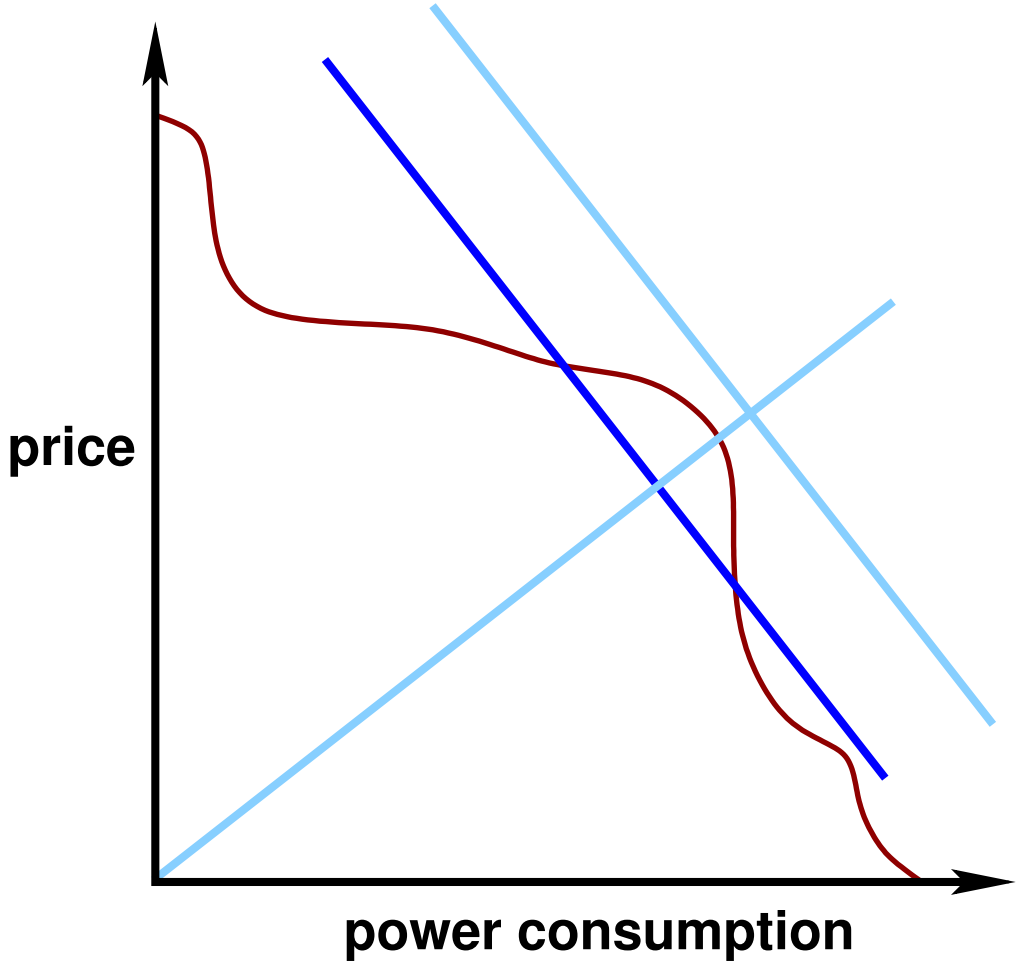
Linear weighting sum



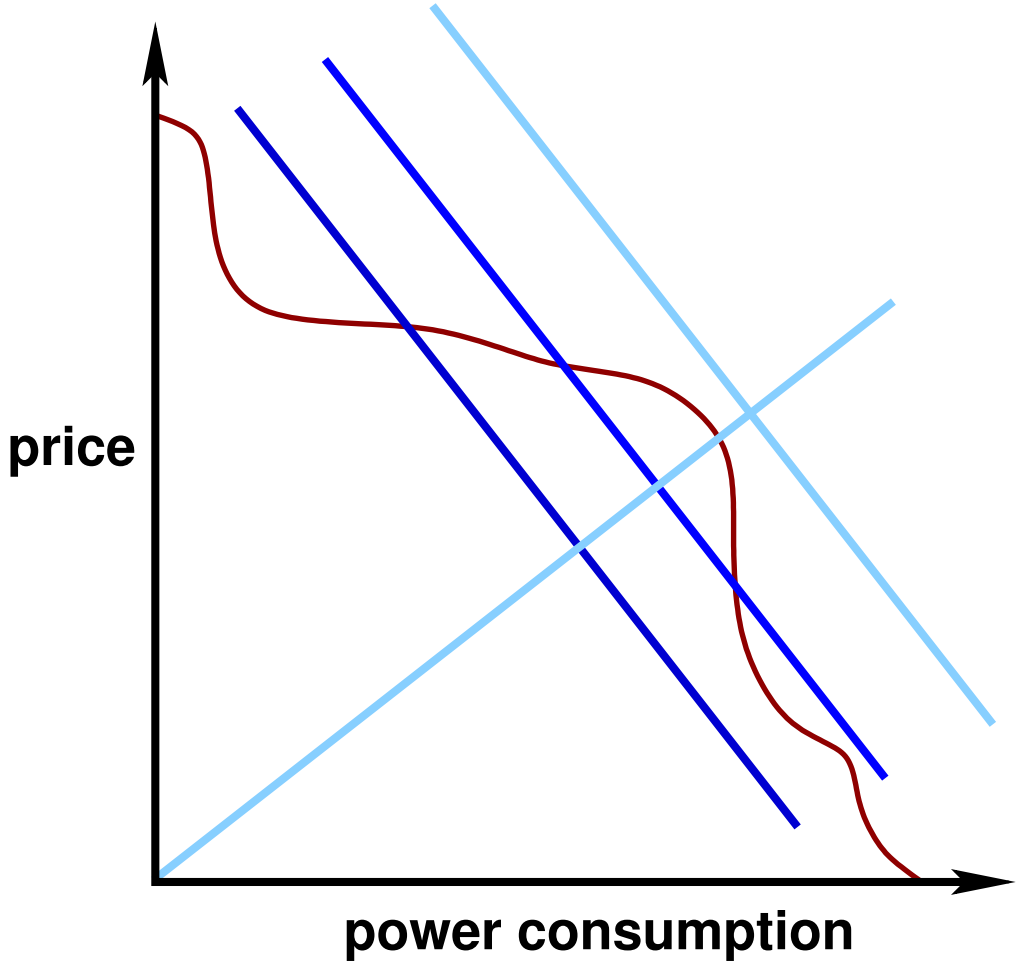
Linear weighting sum



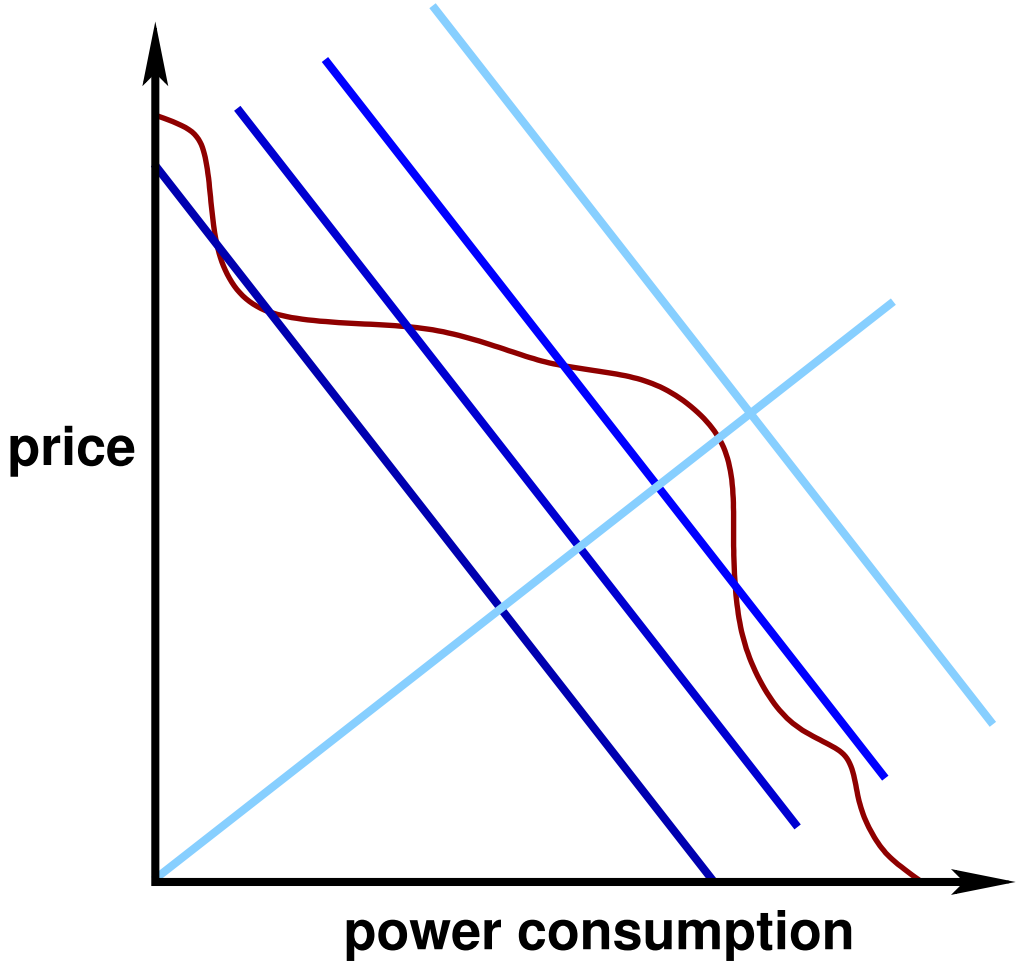
Linear weighting sum



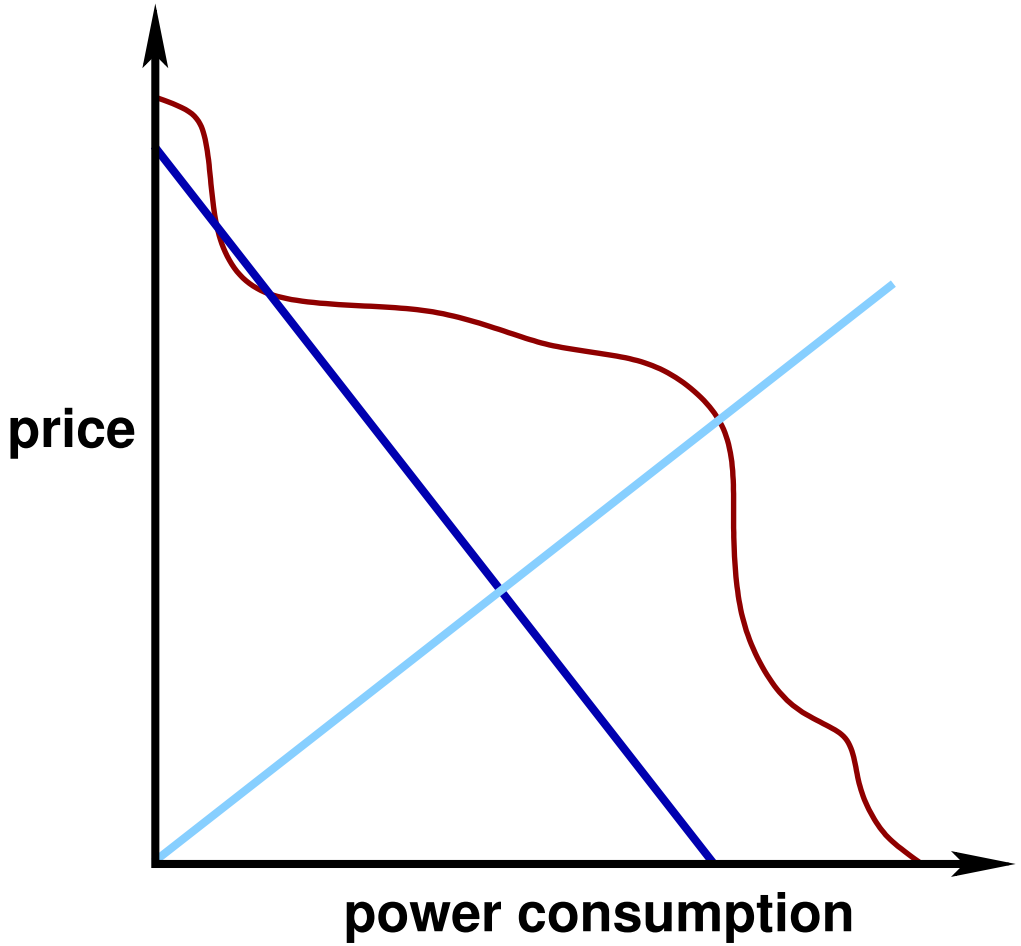
Linear weighting sum



