CSE 6140/ CX 4140:

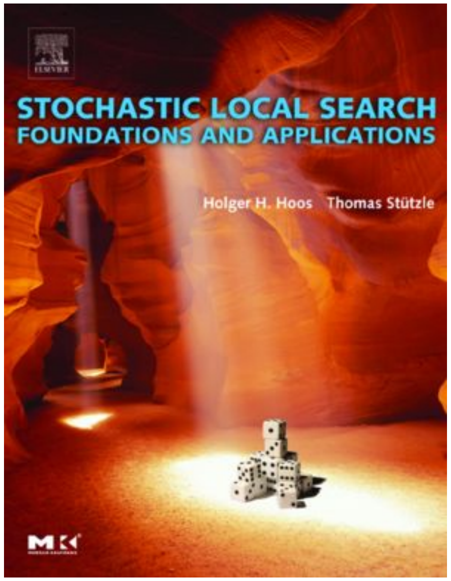
Computational Science and Engineering ALGORITHMS

Instructor: Anne Benoit

Visiting Associate Professor, CSE

Based on slides by Bistra Dilkina and Holger Hoos

HEURISTICS/LOCAL SEARCH [SLS2]



'This material is based on slides provided with the book 'Stochastic Local Search: Foundations and Applications' by Holger H. Hoos and Thomas Stützle (Morgan Kaufmann, 2004) - see www.sls-book.net for further information.'

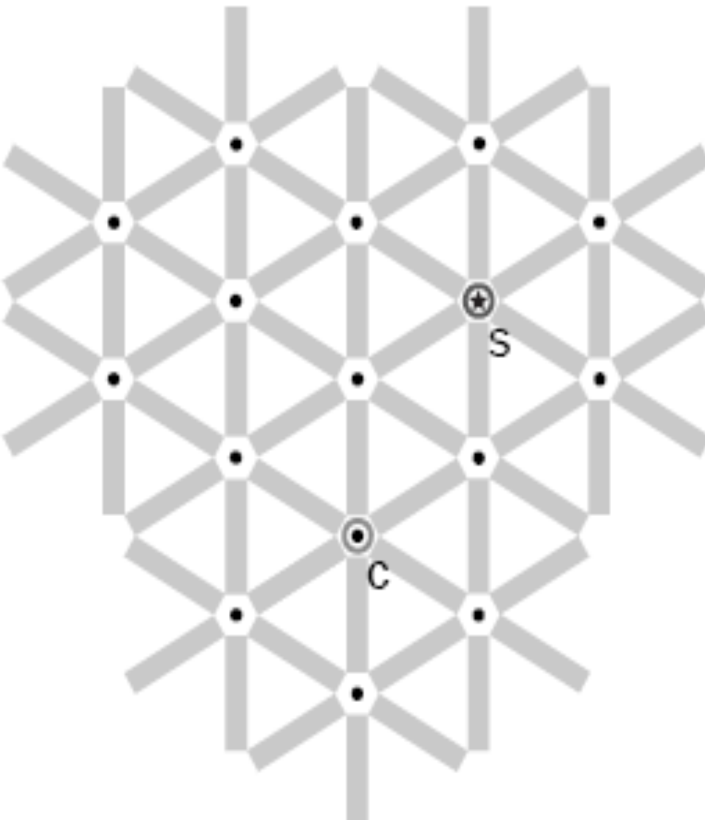
Local Search (LS) Algorithms

- search space S
  - SAT: set of all complete truth assignments to propositional variables (all "potential solutions")
- solution set $S' \subseteq S$
  - SAT: all satisfying assignments for given formula
- neighborhood relation $N \subseteq SxS$
  - A way to move from one potential solution to another
  - SAT: neighboring variable assignments differ in the truth value of exactly one variable
- evaluation function g : S → R+
  - SAT: number of clauses satisfied under given assignment

