Simulation

Lecture H2

Heuristic Methods:
Iterated Local Search, Simulated Annealing and Tabu Search

Saeed Bastani
saeed.bastani@eit.lth.se
Spring 2017

Thanks to Prof. Arne Løkketangen at Molde University College (Norway), for sharing his presentation slides

Outline

- ✓ Iterated Local Search (ILS)
- ✓ Simulated Annealing (SA)
- ✓ Tabu Search (TA)

Iterated Local Search

- It is a meta-heuristic
- It is a simple extension of Local Search
- Aims at escaping local optima
- Relies on controlled **restarts**
  - Repeat (iterate) the same procedure over and over again, possibly with different starting solutions

Restarts (1)

- Given a Local Search procedure
  - After a while the algorithm stops
    - A Local Search stops in a local optimum
    - SA stops when the temperature has reached some lowest possible value (according to a cooling schedule)
  - What to do then?
- Restarts
  - Repeat (iterate) the same procedure over and over again, possibly with different starting solutions

Restarts (2)

- If everything in the search is deterministic (no randomization), it does no good to restart
- If something can be changed…
  - The starting solution
  - The random neighbor selection
  - Some controlling parameter (e.g., the temperature)
- … then maybe restarting can lead us to a different (and thus possibly better) solution

Iterated Local Search (1)

- We can look at a Local Search (using ”Best Improvement”-strategy) as a function
  - Input: a solution
  - Output: a solution
  - LS: S → S
  - The set of local optima (with respect to the neighborhood used) equals the range of the function
- Applying the function to a solution returns a locally optimal solution (possibly the same as the input)

Iterated Local Search (2)

- A simple algorithm (Multi-start Local Search):
  - Pick a random starting solution
  - Perform Local Search
  - Repeat (record the best local optimum encountered)
- Generates multiple independent local optima
- Theoretical guarantee: will encounter the global optimum at some point (due to random starting solution)
- Not very efficient: wasted iterations

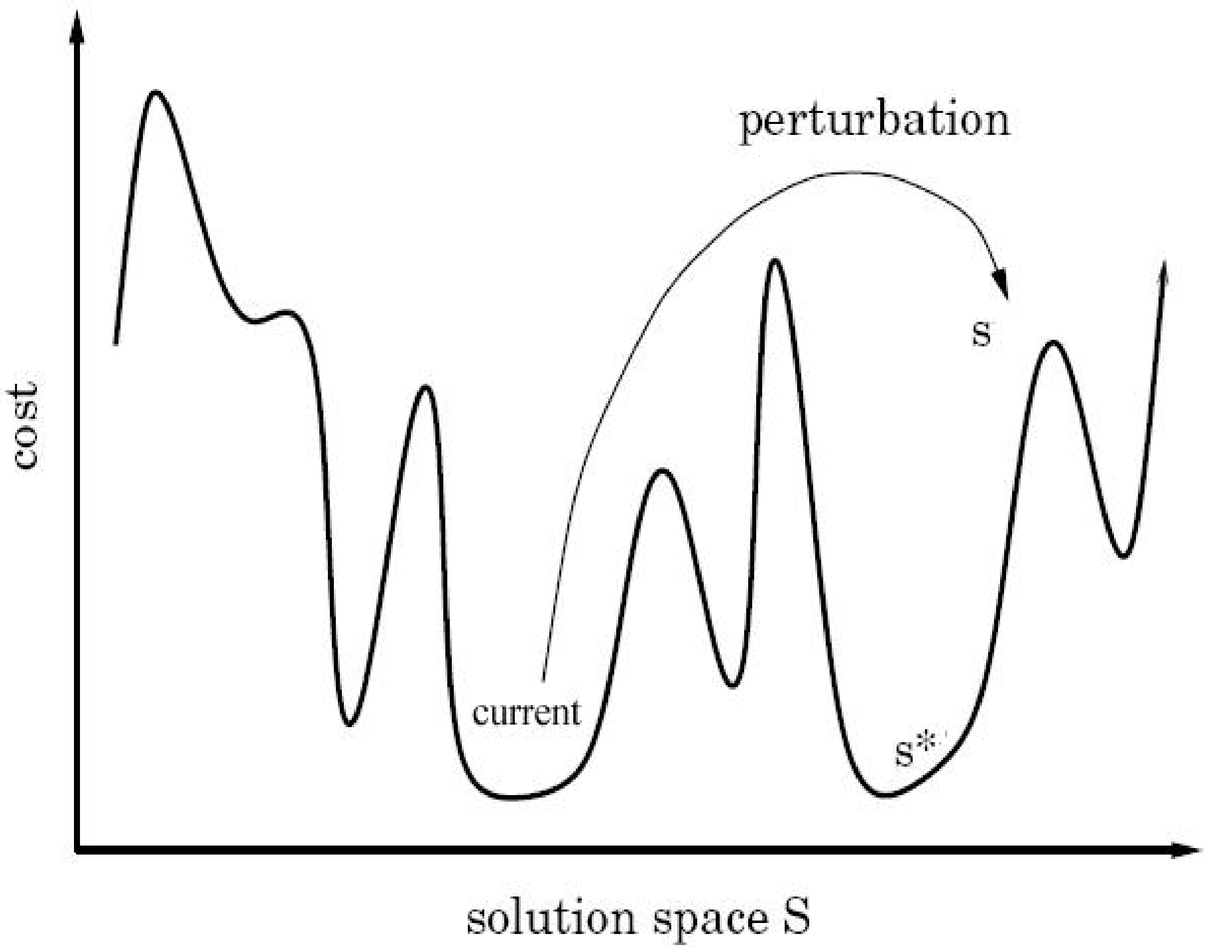
Iterated Local Search (3)

- Iterated Local Search tries to benefit by restarting close to a currently selected local optimum
  - Possibly quicker convergence to the next local optimum (already quite close to a good solution)
  - Has potential to avoid unnecessary iterations in the Local Search loop, or even unnecessary complete restarts
    - Uses information from current solution when starting another Local Search

Iterated Local Search

input: starting solution, s_0
input: Local Search procedure, LS
$current \Leftarrow LS(s_0)$
while stopping criterion not met **do**
    $s \Leftarrow$ perturbation of $current$ based on search history
    $s^* \Leftarrow LS(s)$
    if s^* is accepted as the new current solution **then**
        $current \Leftarrow s^*$
    end if
end while

Pictorial Illustration of ILS



Principle of Iterated Local Search

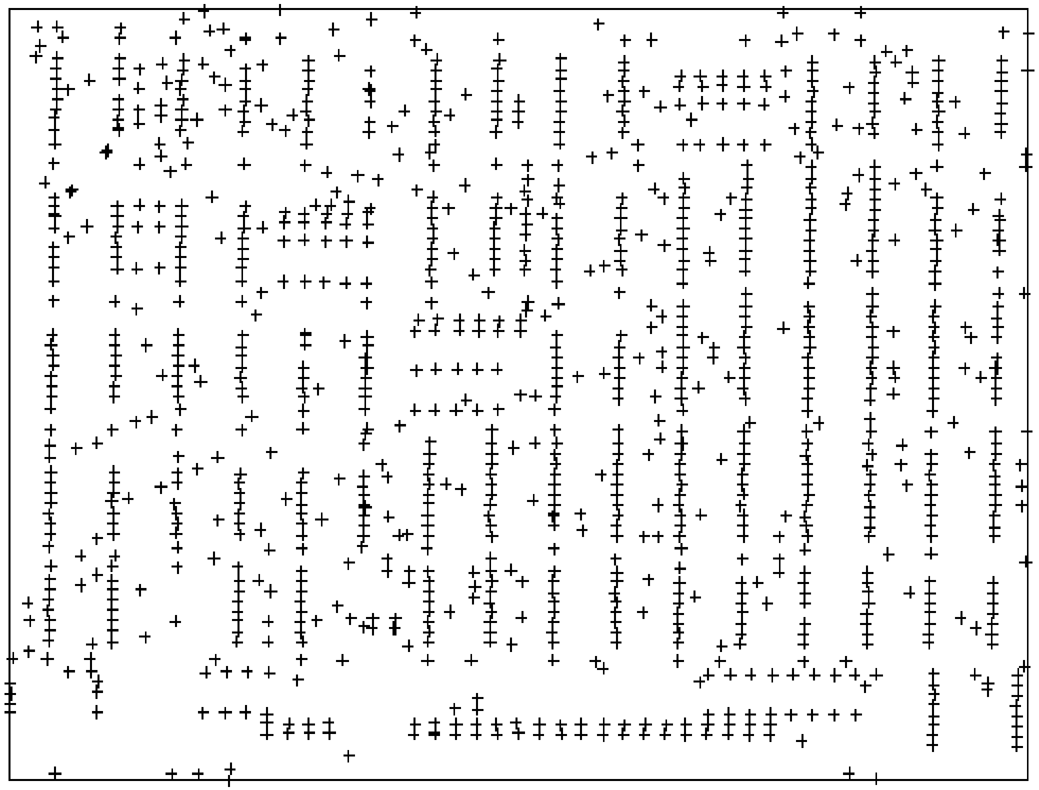
- The Local Search algorithm defines a set of locally optimal solutions
- The Iterated Local Search metaheuristic searches among these solutions, rather than in the complete solution space
  - The search space of the ILS is the set of local optima
  - The search space of the LS is the solution space (or a suitable subspace thereof)