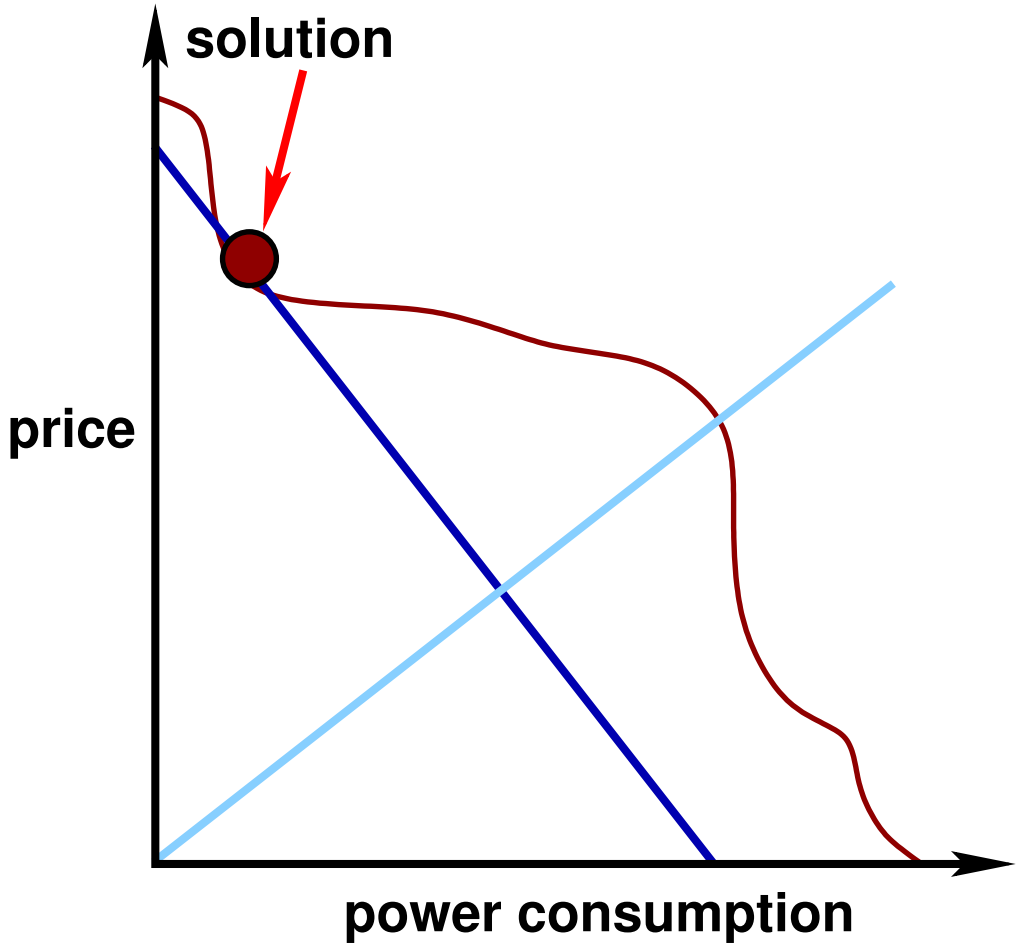
Linear weighting sum



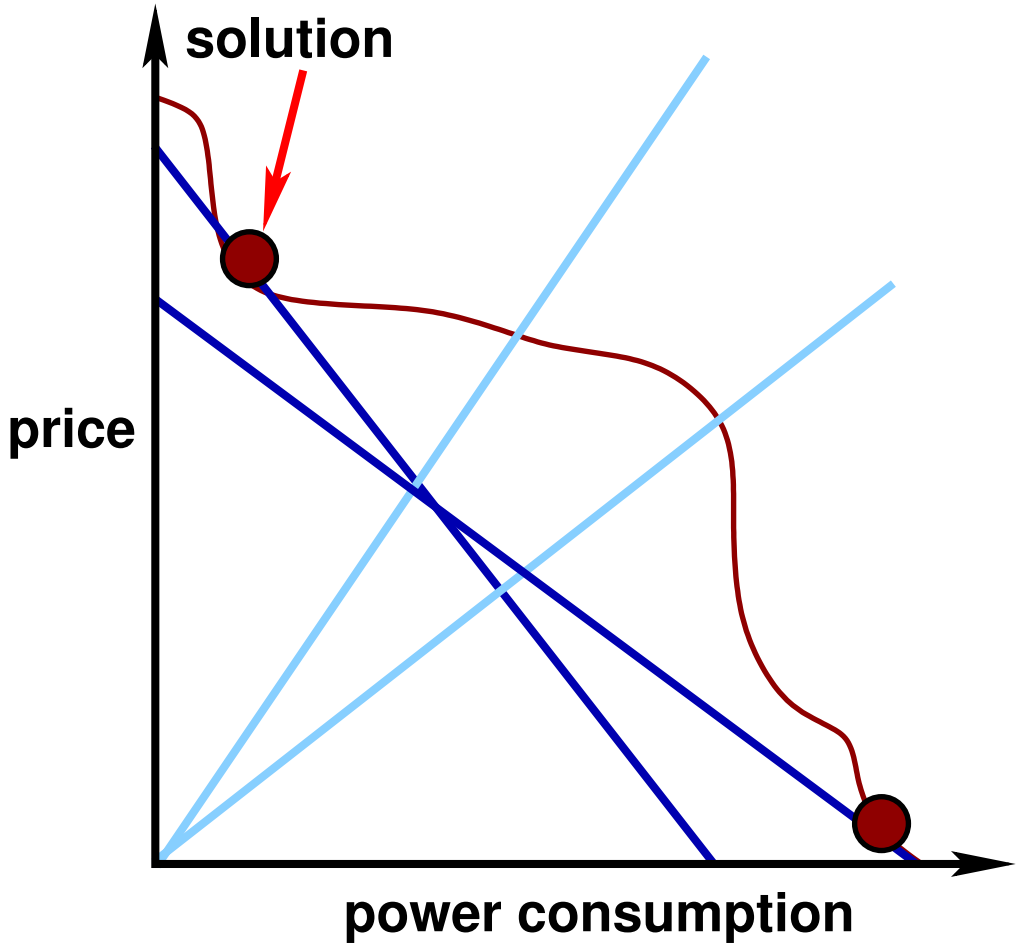
Linear weighting sum



Linear weighting sum



Linear weighting sum



Pareto-ranking

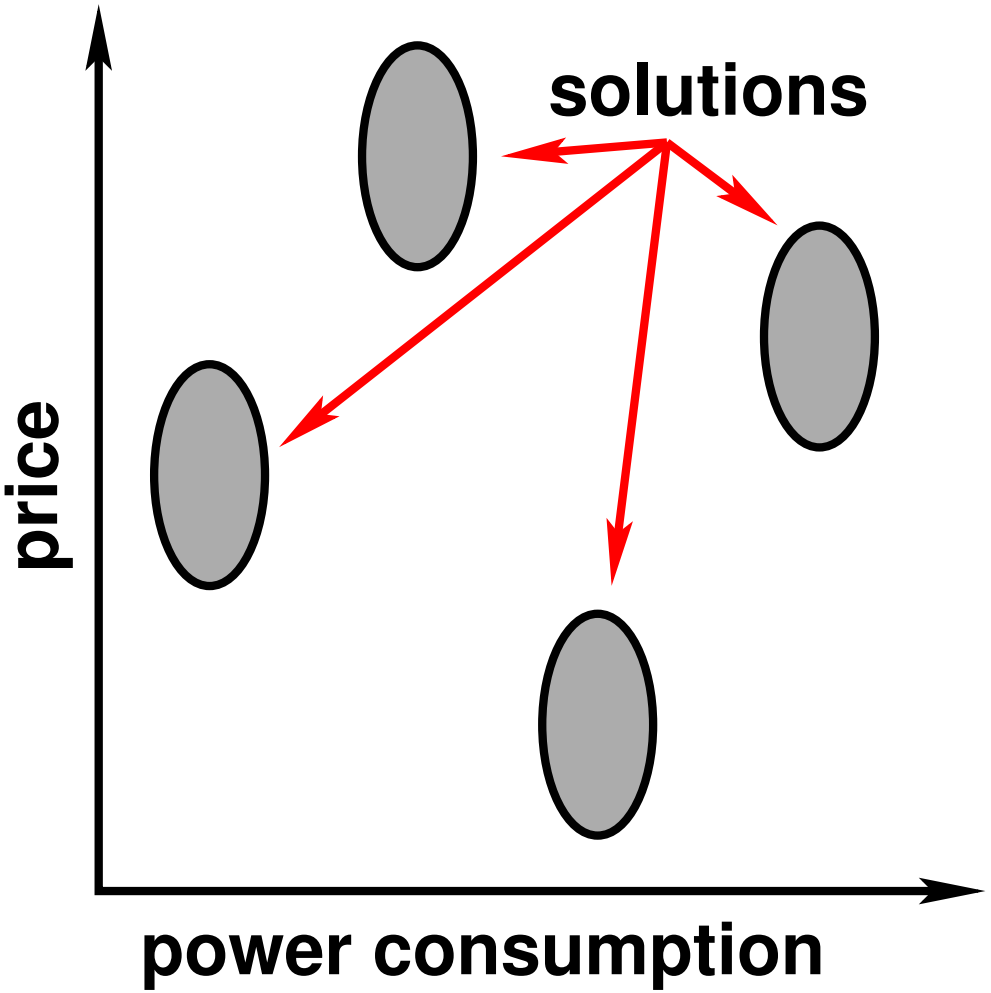
A solution dominates another if all its costs are lower, i.e.,