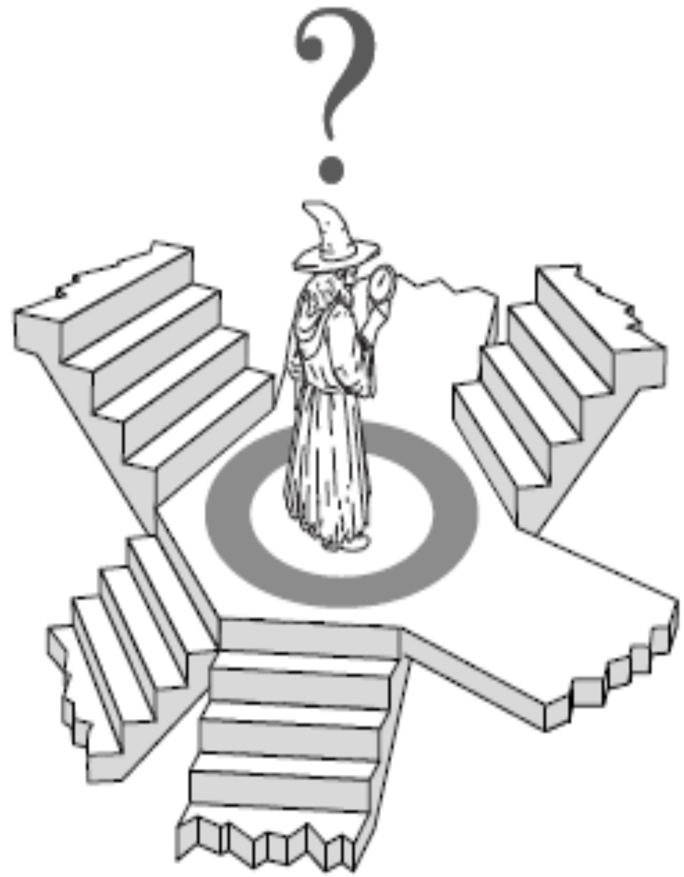
Local Search (LS)

- Start from initial position
- Iteratively move from current position to one of neighboring positions
- Use evaluation function to choose among neighboring positions