A Basic Iterated Local Search

- Initial solution:
  - Random solution
  - Construction heuristic
- Local Search:
  - Usually readily available (given some problem, someone has already designed a local search, or it is not too difficult to do so)
- Perturbation:
  - A random move in a ”higher order neighborhood”
  - If returning to the same solution (*s*=current*), then increase the strength of the perturbation?
- Acceptance:
  - Move only to a better local optimum

ILS Example: TSP (1)

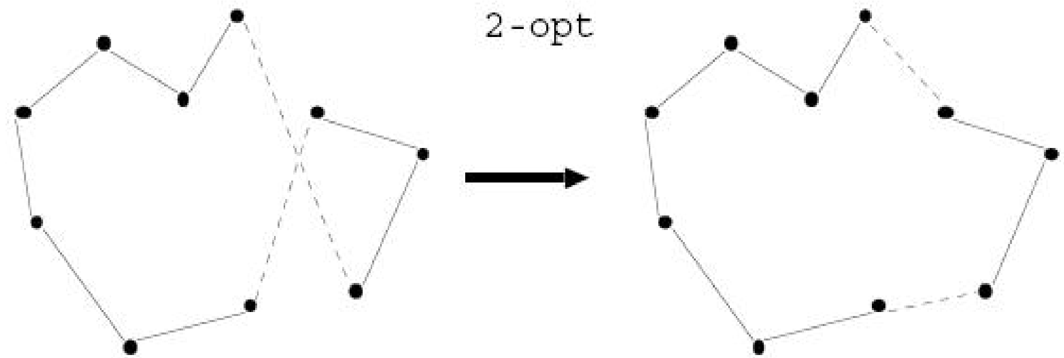
- Given:
  - Fully connected, weighted graph
- Find:
  - Shorted cycle through all nodes
- Difficulty:
  - NP-hard
- Interest:
  - Standard benchmark problem



(Example stolen from slides by Thomas Stützle)

ILS Example: TSP (2)

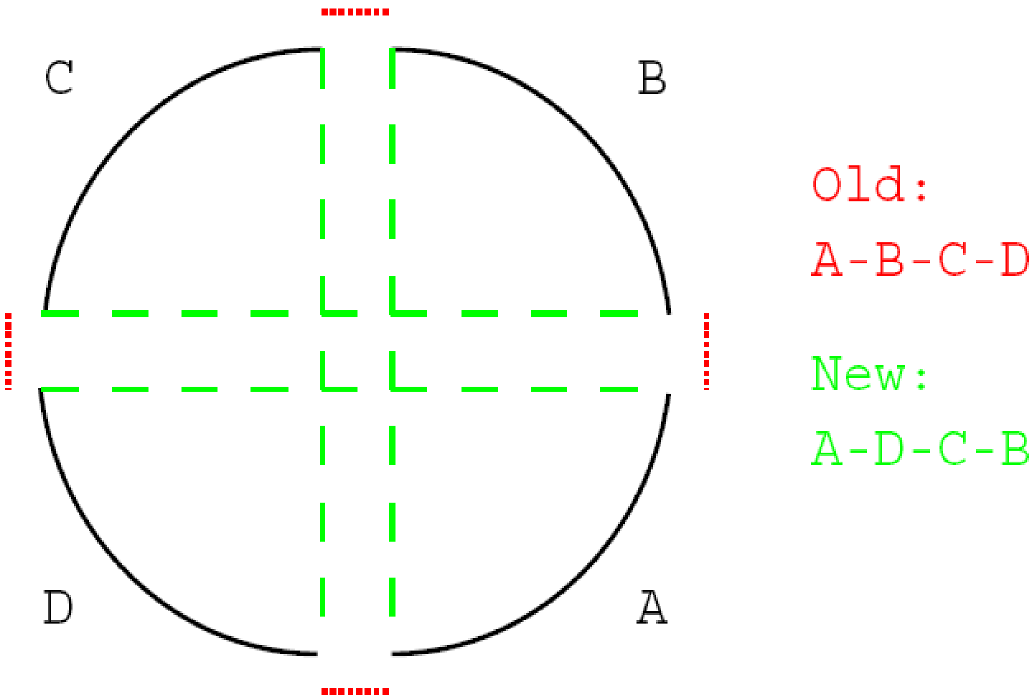
- Initial solution: greedy heuristic
- Local Search: 2-opt



- Perturbation: double-bridge move (a specific 4-opt move)
- Acceptance criterion: accept s* if f(s*) ≤ f(current)

ILS Example: TSP (3)

- Double-bridge move for TSP:



About Perturbations

- The strength of the perturbation is important
  - Too strong: close to random restart
  - Too weak: Local Search may undo perturbation
- The strength of the perturbation may vary at run-time
- The perturbation should be complementary to the Local Search
  - E.g., 2-opt and Double-bridge moves for TSP

About the Acceptance Criterion

- Many variations:
  - Accept s* only if f(s*)<f(current)
    - Extreme intensification
    - Random Descent in space of local optima
  - Accept s* always
    - Extreme diversification
    - Random Walk in space of local optima
  - Intermediate choices possible
- For TSP: high quality solutions known to cluster
  - A good strategy would incorporate intensification

ILS Example: TSP (4)

- $\Delta_{avg}(x)$ = average deviation from optimum for method x
- RR: random restart
- RW: ILS with random walk as acceptance criterion
- Better: ILS with First Improvement as acceptance criterion

instance	Δ_{avg}(RR)	Δ_{avg}(RW)	Δ_{avg}(Better)
kroA100	0.0	0.0	0.0
d198	0.003	0.0	0.0
lin318	0.66	0.30	0.12
pcb442	0.83	0.42	0.11
rat783	2.46	1.37	0.12
pr1002	2.72	1.55	0.14
d1291	2.21	0.59	0.28
fl1577	10.3	1.20	0.33
pr2392	4.38	2.29	0.54
pcb3038	4.21	2.62	0.47
fl3795	38.8	1.87	0.58
rl5915	6.90	2.13	0.66

ILS: The Local Search

- The Local Search used in the Iterated Local Search metaheuristic can be handled as a ”Black Box”
  - If we have any improvement method, we can use this as our Local Search and focus on the other parts of the ILS
  - Often though: a good Local Search gives a good ILS
- Can use very complex improvement methods, even such as other metaheuristics (e.g., SA)

Guidelines for ILS

- The starting solution should to a large extent be irrelevant for longer runs
- The Local Search should be as effective and fast as possible
- The best choice of perturbation may depend strongly on the Local Search
- The best choice of acceptance criterion depends strongly on the perturbation and Local Search
- Particularly important: the interaction among perturbation strength and the acceptance criterion

A Comment About ILS and Metaheuristics

- After seeing Iterated Local Search, it is perhaps easier to understand what a metaheuristic is
- ILS required that we have a Local Search algorithm to begin with
  - When a local optimum is reached, we perturb the solution in order to escape from the local optimum
  - We control the perturbation to get good behaviour: finding an improved local optimum
- ILS ”controls” the Local Search, working as a ”meta”-heuristic (the Local Search is the underlying heuristic)
  - Meta- in the meaning ”more comprehensive”; ”transcending”

Simulated Annealing

Simulated Annealing

- A metaheuristic inspired by statistical thermodynamics
  - Based on an analogy with the cooling of material in a heat bath
- Used in optimization for 20 years
- Very simple to implement
- A lot of literature
- Converges to the global optimum under weak assumptions (- usually slowly)

Simulated Annealing - SA

- Metropolis' algorithm (1953)
  – Algorithm to simulate energy changes in physical systems when cooling
- Kirkpatrick, Gelatt and Vecchi (1983)
  – Suggested to use the same type of simulation to look for good solutions in a COP

SA - Analogy

Thermodynamics	Discrete optimization
1. Configuration of particles	1. Solution
2. System state	2. Feasible solution
3. Energy	3. Objective Function
4. State change	4. Move to neighboring solution
5. Temperature	5. Control Parameter
6. Final state	6. Final Solution

Simulated Annealing

- Can be interpreted as a modified random descent in the space of solutions
  - Choose a random neighbor
  - Improving moves are always accepted
  - Deteriorating moves are accepted with a probability that depends on the amount of the deterioration and on the *temperature* (a parameter that decreases with time)
- Can escape local optima

Move Acceptance in SA

- We assume a minimization problem
- Set Δ = Obj(random neighbor) – Obj(current solution)
- If Δ < 0 → accept (we have an improving move)
- Else accept if

$$Random(0,1) < e^{-\frac{\Delta}{t}}$$

- If the move is not accepted: try another random neighbor

SA - Structure

- Initial temperature t_0 high
  - (if $\infty \rightarrow$ random walk)
- Reduce t regularly
  - need a *cooling schedule*
  - if too fast $\rightarrow$ stop in some local optimum too early
  - if too slow $\rightarrow$ too slow convergence
- Might restart
- Choice of neighborhood structure is important