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

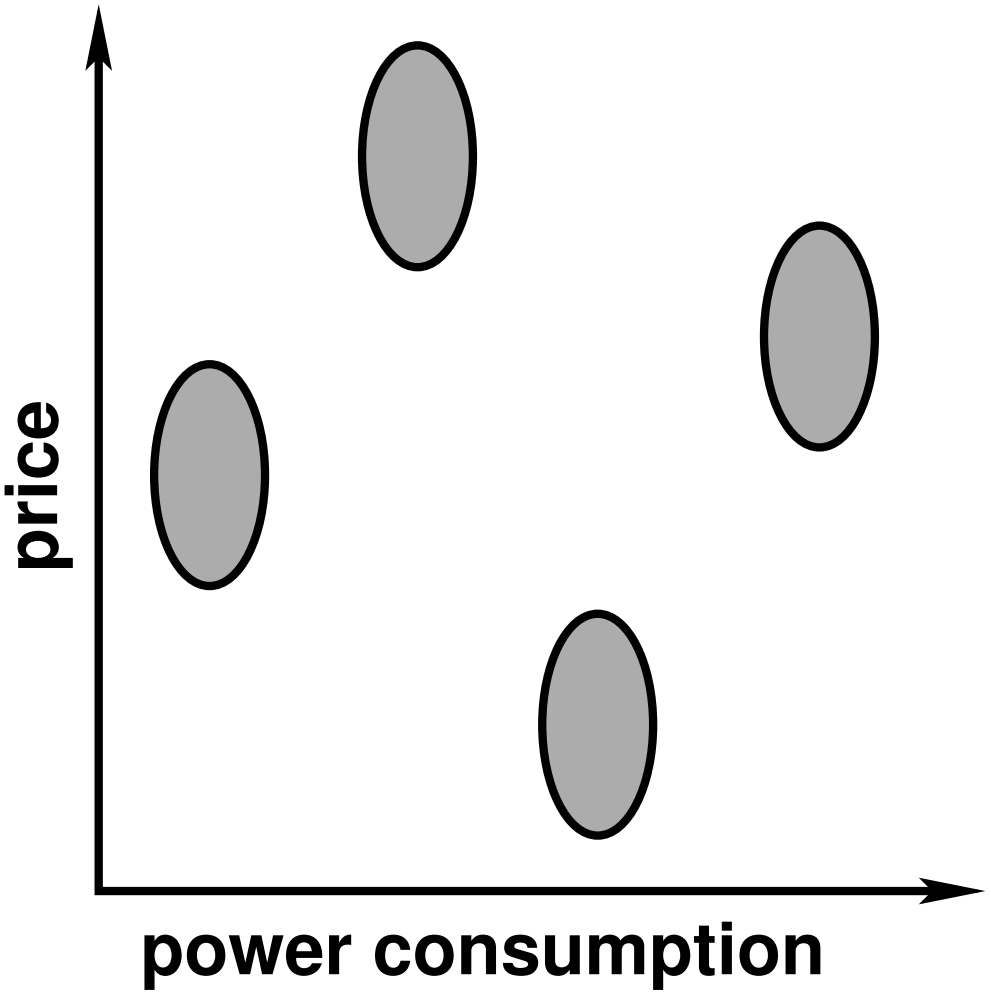
A solution's rank is the number of other solutions which do not dominate it, i.e.,