Local Search



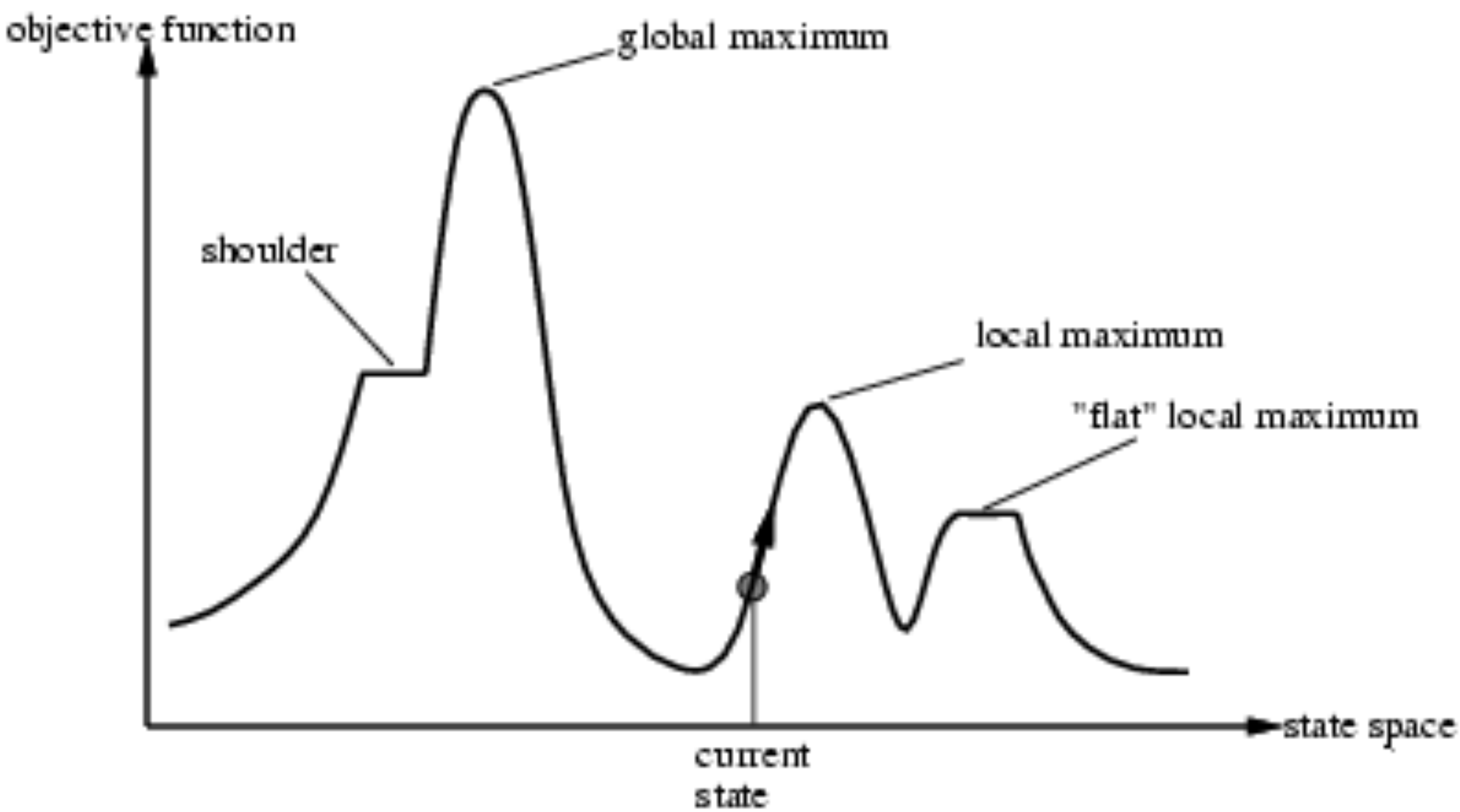
Global view of search space



Local view of search space

State space landscape

- Objective function defines *state space landscape*



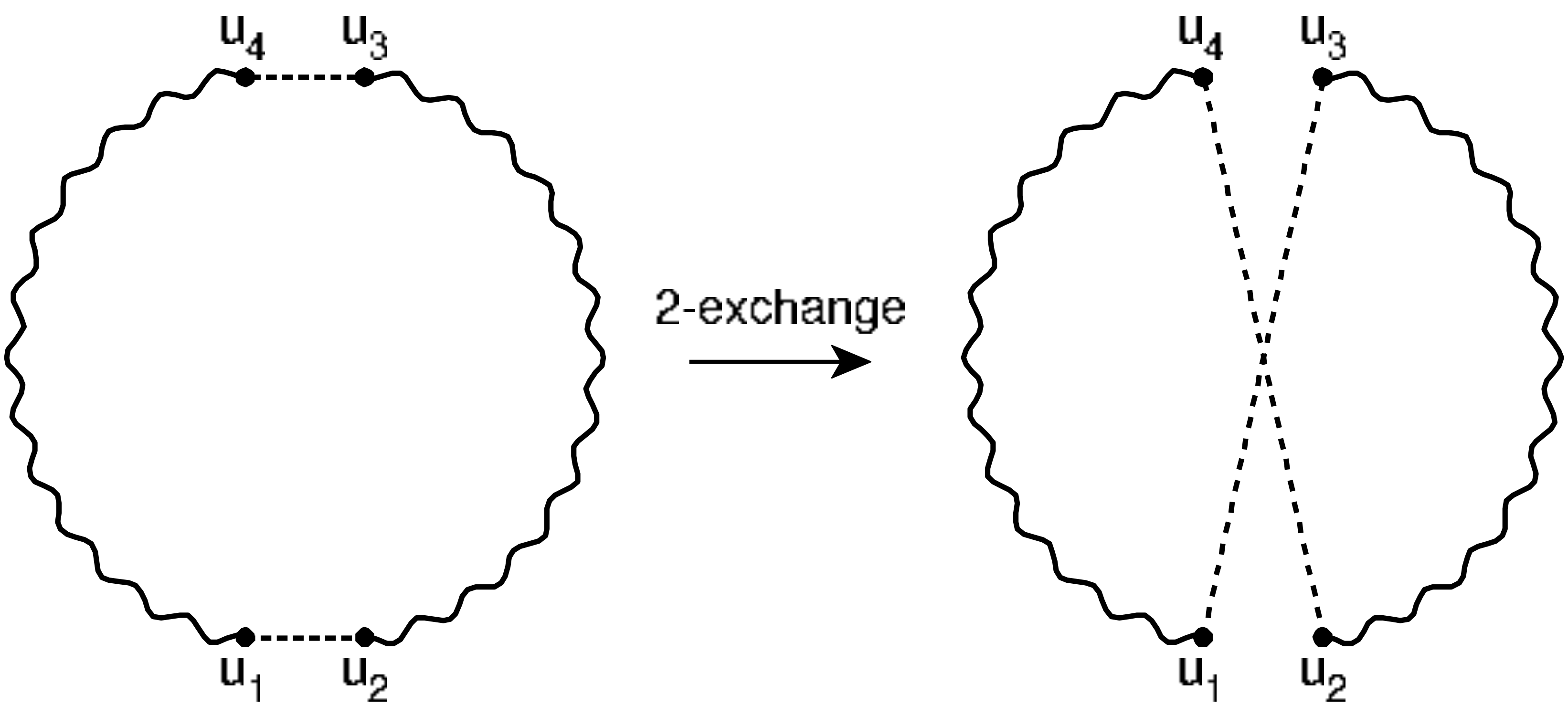
Local Search (LS) Algorithms

- search space S
  - SAT: set of all complete truth assignments to propositional variables (all "potential solutions")
- solution set $S' \subseteq S$
  - SAT: all satisfying assignments for given formula
- neighborhood relation $N \subseteq SxS$
  - A way to move from one potential solution to another
  - SAT: neighboring variable assignments differ in the truth value of exactly one variable
- evaluation function g : S → R+