SA

- Statistical guarantee that SA finds the global optimum
- In practice this requires exponential (or ∞) running time
- The cooling schedule is vitally important
  - Much research on this
  - Static schedules: specified in advance
  - Adaptive schedules: react to information from the search

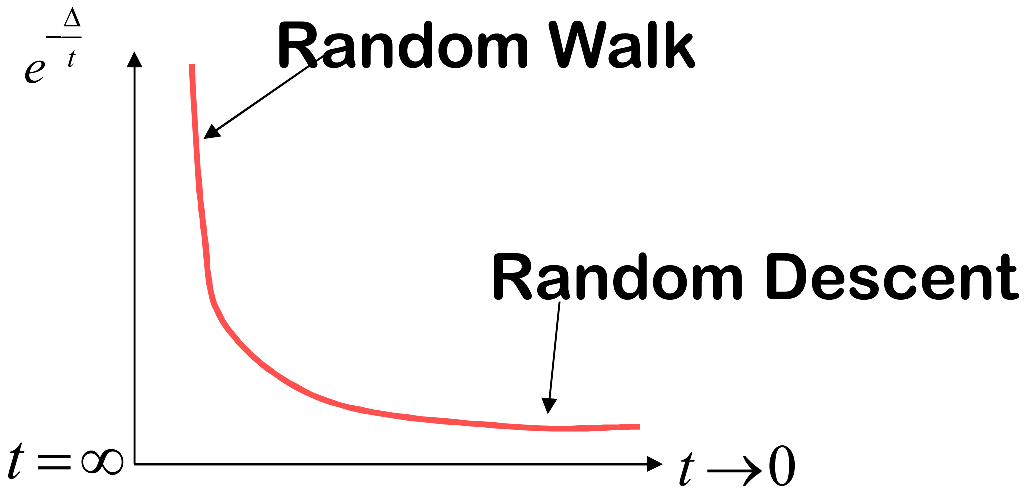
Simulated Annealing

```
input: starting solution, s_0
input: neighborhood operator, N
input: evaluation function, f
input: the cooling schedule, t_k
input: the number of iterations for each temperature, M_k
current ⇐ s_0
k ⇐ 0
while stopping criterion not met do
   m ⇐ 0
   while m < M_k do
      s ⇐ randomly selected solution from N(current)
      if f(s) ≤ f(current) then
         current ⇐ s
      else
         Δ ⇐ f(s) − f(current)
         ξ ⇐ a random number, uniformly drawn from [0, 1]
         if ξ ≤ e^{−Δ/t_k} then
            current ⇐ s
         end if
      end if
      m ⇐ m + 1
   end while
   k ⇐ k + 1
end while
```

Choice of Move in SA

- Modified ”Random Descent”
- Select a random solution in the neighborhood
- Accept this
  - Unconditionally if better than current
  - With a certain, finite probability if worse than current
- The probability is controlled by a parameter called the *temperature*
- Can escape from local optima

SA – Cooling Schedule



- Requires:
  - Good choice of cooling schedule
  - Good stopping criterion
  - Faster cooling at the beginning and end
  - Testing is important

SA – Overall Structure

- Set the initial value of the control variable t (t_0) to a high value
- Do a certain number of iterations with the same temperature
- Then reduce the temperature $t_{i+1} = \alpha(t_i)$
- Need a ”cooling schedule”
- Stopping criterion – e.g. ”minimum temperature”
  - Repetition is possible
- Solution quality and speed are dependent on the choices made
- Choice of neighborhood structure is important

Statistical Analysis of SA

- Model: State transitions in the search space
- Transition probabilities $[p_{ij}]$ (i,j are solutions)
- Only dependent on i and j: homogenous Markov chain
- If all the transition probabilities are finite, then the SA search will converge towards a stationary distribution, independent of the starting solution.
  - When the temperature approaches zero, this distribution will approach a uniform distribution over the global optima
- Statistical guarantee that SA finds a global optimum
- But: exponential (or infinite) search time to guarantee finding the optimum

SA in Practice (1)

- Heuristic algorithm
- Behaviour strongly dependent on the cooling schedule
- Theory:
  - An exponential number of iterations at each temperature
- Practice:
  - A large number of iterations at each temperature, few temperatures
  - A small number of iterations at each temperature, many temperatures

SA in Practice (2)

- Geometric chain
  - $t_{i+1} = \alpha\, t_i$, $i = 0,\ldots,K$
  - $\alpha < 1$ (0.8 - 0.99)
- Number of repetitions can be varied
- Adaptivity:
  - Variable number of moves before the temperature reduction
- Necessary to experiment

SA – General Decisions

- Cooling Schedule
  - Based on maximum difference in the objective function value of solutions, given a neighborhood
  - Number of repetitions at each temperature
  - Reduction rate, α
- Adaptive number of repetitions
  - more repetitions at lower temperatures
  - number of accepted moves, but a maximum limit
- Very low temperatures are not necessary
- Cooling rate most important

SA – Problem Specific Decisons

- Important goals
  - Response time
  - Quality of the solution
- Important choices
  - Search space
    - Infeasible solutions – should they be included?
  - Neighborhood structure
  - Move evaluation function
    - Use of penalty for violated constraints
    - Approximation – if expensive to evaluate
  - Cooling schedule

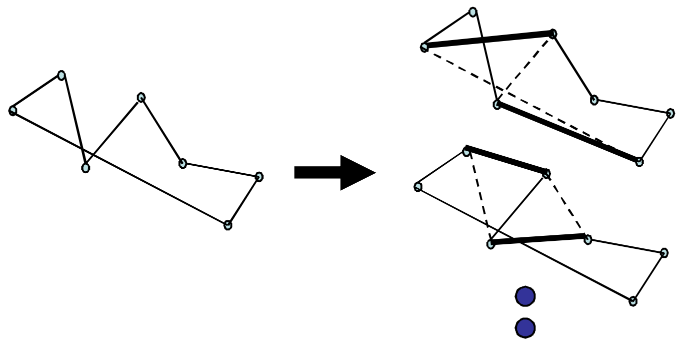
SA – Choice of Neighborhood

- Size
- Variation in size
- Topology
  - Symmetry
  - Connectivity
    - Every solution can be reached from all the others
- Move evaluation function
  - How expensive is it to calculate ?

SA - Speed

- Random choice of neighbor
  - Reduction of the neighborhood
  - Does not search through all the neighbors
- Cost of new candidate solution
  - Difference without full evaluation
  - Approximation (using surrogate functions)
- Move acceptance criterion
  - Simplify

SA – Example: TSP



- Search space - (n-1)!/2
- Neighborhood size:
  - 2-opt: n(n-1)/2
- Connected
- Simple representation of moves
- Natural cost function
- Difference in cost between solutions is easy to calculate
- Generalization: k-Opt

SA – Fine Tuning

- Test problems
- Test bench
- Visualization of solutions
- Values for
  - cost / penalties
  - temperature
  - number / proportion of accepted move
  - iterations / CPU time
- Depencies between the SA-parameters
- The danger of overfitting

SA – Modifications and Extensions

- Probabilistic
  - Altered acceptance probabilities
  - Simplified cost functions
  - Approximation of exponential function
    - Can use a look-up table
  - Use few temperatures
  - Restart
- Deterministic
  - Threshold Accepting, TA
  - Cooling schedule
  - Restart

SA – Combination with Other Methods

- Preprocessing – find a good starting solution
- Standard local search during the SA
  - Every accepted move
  - Every improving move
- SA in construction heuristics

Threshold Accepting

- Extensions/generalizations
  - Deterministic annealing
  - Threshold acceptance methods
  - Why do we need randomization?
- Local search methods in which deterioration of the objective up to a ***threshold*** is accepted
  - Accept if and only if $\Delta \leq \Theta_k$
- Does not have proof of convergence, but in practice results have been good compared to SA

Threshold Accepting

input: starting solution, s_0
input: neighborhood operator, N
input: evaluation function, f
input: threshold, Θ
$current \Leftarrow s_0$
while stopping criterion not met **do**
    $s \Leftarrow$ randomly selected solution from $N(current)$
    $\Delta \Leftarrow f(s) - f(current)$
    if $\Delta < \Theta$ **then**
        $current \Leftarrow s$
    end if
end while

Generalized Hill-Climbing Algorithms

- Generalization of SA
- General framework for modeling Local Search Algorithms
  - Can describe Simulated Annealing, Threshold Accepting, and some simple forms of Tabu Search
  - Can also describe simple Local Search variations, such as the ”First Improvement”, ”Best Improvement”, ”Random Walk” and ”Random Descent”-strategies

Generalized Hill-Climbing Algorithm

```
input: starting solution, s_0
input: neighborhood operator, N
input: evaluation function, f
input: outer loop bound, K, inner loop bounds M_k, k = 1, 2, ..., K
input: hill-climbing (random) functions R_k : S × S → ℝ ∪ {−∞, +∞}
current ⇐ s_0
k ⇐ 1
m ⇐ 1
while k ≤ K do
    while m ≤ M_k do
        s ⇐ solution generated from N(current)
        Δ ⇐ f(s) − f(current)
        if R_k(current, s) ≥ Δ then
            current ⇐ s
        end if
        m ⇐ m + 1
    end while
    k ⇐ k + 1
end while
```

Generalized Hill-Climbing Algorithms (2)

- The flexibility comes from
  - Different ways of generating the neighbors
    - Randomly
    - Deterministically
    - Sequentially, sorted by objective function value?
  - Different acceptance criteria, R_k
    - Based on a threshold (e.g., Threshold Accepting)
    - Based on a temperature and difference in evaluation (e.g., SA)
    - Other choices?