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

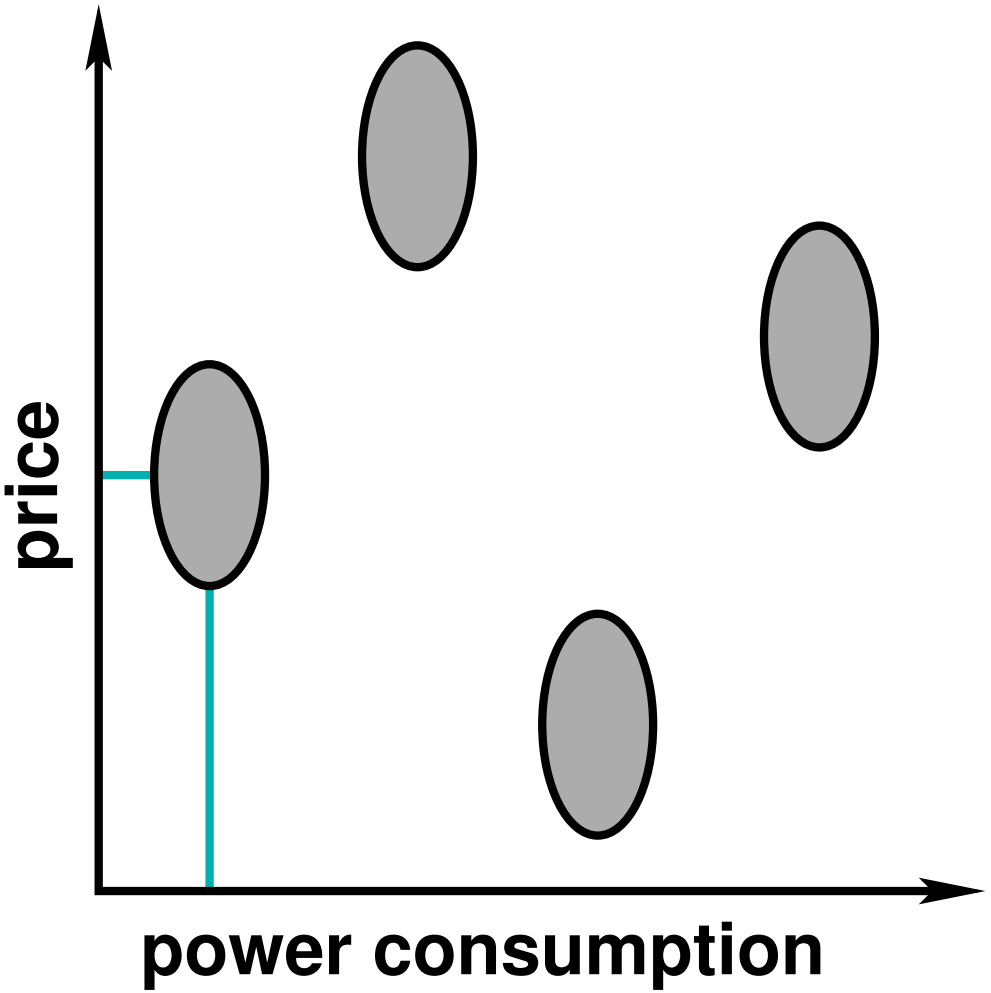
Pareto-ranking



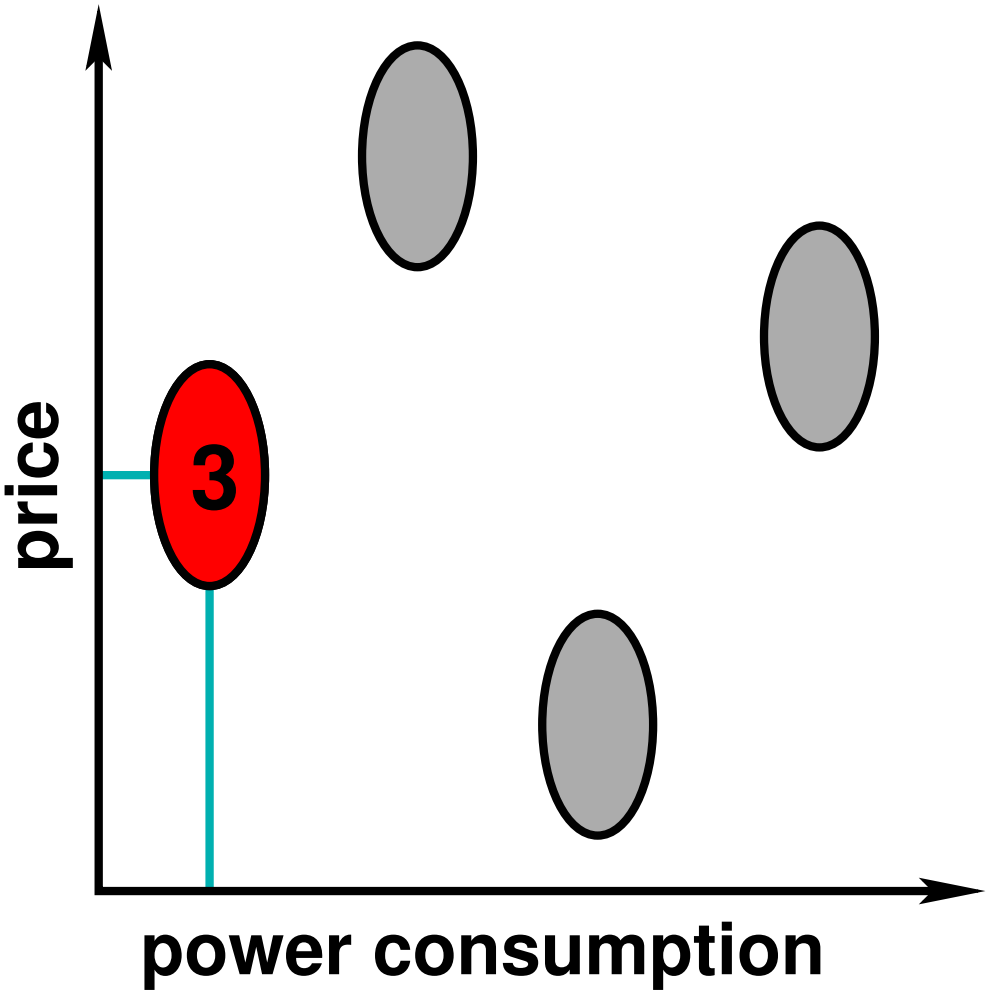
Pareto-ranking



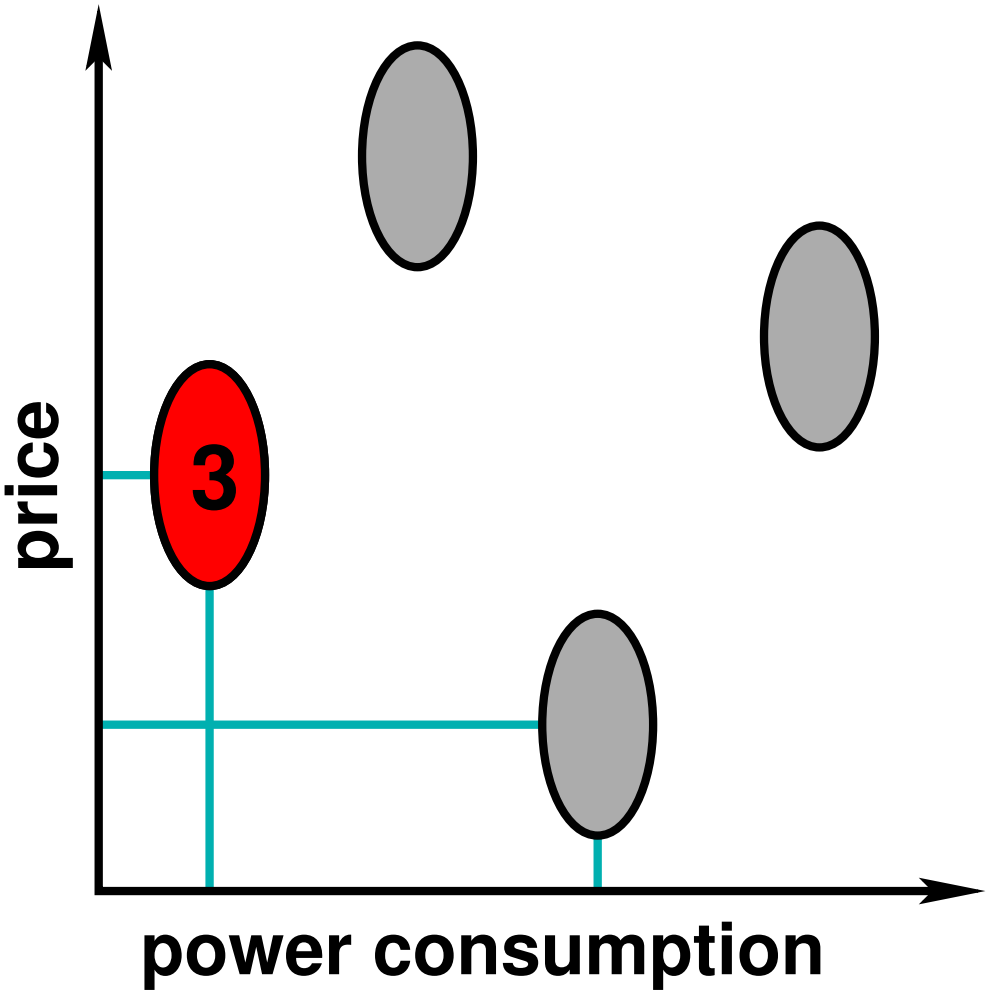
Pareto-ranking



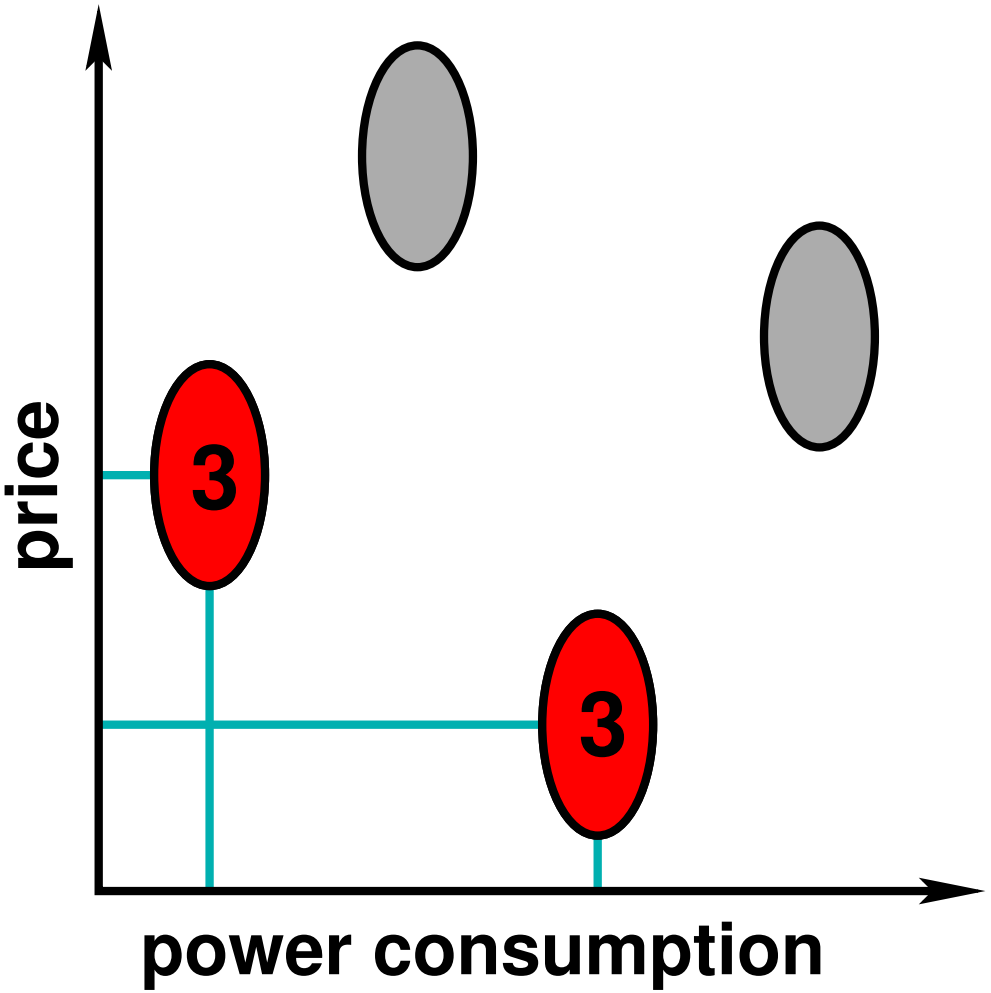