  - SAT: number of clauses satisfied under given assignment

Local Search (LS) Algorithms

- search space S
  - TSP: set of all permutations of vertices (all "potential solutions")
- solution set $S' \subseteq S$
  - TSP: the tours of minimum length
- neighborhood relation $N \subseteq SxS$
  - A way to move from one potential solution to another
  - TSP: neighboring tour differ in several edges
- evaluation function g : S → R+
  - TSP: length of tour

Symm. TSP --- search neighborhood



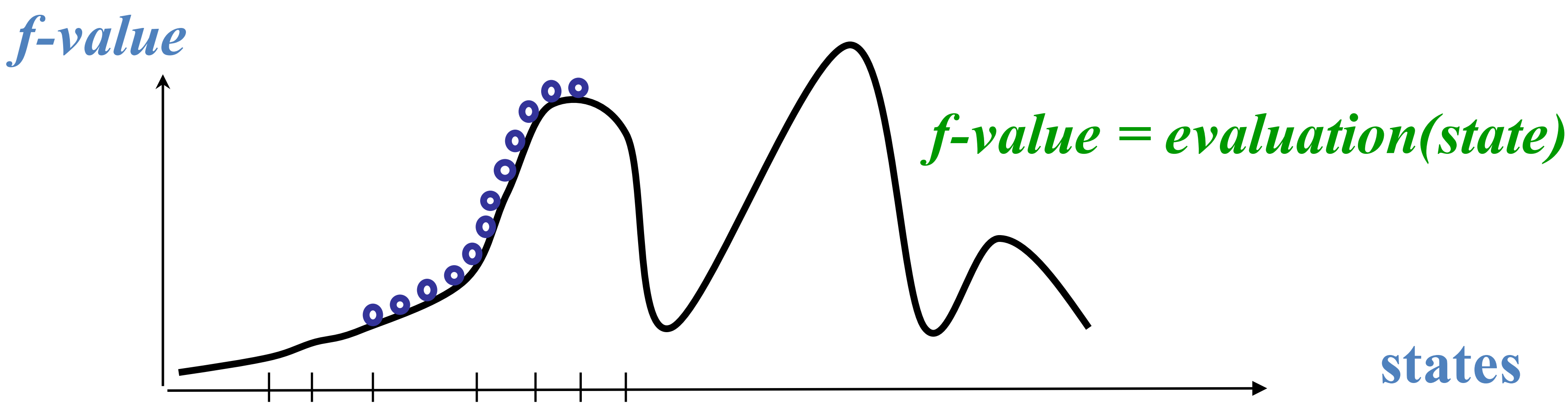
Search Space: all permutations of the cities (each defines a cycle)
3-opt – delete 3 edges and reconnect fragments into 1 cycle
k-opt – delete k edges and reconnect fragments into 1 cycle