Tabu Search

Tabu

- The word tabu (or taboo) comes from Tongan
  - a language of Polynesia
  - used by the aborigines of Tonga island to indicate things that cannot be touched because they are sacred
- Meaning of Tabu:
  - *”Loaded with a dangerous, unnatural force”*
  - *”Banned due to moral, taste or risk”*

Tabu Search

- Tabu Search:
  - Cut off the search from parts of the search space (temporarily)
  - Guide the search towards other parts of the search by using penalties and bonuses
- Uses principles for intelligent problem solving
- Uses structures that are exploring the search history, without remembering everything
  - Branch&Bound, A*: have complete memory
  - Simulated Annealing: have no memory

Origin of Tabu Search

- Fred Glover 1986: ”Future paths for integer programming and links to artificial intelligence”
- Pierre Hansen 1986: ”The Steepest Ascent/Mildest Descent Heuristic for Combinatorial Optimization”
- *Tabu* coined by Glover

Main Ideas of Tabu Search

- Based on Local Search – LS
- Allows non-improving moves
  - can exit local optima
- Uses extra memory to avoid looping, and to diversify the search
- General strategy for controlling a LS, or other “inner” heuristic
- *Meta-Heuristic* (Glover)

General Formulation

Tabu Search

```
current ⇐ a starting solution
Initialize tabu memory
while stopping criterion not met do
    Find a list of candidate moves, a subset of N(current)
    Select the solution, s, in the candidate list that minimizes an extended
    cost function
    Update tabu memory and perform the move: current ⇐ s
end while
```

Some Critical Choices

- Choice of neighborhood, N
- Definition of the tabu memory
- How to select the candidate list
- The definition of the evaluation function
  - Improvement in solution values
  - Tabu criteria
  - Aspiration criteria
  - Long term strategies
    - Diversification, intensification, …

Basic Tabu Search

- Local Search with "Best Improvement" strategy
  - Always select the best move
- But: some neighbors are *tabu,* and cannot be selected
  - Defined by the *tabu criterion*
  - Tabu neighbors might be selected anyway if they are deemed to be good enough
    - *Aspiration criterion*
- Memory – tabu list

The Tabu Criterion (1)

- Since we (in basic TS) always select the ”Best Improvement”, how can we avoid cycling between solutions?
- The answer is the tabu criterion:
  - We are not allowed to move to solutions that we have visited before
    - They are tabu!

The Tabu Criterion (2)

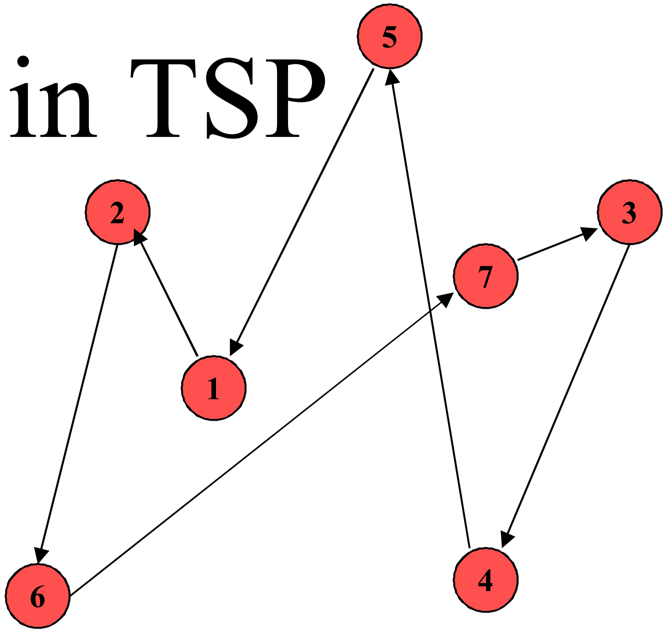
- The basic job of the tabu criterion is thus to avoid visiting the same solution more than once
- How to accomplish this?
  - Store all the solutions visited during the search, and check that the new solution is not among those previously visited
    - Too time consuming!
  - Find some way of (approximately) represent those solutions that we have seen most recently, and avoid returning immediately to those (or similar) solutions

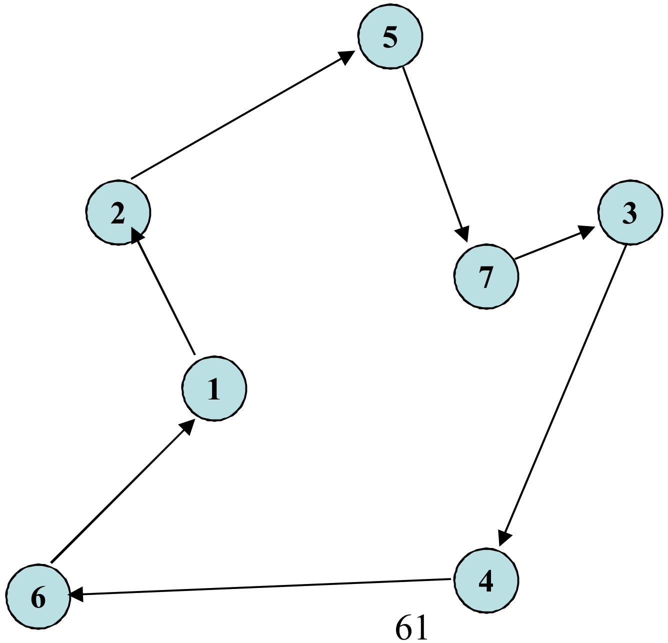
Tabu Attribute Selection

- Attribute
  - A property of a solution or a move
- Can be based on any aspect of the solution that are changed by a move
- Attributes are the basis for tabu restrictions
  - We use them to represent the solutions visited recently
- A move can change more than one attribute
  - e.g. a 2-opt move in TSP involves 4 cities and 4 edges

Example – Attributes in TSP

- Attributes based on the edges
  - A1: Edges added to the tour
  - A2: Edges removed from the tour
- Move
  - Exchanges two cities
  - 4 edges removed
  - 4 edges added
  - Exchange(5,6)
    - A1:(2,5),(5,7),(4,6),(6,1)
    - A2:(2,6),(6,7),(4,5),(5,1)





TS – Tabu Criterion

- The tabu criterion is defined on selected attributes of a move, (or the resulting solution if the move is selected)
- It is very often a component of the solution
- The attribute is tabu for a certain amount of time (i.e. iterations)
  - This is called the ***Tabu Tenure*** **(TT)**
- The tabu criterion usually avoids the immediate move reversal (or repetition)
- It also avoids the other (later) moves containing the tabu attribute. This cuts off a much larger part of the search space

TS – Attributes and Tabu Criteria

- Can have several tabu criteria on different attributes, each with its own tabu tenure
  - These can be disjunct
- If a move is to exchange a component (e.g. *edge*) *in* the solution with a component *not in* the solution, we can have the following tabu attributes and criteria
  - Edge added
  - Edge dropped
  - Edge added or edge dropped
  - Edge added and edge dropped

Use of Attributes in Tabu Restrictions

- Assume that the move from $s_k \rightarrow s_{k+1}$ involves the attribute A
- The usual tabu restriction:
  - Do not allow moves that reverse the status for A
- The TSP example:
  - Move: exchange cities 2 and 5: $x_{2,5}$
  - The tabu criterion could disallow:
    - Moves involving 2 and 5
    - Moves involving 2 or 5
    - Moves involving 2
    - Moves involving 5

Tabu Tenure (1)

- The tabu criterion will disallow moves that change back the value of some attribute(s)
- For how long do we need to enforce this rule?
  - For ever: the search stops because no changes are allowed
  - For too long: the search might become too limited (too much of the search space is cut off due to the tabu criterion)
  - For too short: the search will still cycle
- The number of iterations for which the value of the attribute remains tabu is called the *Tabu Tenure*

Tabu Tenure (2)

- Earlier: The magical number 7, plus or minus 2
- Sometimes: in relation to problem size: $n^{1/2}$
- Static (fixed) tabu tenure is not recommended
  - The search gets more easily stuck in loops
- Dynamic tabu tenure is highly recommended
  - Change the tabu tenure at certain intervals
  - Can use uniform random selection in $[tt_1, tt_2]$
    - This is usually called dynamic, even though it is not
- Reactive Tabu Search
  - Detect stagnation $\rightarrow$ increase TT
  - When escaped $\rightarrow$ reduce TT