Pareto-ranking



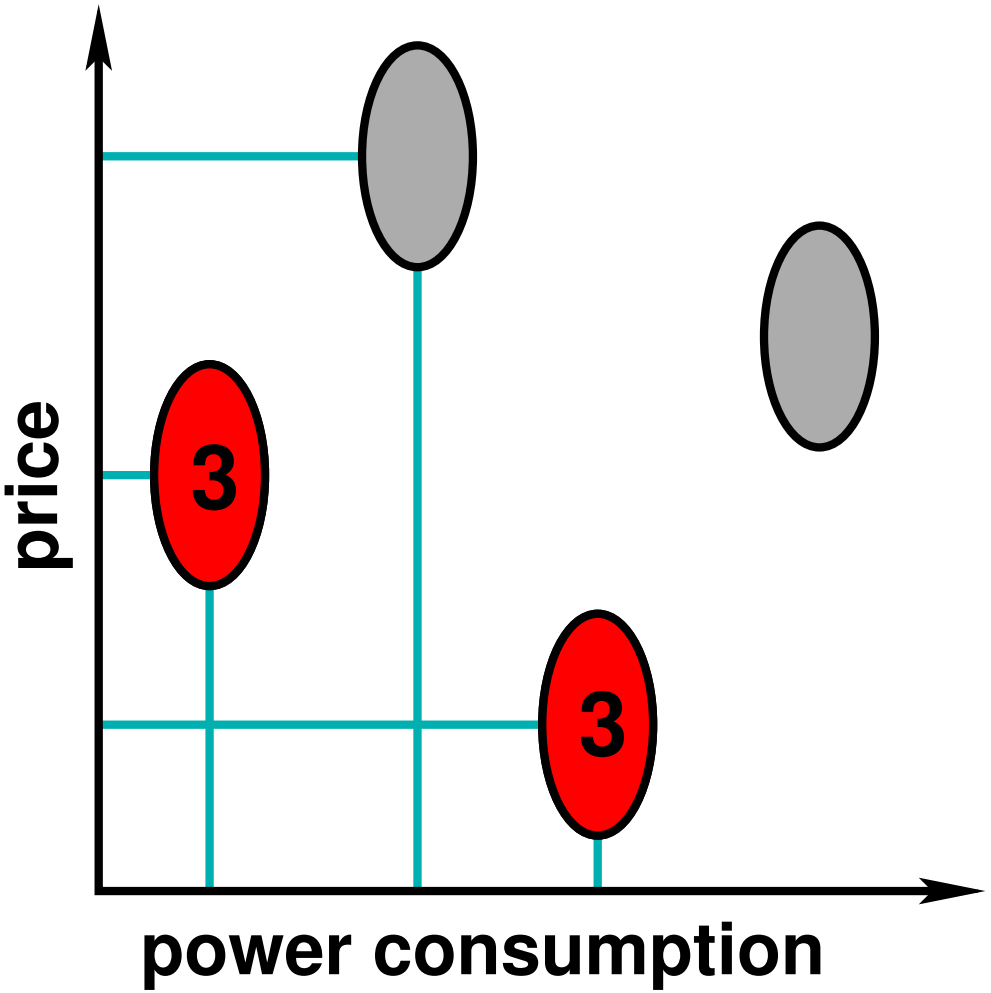
Pareto-ranking



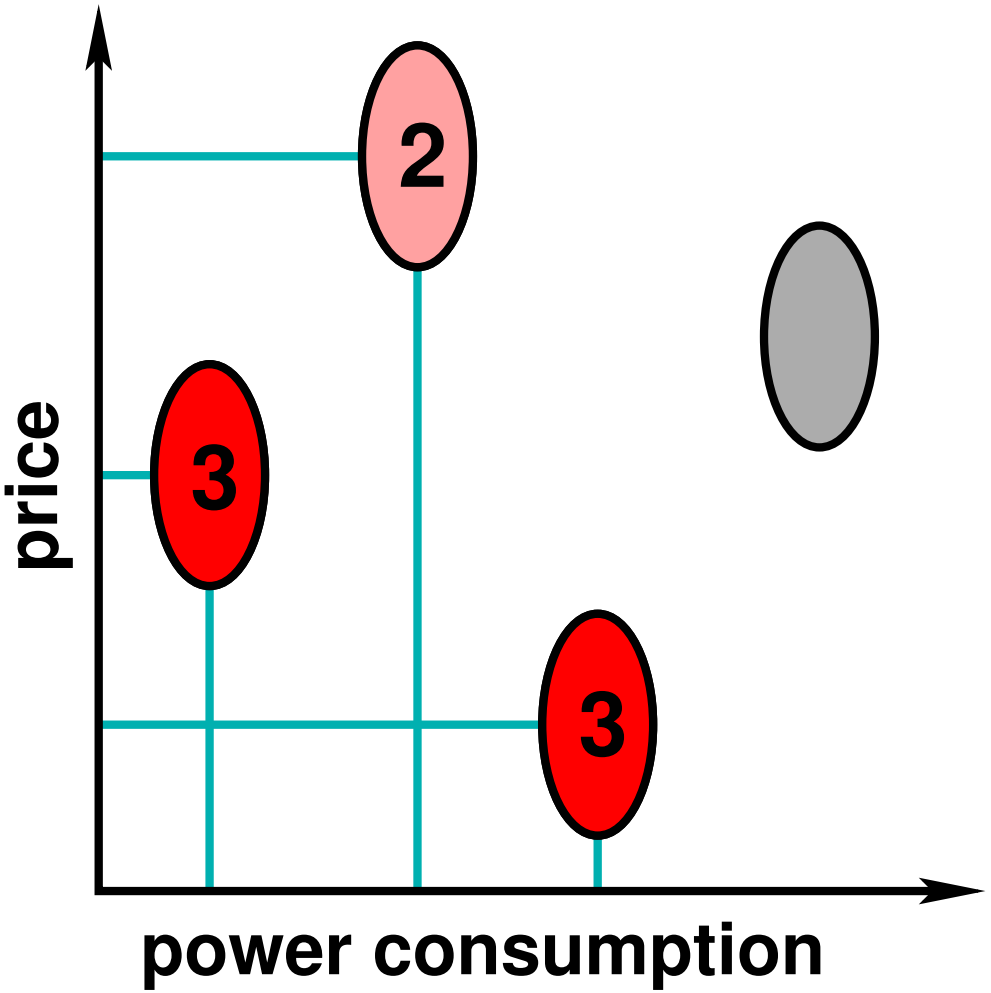
Pareto-ranking



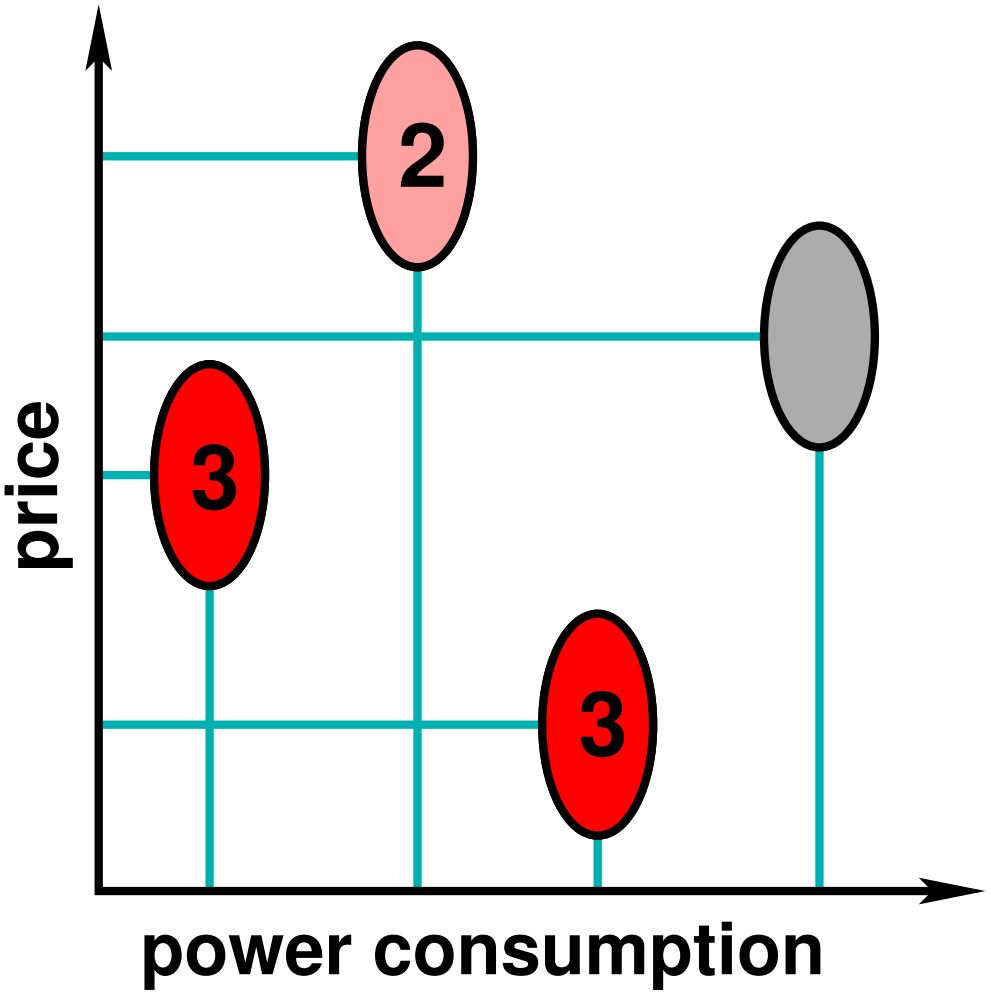
Pareto-ranking



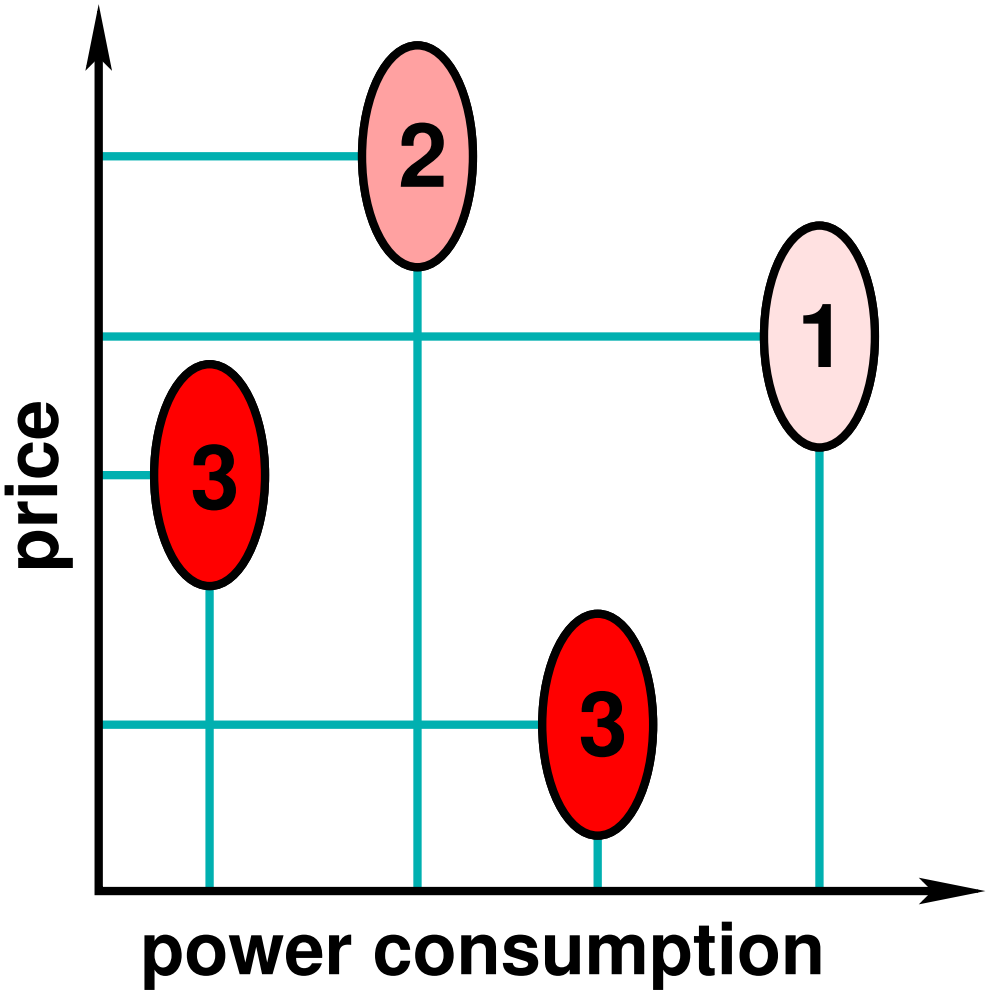
Pareto-ranking



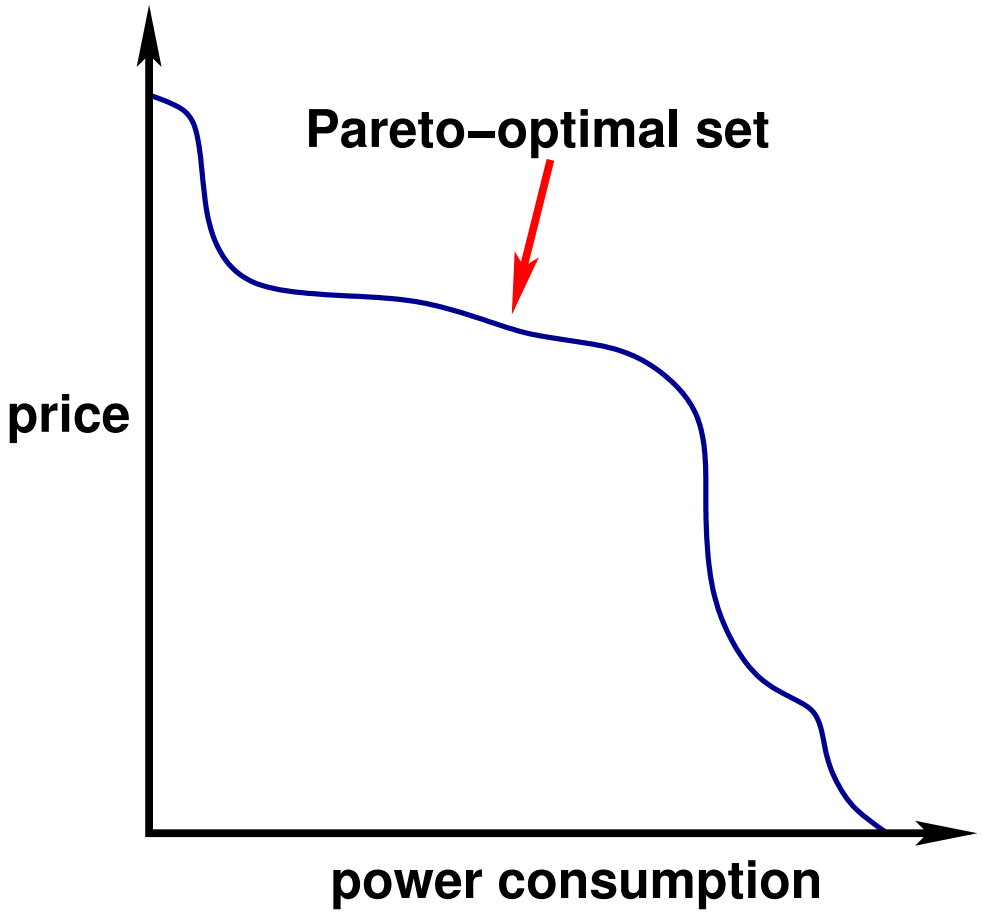