Iterative Improvement (Greedy Search)

- Initialize search at some point of search space
- At each step, move from the current search position to a neighboring position with <u>better</u> evaluation function value

Hill climbing (Best Improvement Search)

- Choose the neighbor with the largest improvement as the next state



> **while** f-value(*state*) < f-value(next-best(*state*))
>
> *state* := next-best(*state*)

Hill climbing

```
function Hill-Climbing(problem) returns a solution state

current ← Make-Node(Initial-State[problem])
loop do
    next ← a highest-valued successor of current
    if Value[next] < Value[current] then return current
    current ← next
end
```

Problems with iterative improvement

- Advantages:
  - Very fast, works well for certain problems

- Disadvantages:
  - What if there are multiple peaks?
    - Hill climbing gets stuck at all peaks, known as local maxima
    - Optimal solution is highest peak – global maximum
    - May result in extremely suboptimal solution if many peaks
  - Being misguided by evaluation/objective function

Stochastic Local Search

- randomize initialization step
- randomize search steps such that suboptimal/worsening steps are allowed

- improved performance & robustness
- typically, degree of randomization controlled by noise parameter

Stochastic Local Search

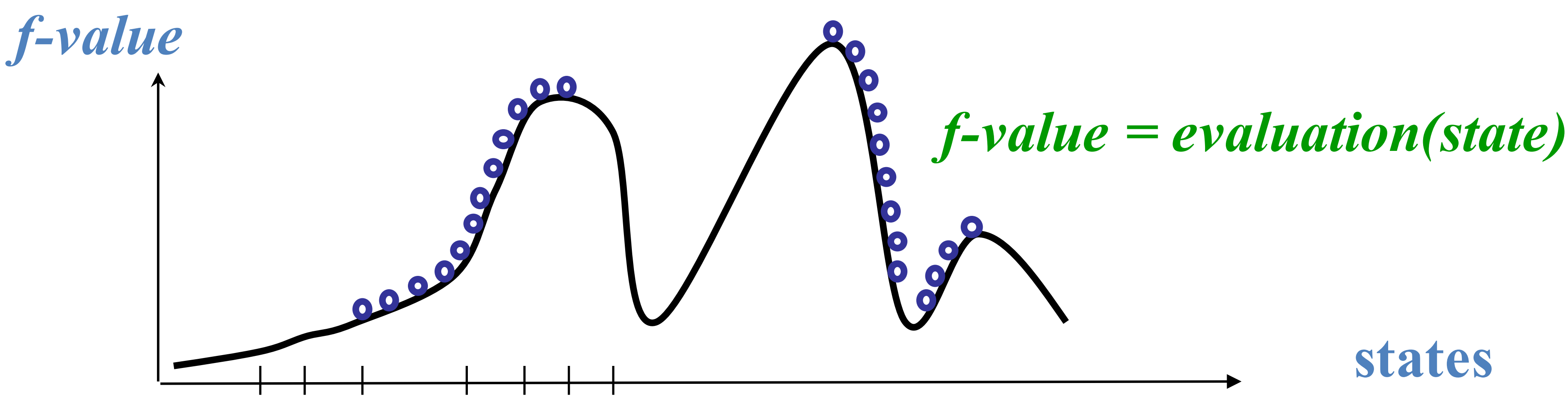
- Pros:
  - for many combinatorial problems, more efficient than systematic search
  - easy to implement
  - easy to parallelize
- Cons:
  - often incomplete (no guarantees for finding existing solutions)
  - highly stochastic behavior
  - often difficult to analyze theoretically/empirically

Simple SLS methods

- Random Search (Blind Guessing):
  - In each step, randomly select one element of the search space.
- (Uninformed) Random Walk:
  - In each step, randomly select one of the neighboring positions of the search space and move there.