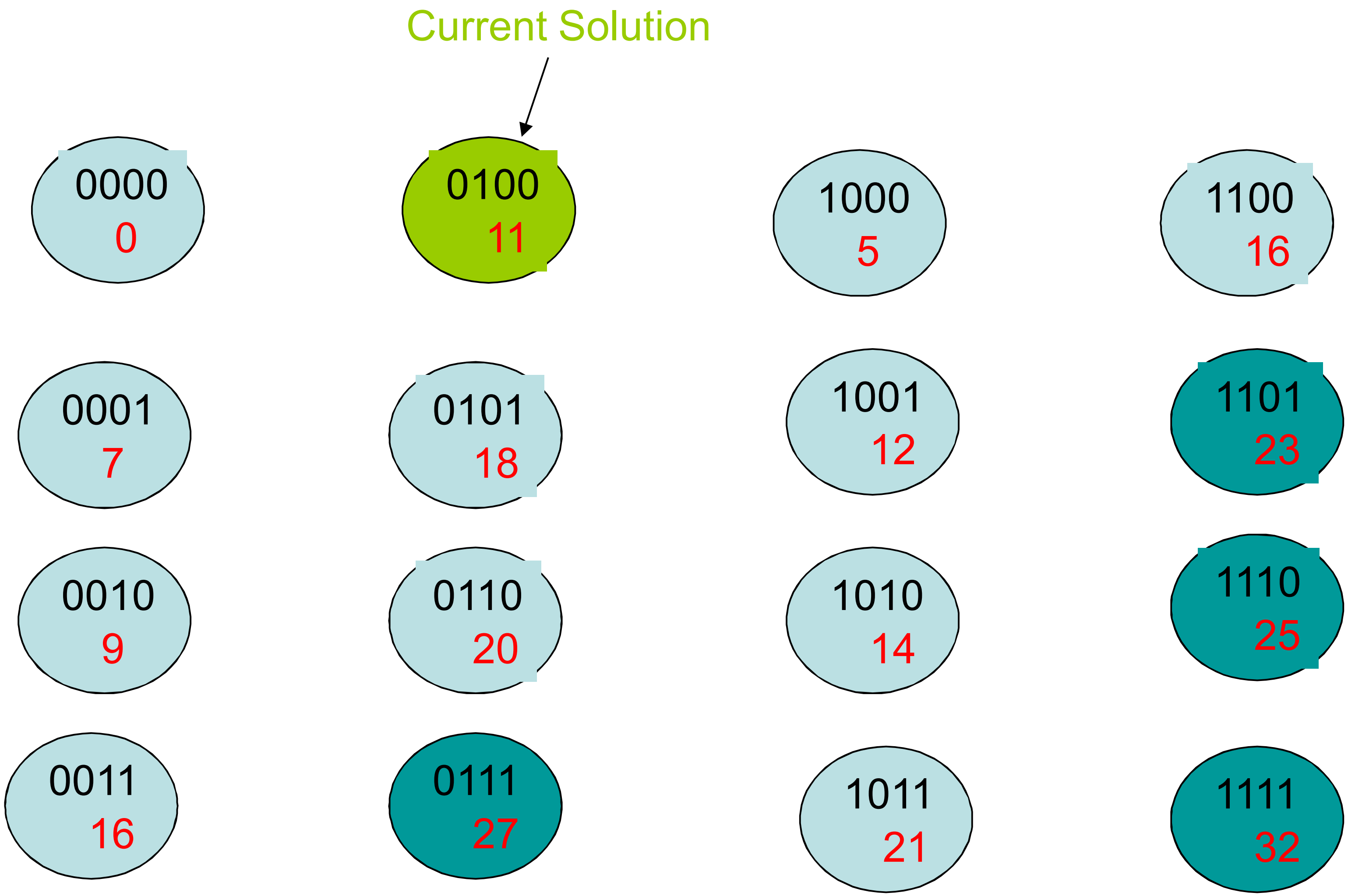
Tabu Tenure (3)

- Dependent on the tabu attributes
- Example: TSP – n cities – 2-opt
  - Use *edges-added* and *edges-dropped* as tabu attributes
  - $|n^2|$ edges in the problem instance
  - $|n|$ edges in the solution
  - Many more edges outside the solution than in the solution
  - Using the same TT would be unbalanced

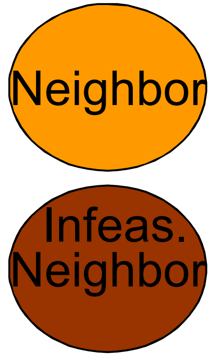
Example: 0/1 Knapsack

- Flip-Neighborhood
- If the move is selecting an item to include in the solution, then any move trying to remove the same item is *tabu* for the duration of the *tabu tenure*
- Similarly, an item thrown out is not allowed in for the duration of the tabu tenure iterations
- Here the attribute is the same as the whole move