Pareto-ranking



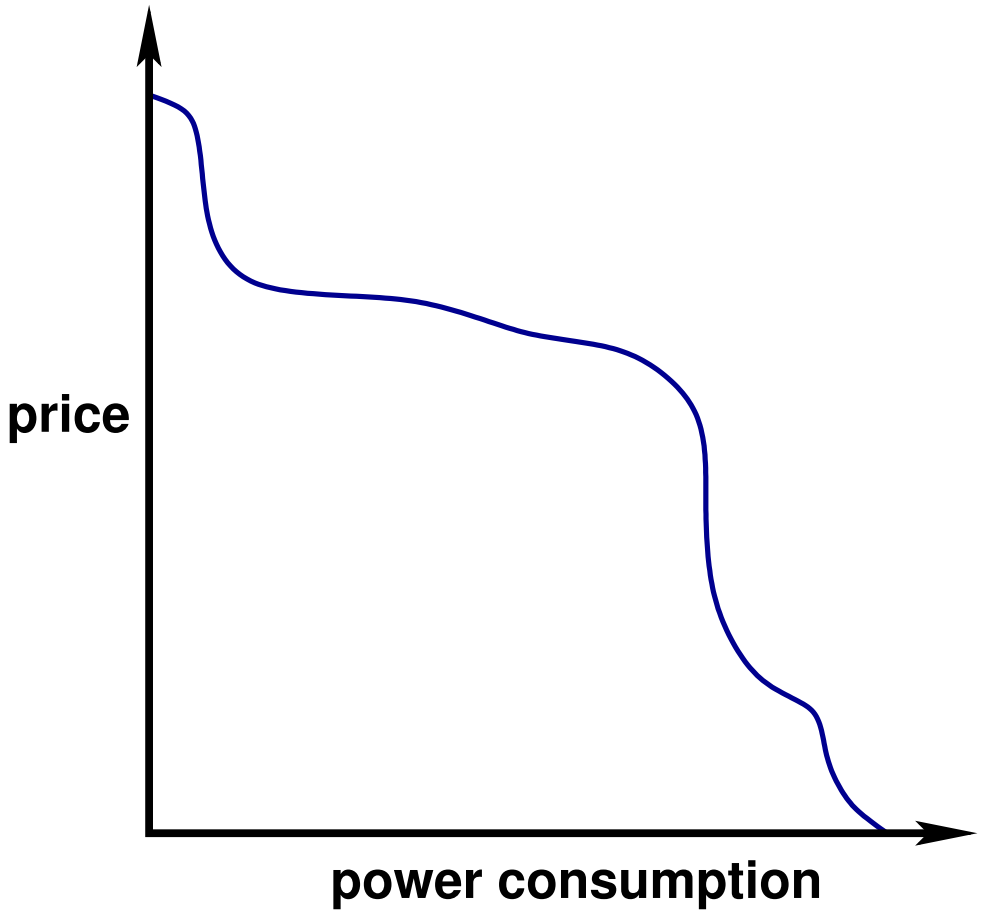
Pareto-ranking



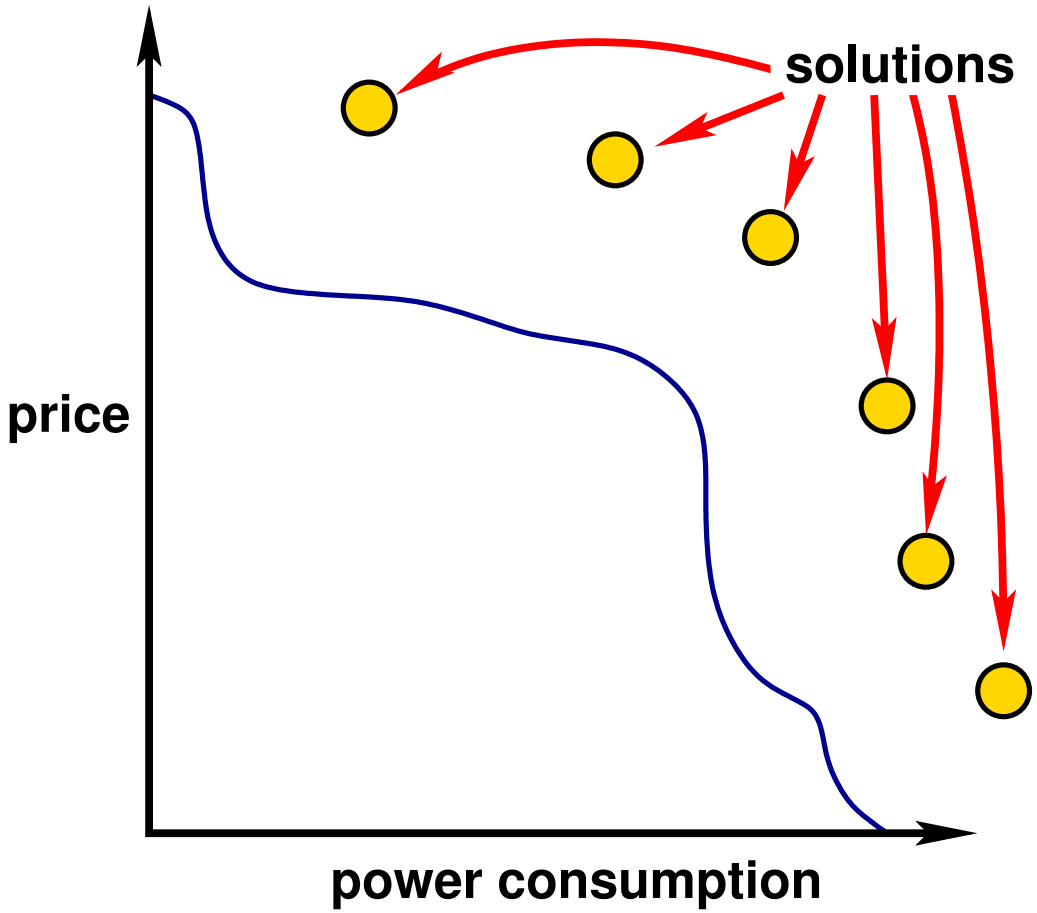
Pareto-rank based multiobjective optimization



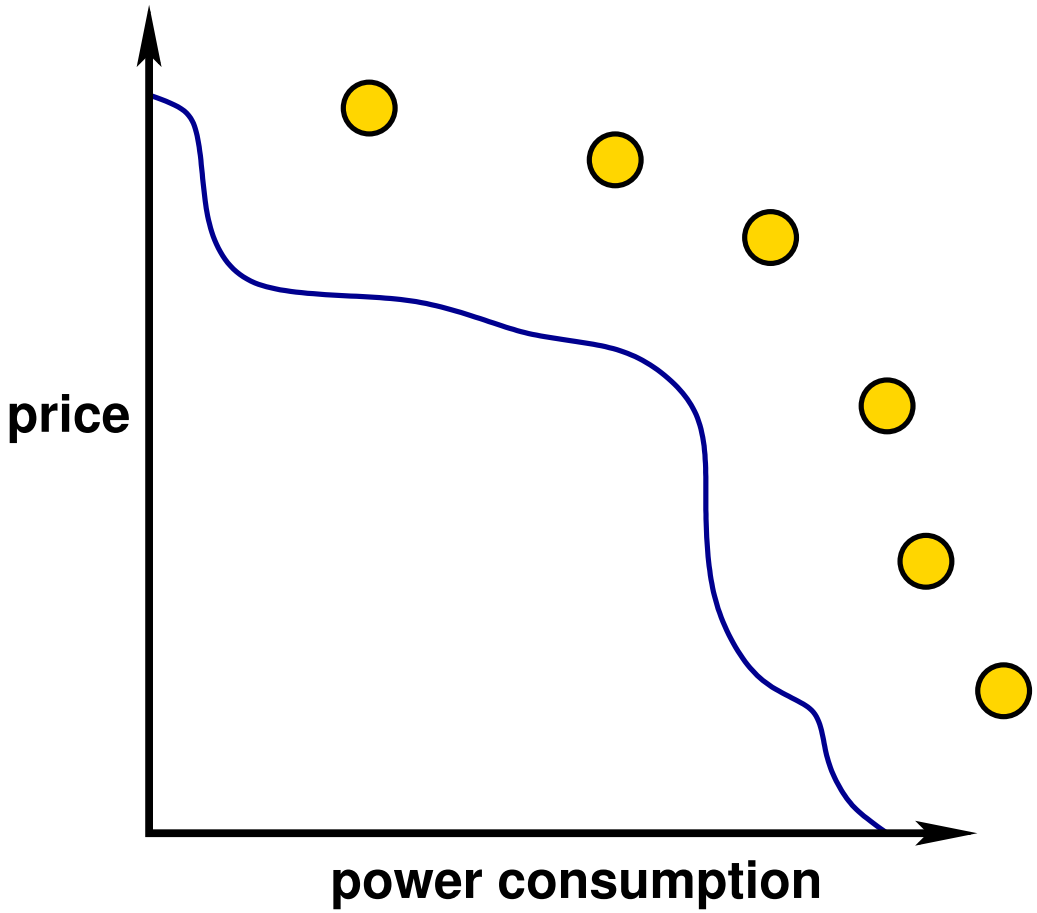
Pareto-rank based multiobjective optimization