Random restart hill climbing

- Start at random solution
- Hill-climb until local optima
- Start at another random position



Randomized Iterative Improvement:

- Idea: escape local maxima by <u>allowing some "bad" moves</u>

- initialize search at some point of search space
- search steps:
  - with probability p, move from current search position to a randomly selected neighboring position
  - otherwise, move from current search position to neighboring position with better evaluation function value

- Has many variations of how to choose the random neighbor, and how many of them

The WalkSAT Algorithm

```
WalkSAT(CNF, max-tries, max-flips, p) {
   for i ← 1 to max-tries do
      solution = random truth assignment
      for j ← 1 to max-flips do
         if all clauses in CNF satisfied then
            return solution
         c ← random unsatisfied clause in CNF
         with probability p
            flip a random variable in c
         else
            flip variable in c that maximizes
               number of satisfied clauses
  return failure
}
```

Simulated annealing (SA)

- Combinatorial search technique inspired by the physical process of annealing [Kirkpatrick et al. 1983, Cerny 1985]
- Outline
  - Select a neighbor at random
  - If better than current state, go there (improving move)
  - Otherwise, go there with some probability (worsening move)
  - Probability goes down with time (similar to temperature cooling)

- When probability is high → diversify (many worsening moves)
- When probability is low → intensify (focus on improving moves)

SA analogy

- Annealing is process of heating and cooling metals in order to improve strength
- Idea: Controlled heating and cooling of metal
  - When hot, atoms move around
  - When cooled, atoms find configuration with lower internal energy (i.e. makes metal stronger)
- Analogy:
  - Temperature = probability of accepting worse neighboring solution
    - When temperature is high, likely to accept worse neighboring solutions (but may lead to better overall solution)
      - Analogous to atoms wandering around
  - Cooling represents shrinking probability of accepting worse solutions

SA Pseudo code

function Simulated-Annealing(*problem, schedule*) **returns** *solution state*

current ← Make-Node(Initial-State[*problem*])

for t ← 1 to *infinity (Iters, Time cutoff)*

T ← *schedule*[*t*] // T goes downwards.

if *T = 0* **then return** *greedy from current*

next ← Random-Successor*(current)*

ΔE ← f-Value[*next*] - f-Value[*current*]

if *ΔE* **> 0** then *current* ← *next*

else *current* ← *next* with probability $e^{\Delta E/T}$

end

Simulated annealing (SA)

- Acceptance criterion (Metropolis condition): choose new solution s' over old solution s with probability (maximization)

$$\Pr(s', s) = \begin{cases} 1 & \text{if } f(s') > f(s) \\ \exp\left\{ \dfrac{f(s') - f(s)}{T} \right\} & \text{otherwise} \end{cases}$$

- Initial temperature T_0
- Annealing (cooling) schedule: how to update the temperature
  - E.g. T = a T with a =0.95 (geometric schedule)
  - Number of iterations at each temperature (e.g. multiple of the neighborhood size)
- Stopping criterion
  - E.g. no improved solution found for a number of iterations (or number of temperature values)

SA for TSP [Johnson & McGeoch 1997]

- baseline implementation:
  - start with random initial solution
  - use 2-exchange neighborhood
  - simple annealing schedule

- → relatively poor performance
- improvements:
  - look-up table for acceptance probabilities
  - neighborhood pruning
  - low-temperature starts

SA with restarts

- RESTARTS: Sometimes it is better to move back to a solution that was significantly better rather than always moving from the current state.

- The decision to restart could be based on several criteria.
  - based on a fixed number of steps,
  - based on whether the current energy is too high compared to the best energy obtained so far,
  - too many iterations without improvement,
  - restarting randomly, etc.