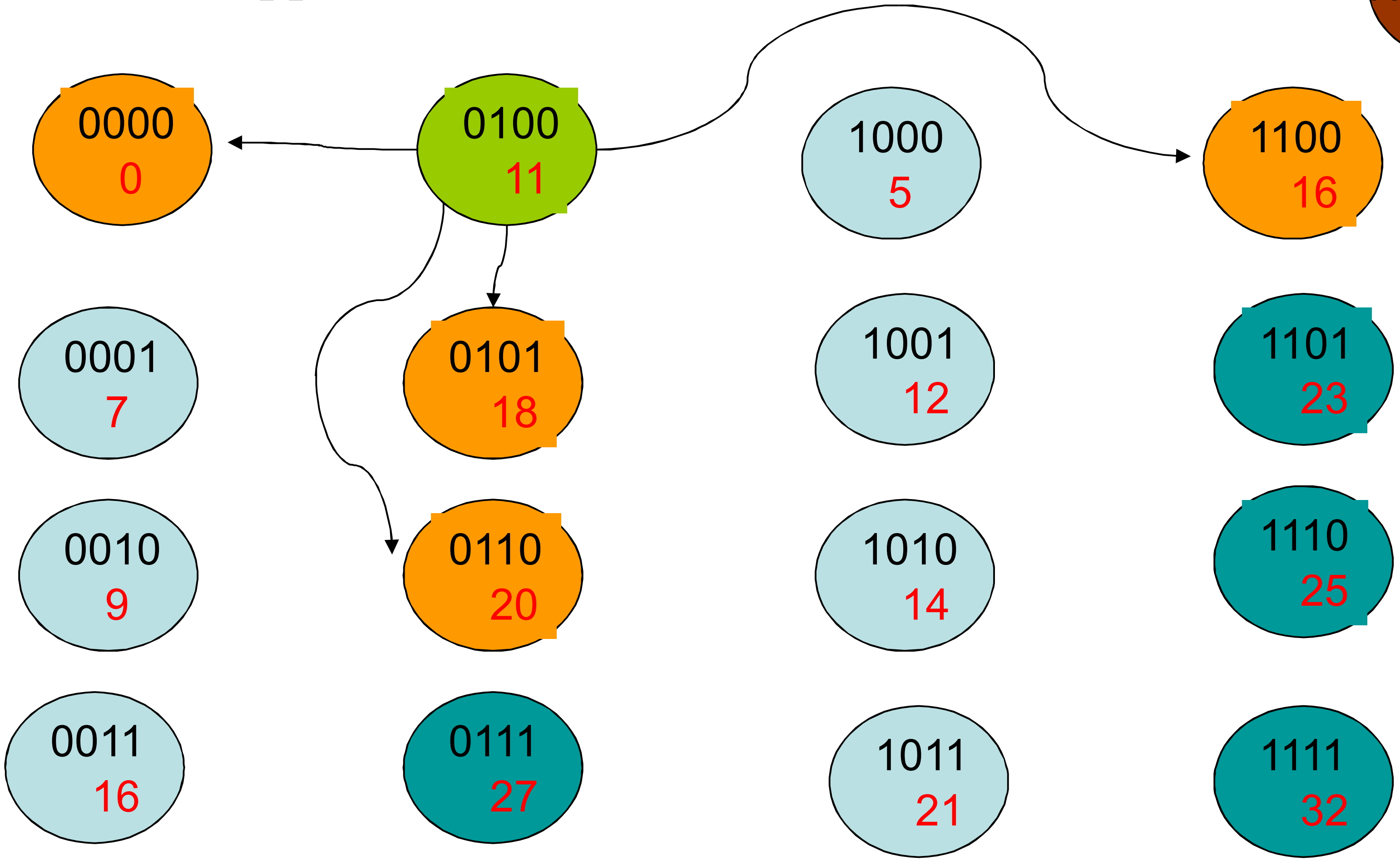
Flip Neighborhood



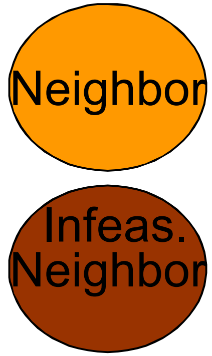
Flip Neighborhood



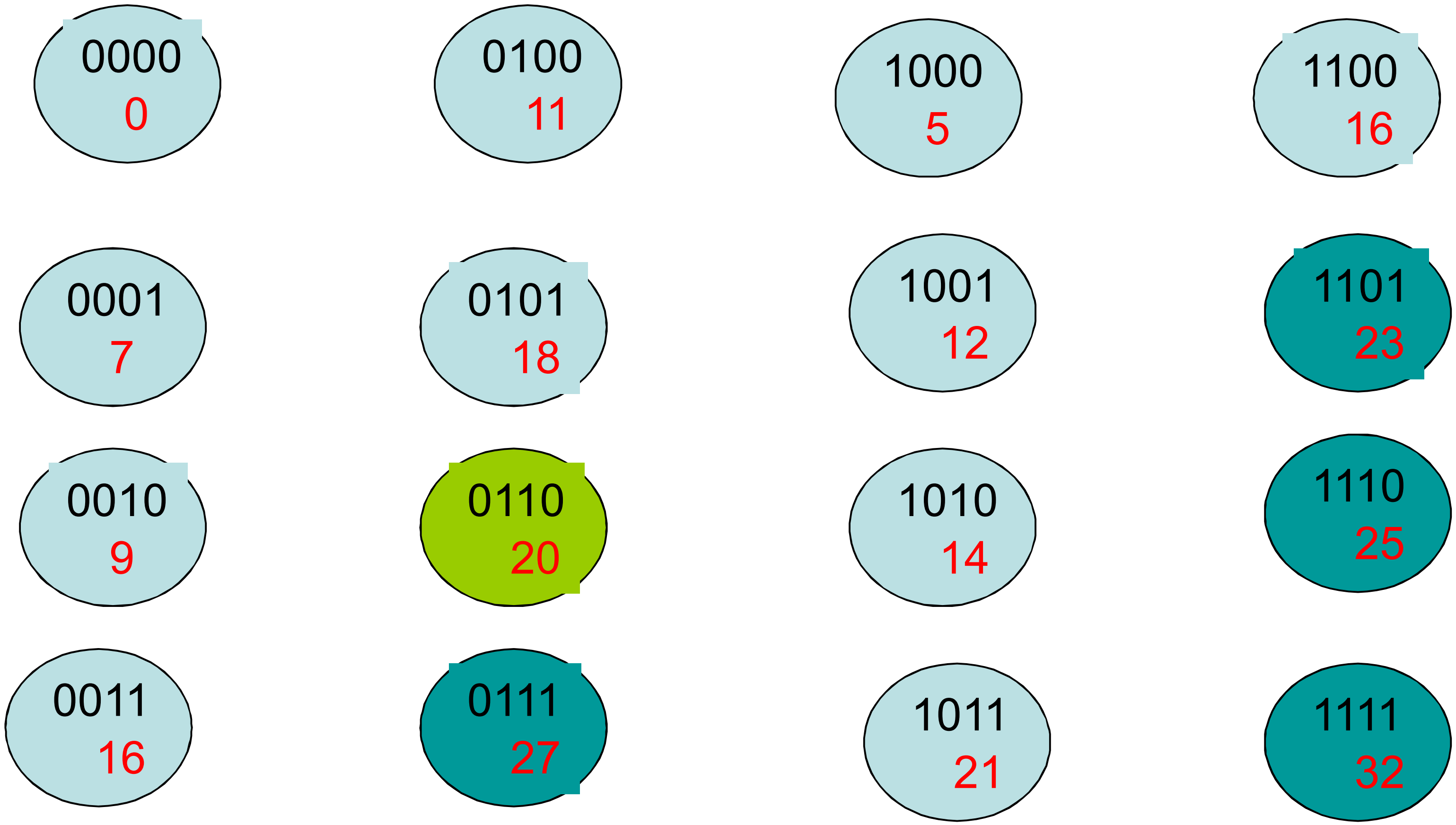
Variables flipped so far: none



Flip Neighborhood

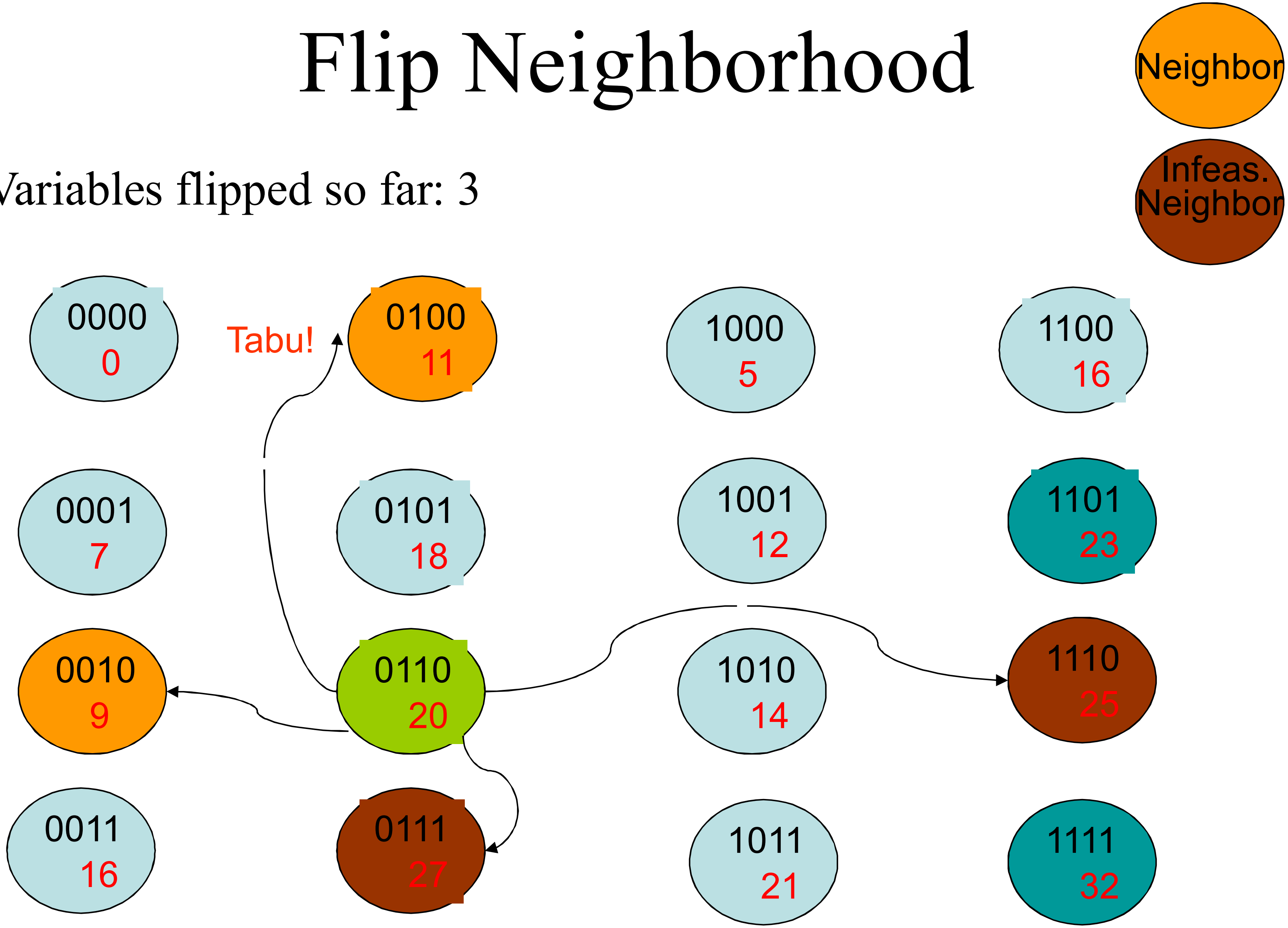


Variables flipped so far: 3

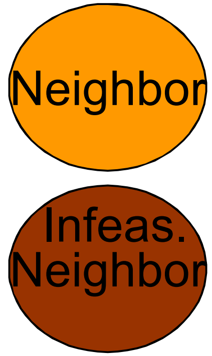