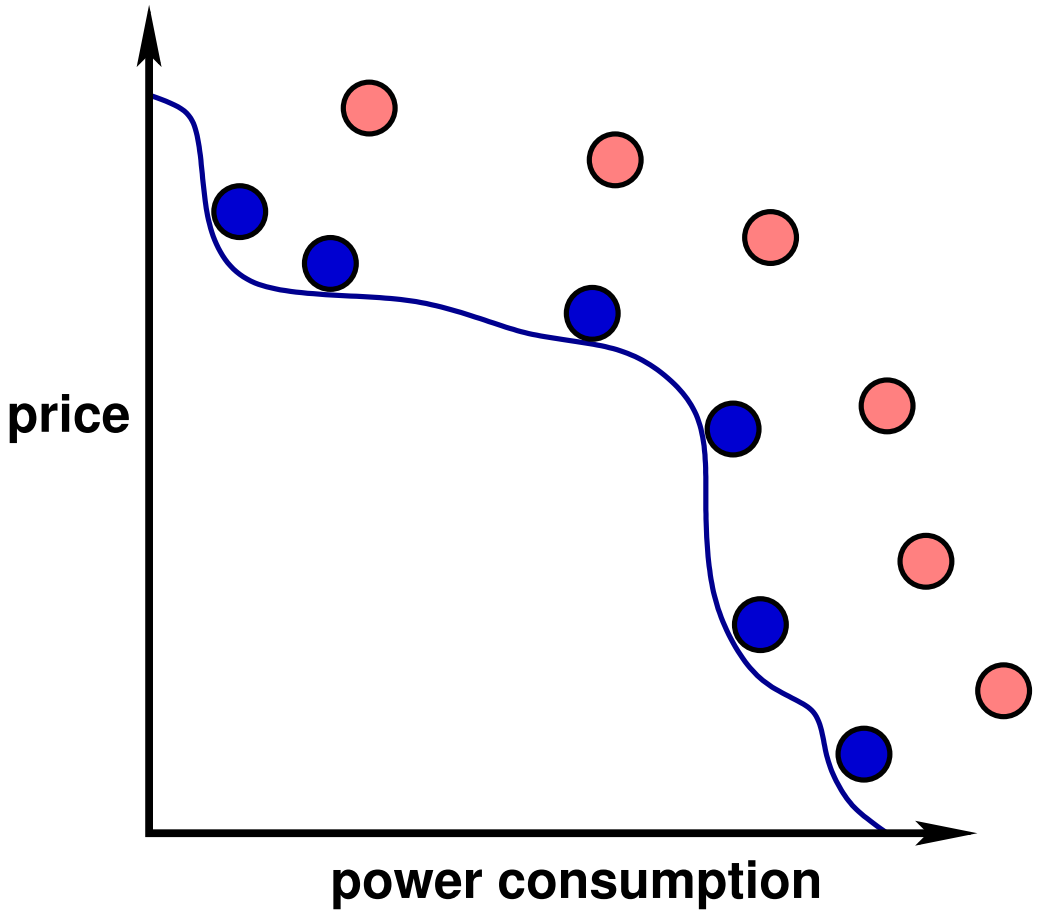
Pareto-rank based multiobjective optimization



Pareto-rank based multiobjective optimization



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.

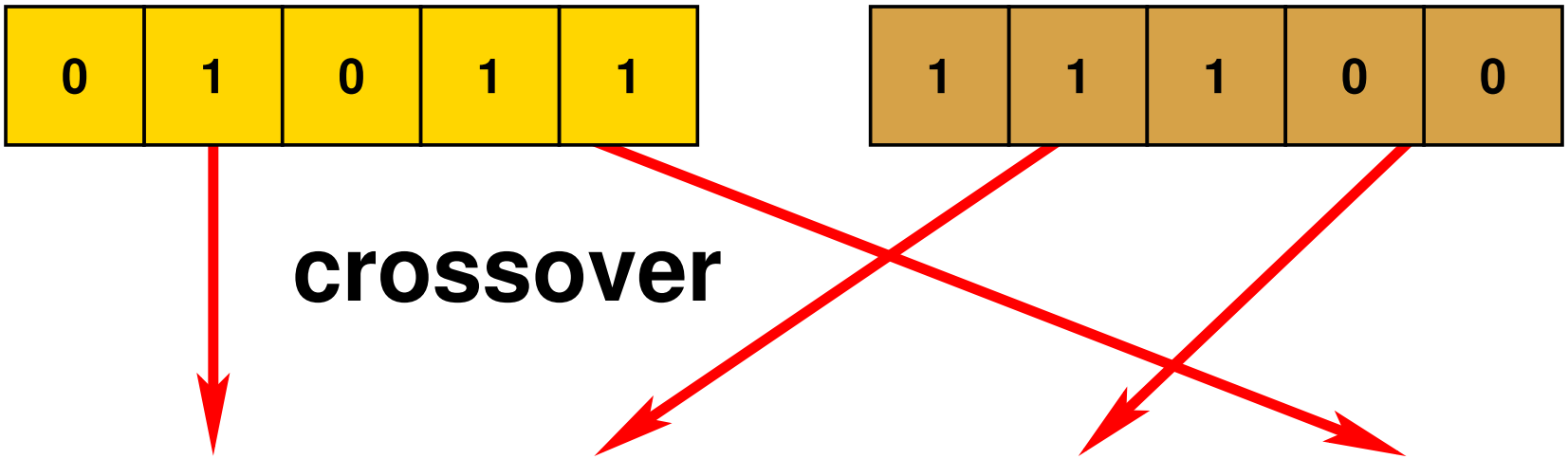
Simulated annealing if only one solution.

Easily parallizable.

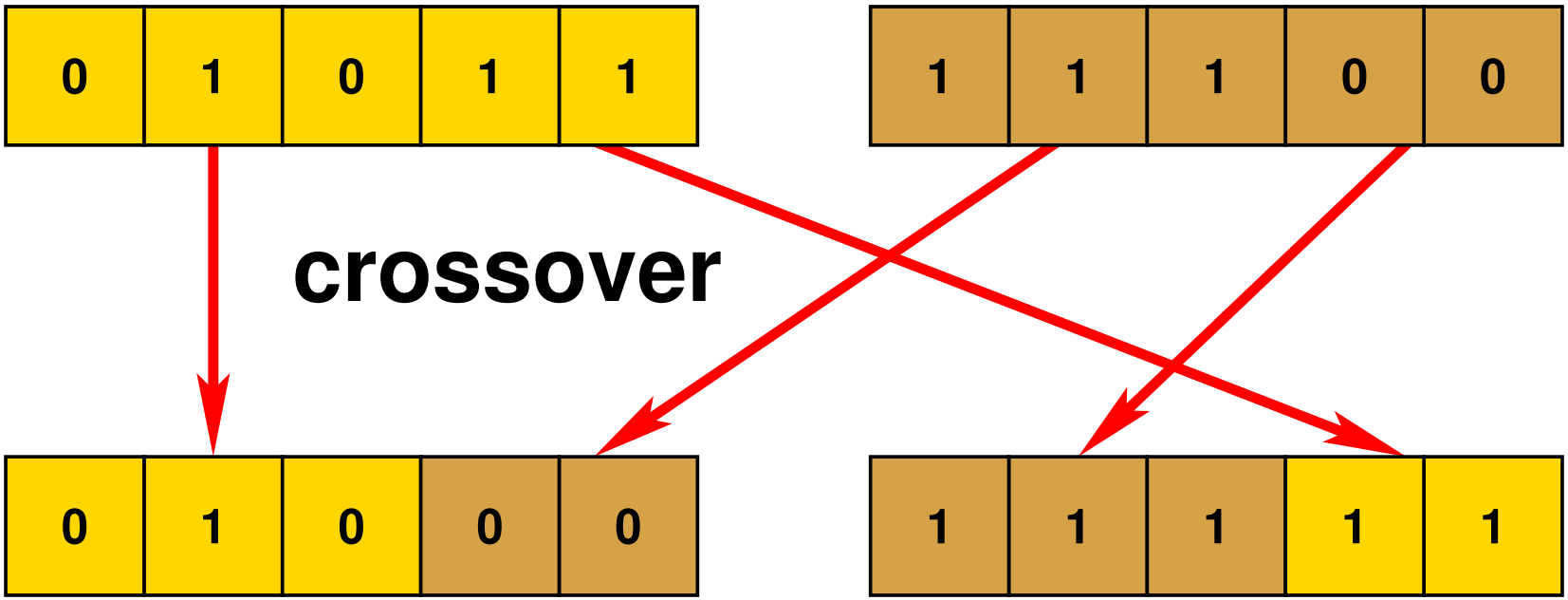
Has strengths of genetic algorithms and simulated annealing.

Difficult to implement but not more difficult than genetic algorithms.