Summary of Simulated Annealing

- is historically important
- is easy to implement
- has interesting theoretical properties (convergence), but these are of very limited practical relevance
- achieves good performance often at the cost of substantial run-times

Tabu Search

- Combinatorial search technique that heavily relies on the use of an explicit memory of the search process [Glover 1989, 1990] to guide search process
- memory typically contains only specific attributes of previously seen solutions
- simple tabu search strategies exploit only short term memory
- more complex tabu search strategies exploit long term memory

Tabu search – exploiting short term memory

- in each step, move to best neighboring solution although it may be worse than current one
- to avoid cycles, tabu search tries to avoid revisiting previously seen solutions by basing the memory on attributes of recently seen solutions
- tabu list stores attributes of the TL most recently visited solutions; parameter TL is called tabu list length or tabu tenure
- solutions that contain tabu attributes are forbidden

Tabu Search

- Problem: previously unseen solutions may be tabu → use of aspiration criteria to override tabu status
- Stopping criteria:
  - all neighboring solutions are tabu
  - maximum number of iterations exceeded
  - number of iterations without improvement

Example: Tabu Search for SAT / MAX-SAT

- Neighborhood: assignments thatdiffer in exactly one variable instantiation
- Tabu attributes: variables
- Tabu criterion: flipping a variable is forbidden for a given number of iterations
- Aspiration criterion: if flipping a tabu variable leads to a better solution, the variable’s tabu status is overridden

- [Hansen & Jaumard 1990; Selman & Kautz 1994]

Iterated local search

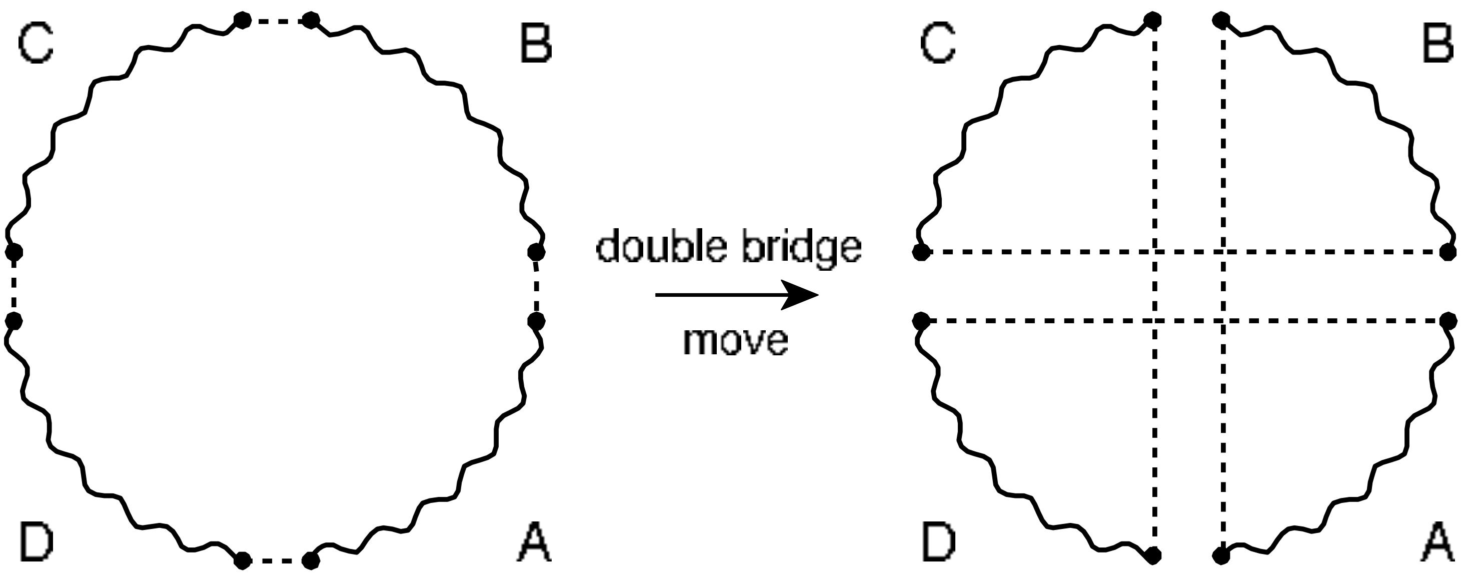
- Generate initial candidate solution s
- Perform local search on s (for example iterative improvement starting from s)
- While termination condition not met
  - Set r=s
  - Perform perturbation on s
  - Perform local search on perturbed s
  - Based on acceptance criterion, keep s or revert to r