Flip Neighborhood

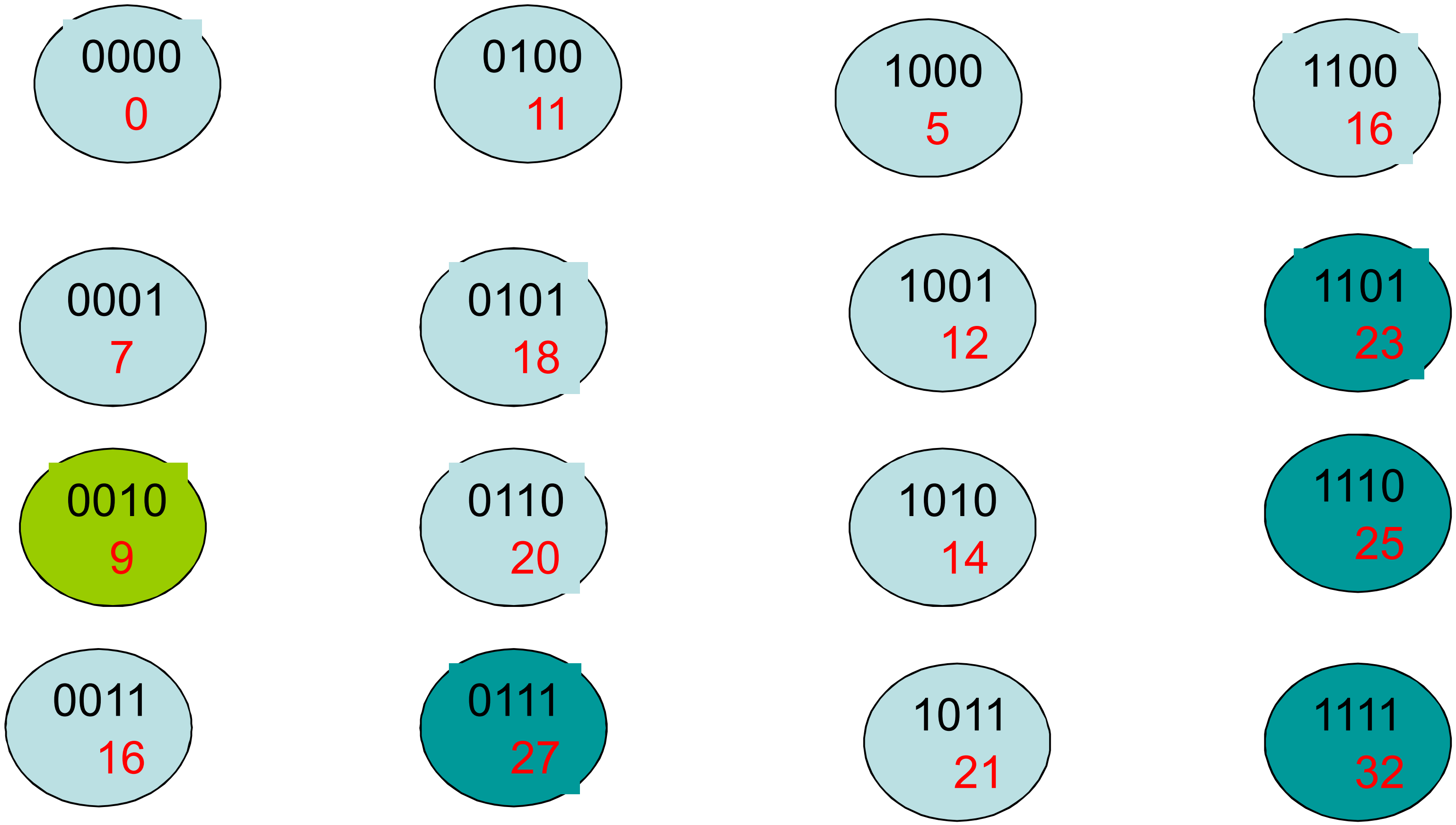
Variables flipped so far: 3



Flip Neighborhood

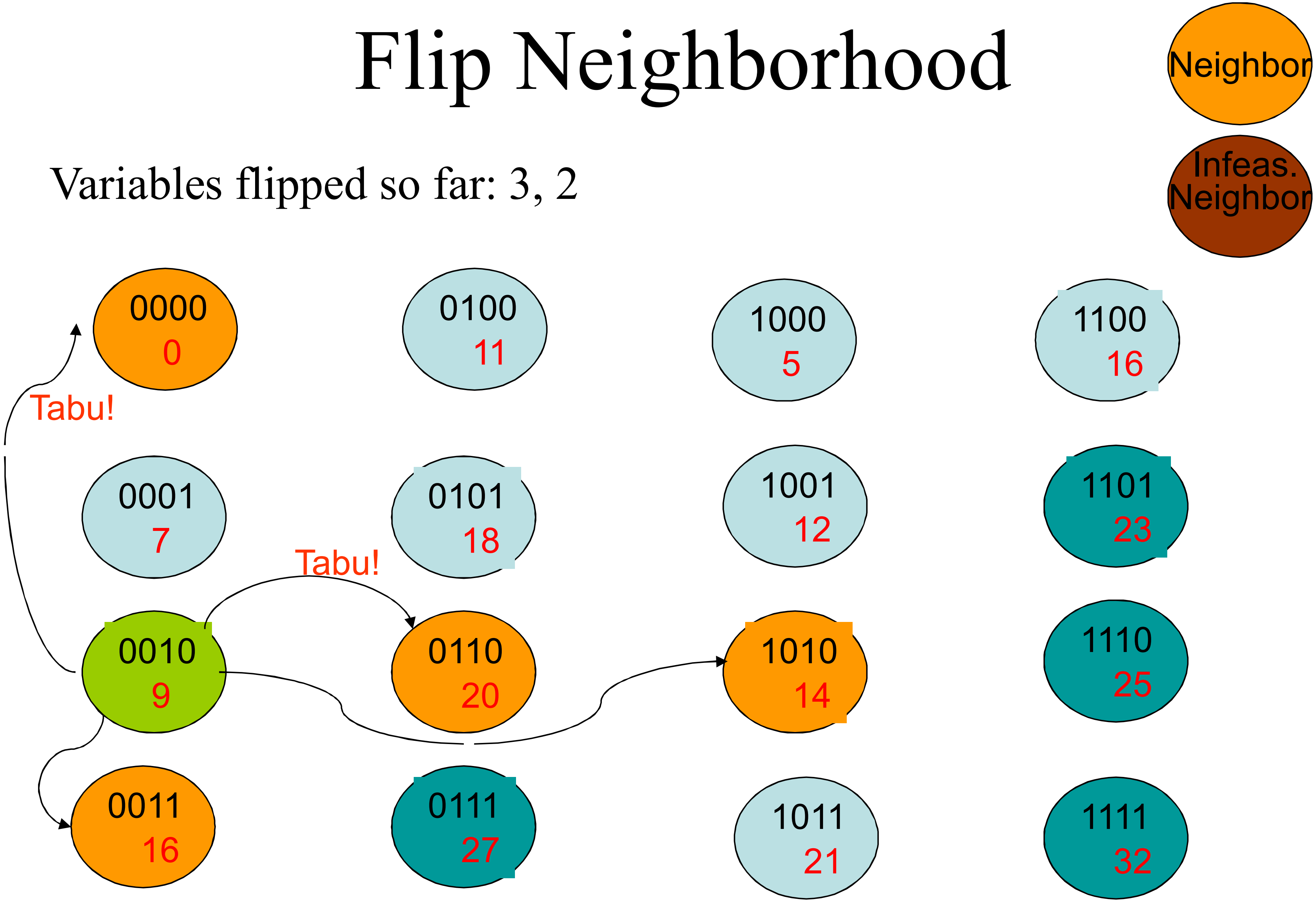


Variables flipped so far: 3, 2

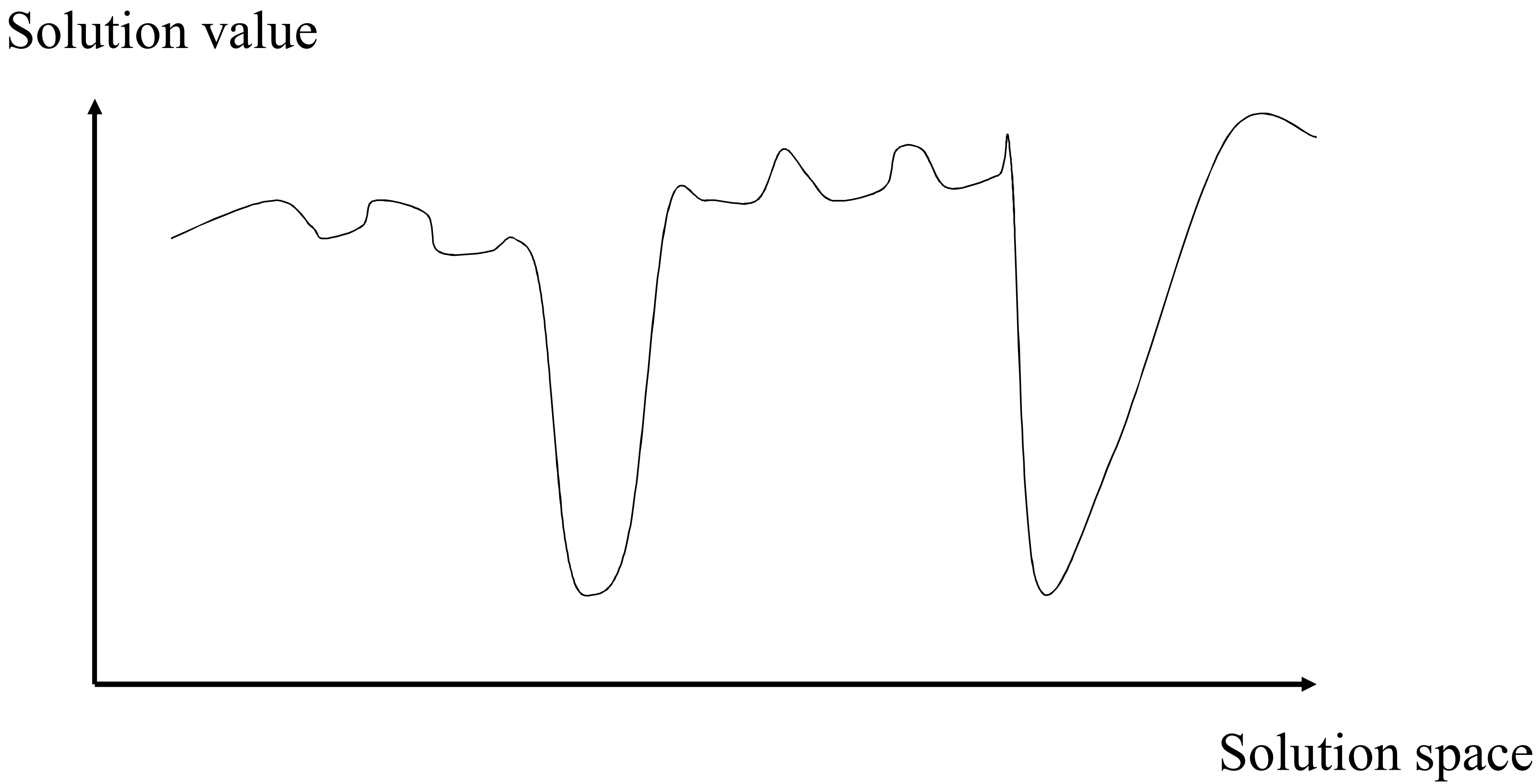


Flip Neighborhood

Variables flipped so far: 3, 2



Local and Global optima



Aspiration Criterion (1)