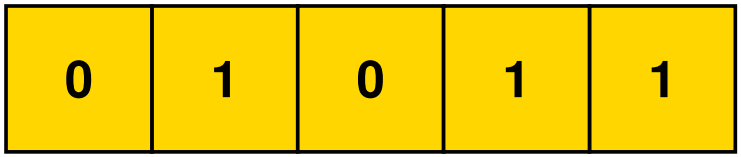
PRSA example

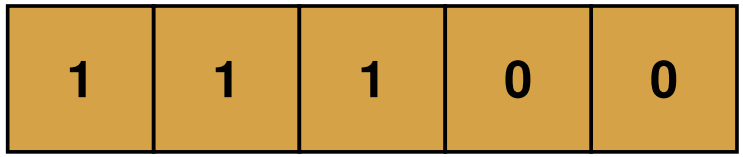


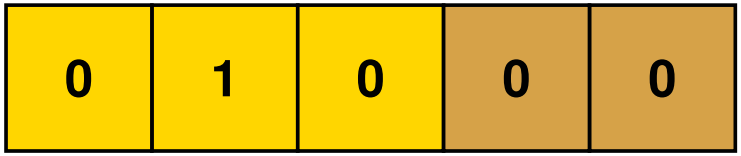
PRSA example

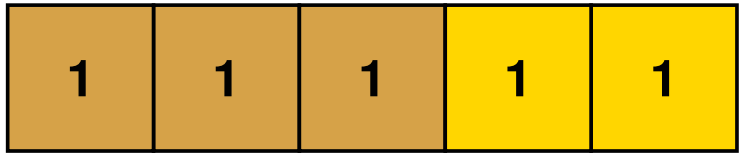


PRSA example

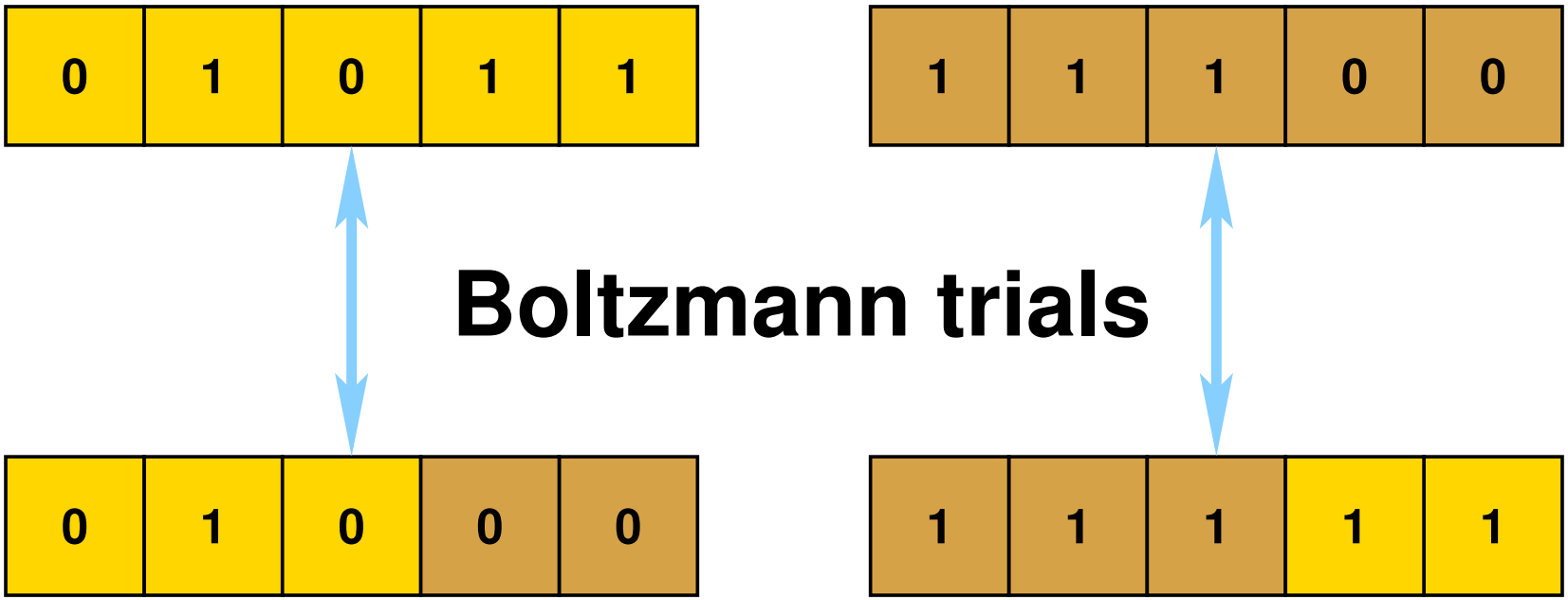






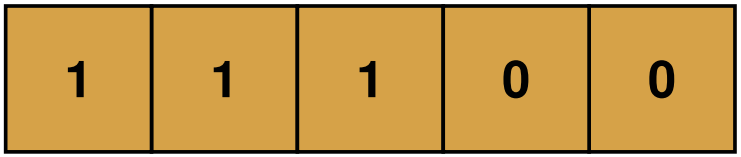


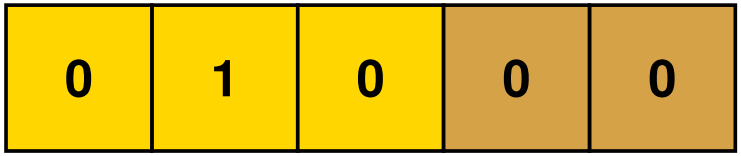
PRSA example



PRSA example

PRSA example





Multiobjective GAs

C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization," in *Proc. Int. Conf. Genetic Algorithms*, July 1993, pp. 416–423.

Explains importance of multiobjective optimization.

Shows simple way to use Pareto-rank in parallel optimization.

Meta-heuristics.

Very high-level optimization reference

R. P. Dick, "Multiobjective synthesis of low-power real-time distributed embedded systems," Ph.D. dissertation, 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*, Aug. 2002.

Interesting short article on a physical application on evolutionary algorithms.

Similar results for FPGA-based filter.

Genetic algorithms reference

D. 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

S. W. Mahfoud and D. E. Goldberg, "Parallel recombinative simulated annealing: A genetic algorithm," *Parallel Computing*, vol. 21, pp. 1–28, Jan. 1995.

Section outline

Definitions

Let Γ_{j_1,j_2} represents the dependency between tasks j_1 and j_2 where

$$\Gamma_{j_1,j_2} = \begin{cases} 1 & \text{if task } j_1 \text{ is an immediate predecessor of } j_2 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

$$\delta(j,m) = \begin{cases} 1 & \text{if task } j \text{ is assigned to core } m \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Credit to Tam Chantem and Xiaobo Sharon Hu for formulation.

Definitions

$$\alpha(j,k) = \begin{cases} 1 & \text{if task } j \text{ starts at time instant } k \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$\sigma(j,k) = \begin{cases} 1 & \text{if task } j \text{ ends at time instant } k \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Constraints

- Every task j is assigned to exactly one core m:

$$\forall j \in J \qquad \sum_{m \in M} \delta(j, m) = 1 \tag{5}$$

- Every task j meets its deadline:

$$\forall j \in J \qquad ts(j) + te(j) \leq d(j) \tag{6}$$

- Precedence constraints are honored:

$$\forall j \in J \qquad ts(j_2) \geq tf(j_1) \cdot \Gamma_{j_1, j_2} \tag{7}$$

$$\forall k, \forall j_1, j_2 \in J \qquad \sum_{k'=0}^{k} (\sigma(j_1, k') - \alpha(j_2, k')) \cdot \Gamma_{j_1, j_2} \geq 0 \tag{8}$$

Constraints

- Every task has only one start time instant:

$$\forall j \in J \qquad \sum_{k=0}^{2|J|-1} \alpha(j,k) = 1 \tag{9}$$

- Every task has only one finish time instant:

$$\forall j \in J \qquad \sum_{k=0}^{2|J|-1} \sigma(j,k) = 1 \tag{10}$$

- The start time and the finish time instants of a task must be different:

$$\forall j \in J, \forall k \qquad \alpha(j,k) + \sigma(j,k) \leq 1 \tag{11}$$

Constraints

- At each time instant, at most one task can be active on a core:

$$\forall k, \forall m \in M \qquad \sum_{j \in J} \beta(k, j, m) \leq 1 \tag{12}$$

- A task j must start before it ends:

$$\forall j \in J, \forall k \qquad \sum_{k'=0}^{k} \alpha(j, k') \geq \sum_{k'=0}^{k} \sigma(j, k') \tag{13}$$

Constraints

If tasks j_1 and j_2 both execute on core m, they must not overlap:

$$\forall j_1, j_2 \in J : j_1 \neq j_2, \forall m \in M, \forall k$$

$$tf(j_1) \leq (2 - \delta(j_1, m) - \delta(j_2, m)) \cdot \Lambda + ts(j_2) + \left(1 - \sum_{k'=0}^{k} (\sigma(j_1, k') - \alpha(j_2, k'))\right) \cdot \Lambda \quad (14)$$

$$tf(j_2) \leq (2 - \delta(j_1, m) - \delta(j_2, m)) \cdot \Lambda + ts(j_1) + \left(1 - \sum_{k'=0}^{k} (\alpha(j_2, k') - \sigma(j_1, k'))\right) \cdot \Lambda \quad (15)$$

Outline

Assignments I

13 Sep.: L. Yang, R. P. Dick, H. Lekatsas, and S. Chakradhar, "High-performance operating system controlled on-line memory compression," *ACM Trans. Embedded Computing Systems*, vol. 9, no. 4, pp. 30:1–30:28, Mar. 2010 summary.

15 Sep.: R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: A design alternative for cache on-chip memory in embedded systems," in *Proc. Int. Wkshp. Hardware/Software Co-Design*, May 2002, pp. 73–78 summary.

20 Sep.: A. Bonde, J. R. Codling, K. Naruethep, Y. Dong, W. Siripaktanakon, S. Ariyadech, A. Sangpetch, O. Sangpetch, S. Pan, H. Y. Noh, and P. Zhang, "PigNet: failure-tolerant pig activity monitoring system using structural vibration," in *Proc. Int. Symp. Information Processing in Sensor Networks*, May 2001 summary.

Assignments II

22 Sep.: C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. of the ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973 summary.

27 Sep.: Final project proposal deadline.

27 Sep.: L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, Oct. 2010, pp. 105–114.

29 Sep.: J. Polastre, R. Szewczyk, A. Mainwaring, D. Culler, and J. Anderson, "Analysis of wireless sensor networks for habitat monitoring," in *Wireless Sensor Networks*, C. S. Raghavendra, K. M. Sivalingam, and T. Znati, Eds. Springer US, 2004, ch. 18, pp. 399–423.