Iterated local search

- ILS can be interpreted as walks in the space of local optima
- Perturbation is key
  - Needs to be chosen so that it cannot be undone easily by subsequent local search
  - It may consist of many perturbation steps
  - Strong perturbation: more effective escape from local optima but similar drawbacks as random restart
  - Weak perturbation: short subsequent local search phase but risk of revisiting previous optima
- Acceptance criteria: usually either the most recent or the better of two

Iterated local search for TSP

- Perturbation: "double-bridge move" = 4-exchange step
- Cannot be directly reversed by 2-exchange moves



Other search techniques

- Genetic algorithms
- Ant colony optimization

- Usually covered in AI courses

Construction heuristics for initial solutions

- search space: space of partial solutions
- search steps: extend partial solutions with assignment for the next element
- solution elements are often ranked according to a greedy evaluation function

TSP construction: Nearest neighbor

- Start at some vertex s; $v=s$;
- While not all vertices visited
  - Select closest unvisited neighbor w of v
  - Go from v to w
  - $v=w$
- Go from v to s

- Running time $O(n^2)$

TSP construction: Many variants

- **Closest insertion**: insert vertex closest to vertex in the tour
- **Farthest insertion**: insert vertex whose minimum distance to a node on the cycle is maximum
- **Cheapest insertion**: insert the node that can be inserted with minimum increase in cost
- **Random insertion**: randomly select a vertex and insert vertex at position that gives minimum increase of tour length

CSE 6140/ CX 4140

Empirical Analysis of Algorithms [SLS4]

textbook: STOCHASTIC LOCAL SEARCH FOUNDATIONS AND APPLICATIONS

based on slides by Holger Hoos

Theoretical *vs.* Empirical Analysis

Ideal: Analytically prove properties of a given algorithm (run-time: worst-case / average-case / distribution, error rates).

Reality: Often only possible under substantial simplifications or not at all.

$\rightsquigarrow$ Empirical analysis

The Three Pillars of CS:

- **Theory:** abstract models and their properties ("eternal thruths")
- **Engineering:** principled design of artifacts (hardware, systems, algorithms, interfaces)
- (**Empirical**) **Science:** principled study of phenomenae (behaviour of hardware, systems, algorithms; interactions)

The Scientific Method

make observations

formulate hypothesis/hypotheses (model)

While not satisfied (and deadline not exceeded) iterate:

1. design experiment to falsify model
2. conduct experiment
3. analyse experimental results
4. revise model based on results

Goals

- Defining standard methodologies
- Comparing relative performance of algorithms so as to identify the best ones for a given application
- Characterizing the behavior of algorithms
- Identifying algorithm separators, *i.e.*, families of problem instances for which the performance differ
- Providing new insights in algorithm design

Issues:

- algorithm implementation (fairness)
- selection of problem instances (benchmarks)
- performance criteria (what is measured?)
- experimental protocol
- data analysis & interpretation

Benchmark Selection

Some criteria for constructing/selecting benchmark sets:

- instance hardness (focus on hard instances)
- instance size (provide range, for scaling studies)
- instance type (provide variety):
  - individual application instances
  - hand-crafted instances (realistic, artificial)
  - ensembles of instances from random distributions ($\leadsto$ random instance generators)
  - encodings of various other types of problems (*e.g.*, SAT-encodings of graph colouring problems)

CPU Time *vs.* Elementary Operations

How to measure run-time?

- Measure CPU time (using OS book-keeping & functions)
- Measure elementary operations of algorithm
  (*e.g.*, local search steps, calls of expensive functions)
  and report cost model (CPU time / elementary operation)

Issues:

- accuracy of measurement
- dependence on run-time environment
- fairness of comparison