- The tabu criterion is usually not exact
  - Some solutions that are not visited are nevertheless tabu for some time
- Possible problem: one of the neighbors is very good, but we cannot go there because some attribute is tabu
- Solution: if we somehow know that the solution is not visited before, we can allow ourselves to move there anyway
  - i.e., the solution is a new best solution: obviously we have not visited it before!

Aspiration Criterion (2)

- Simplest: allow new best solutions, otherwise keep tabu status
- Criteria based on
  - Degree of feasibility
  - Degree of change
  - Feasibility level vs. Objective function value
  - Objective function value vs. Feasibility level
  - Distance between solutions
    - E.g. hamming distance
  - Influence of a move
    - The level of structural change in a solution
- If all moves are tabu:
  - Choose the best move, or choose randomly (in the candidate list)

Frequency Based Memory

- Complementary to the short term memory (tabu status)
- Used for long term strategies in the search
- Frequency counters
  - *residency*-based
  - *transition*-based
- TSP-example
  - how often has an edge been in the solution? (*residency*)
  - how often has the edge status been changed? (*transition*)

TS - Diversification

- Basic Tabu Search often gets stuck in one area of the search space
- Diversification is trying to get to somewhere else
- Historically random restarts have been very popular
- Frequency-based diversification tries to be more clever
  - penalize elements of the solution that have appeared in many other solutions visited

TS - Intensification

- To aggressively prioritize good solution attributes in a new solution
- Usually based on frequency
- Can be based on elite solutions, or part of them (vocabularies)

Intensification and Diversification

- Intensification
  - Aggressively prioritize attributes of good solutions in a new solution
    - Short term: based directly on the attributes
    - Longer term: use of elite solutions, or parts of elite solutions (vocabulary building)
- Diversification
  - The active spreading of the search, by actively prioritizing moves that gives solutions with new composition of attributes

Intensification and Diversification - simple mechanisms

- Use of frequency-based memory
- Based on a subset $\boldsymbol{S_f}$ of all the solutions visited (or moves executed)
- Diversification:
  – Choose S_f to contain a large part of the generated solutions (e.g. all the local optima)
- Intensification:
  – Choose S_f to be a small subset of *elite* solutions
    - E.g., that have overlapping attributes
  – Can have several such subset
    - Partitioning, clustering-analysis

Whips and Carrots

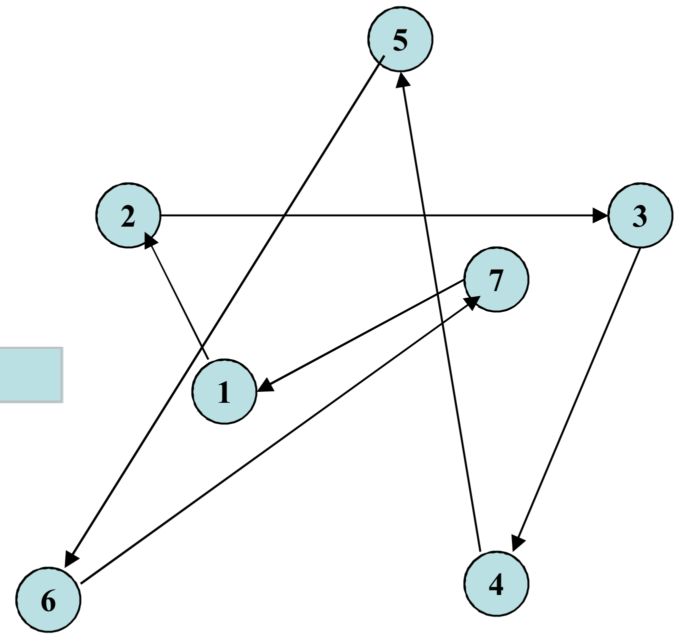
- Used in the move evaluation function, in addition to the change in the objective function value and tabu status
- A carrot for intensification will be a whip for diversification
- Diversification:
  - Moves containing attributes with a high frequency count are penalized
  - TSP-example: $g(x)=f(x)+w_1\Sigma\omega_{ij}$
- Intensification:
  - Moves to solutions containing attributes with a high frequency among the elite solutions are encouraged
  - TSP-example: $g(x)=f(x)-w_2\Sigma\gamma_{ij}$

TS Example: TSP

- Representation: permutation vector
- Move: pairwise exchange

$$(i, j) \quad i < j \quad i, j \in [1, n]$$

1	2	3	4	5	6	7

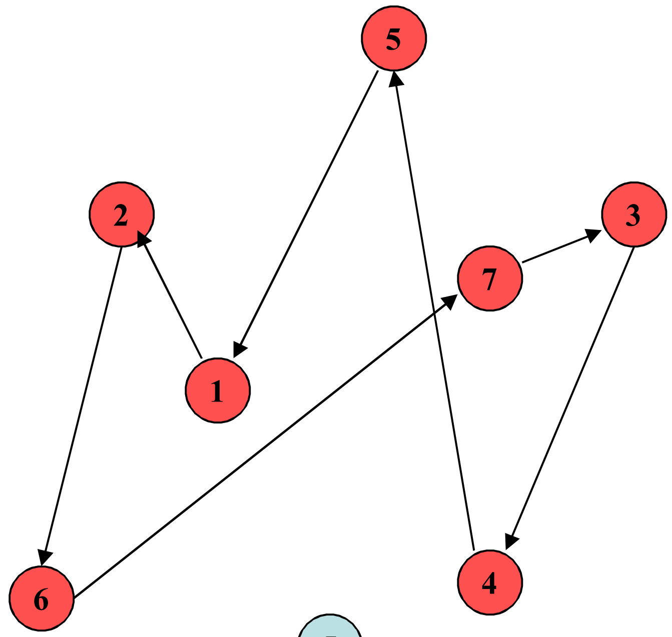


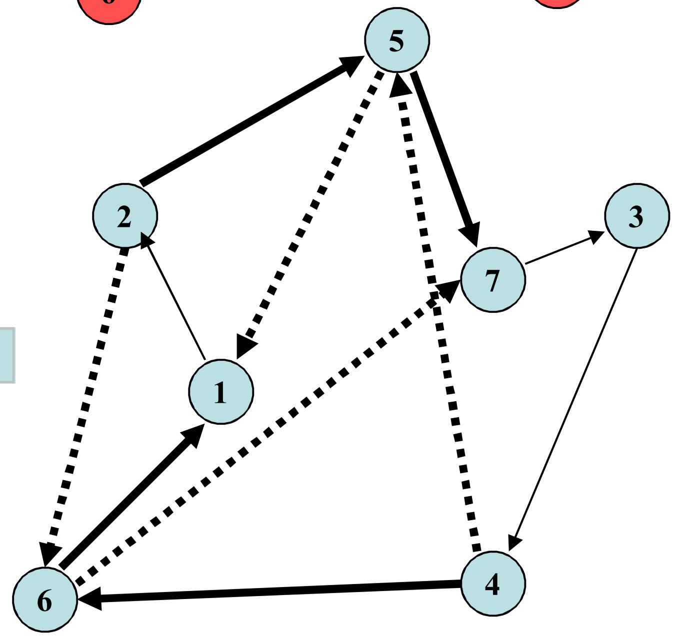
Move: Exchange in permutation vector

2	6	7	3	4	5	1

Move: *Exchange(5,6)*

2	5	7	3	4	6	1





TSP Example

- Number of neighbors: $\binom{n}{2}$
- For every neighbor: *Move value*

$$\Delta_{k+1} = f(i_{k+1}) - f(i_k), \qquad i_{k+1} \in N(i_k)$$

- Choice of tabu criterion
  - Attribute: cities involved in a move
  - Moves involving the same cities are tabu
  - Tabu tenure = 3 (fixed)
- Aspiration criterion
  - new best solution

TSP Example: Data structure

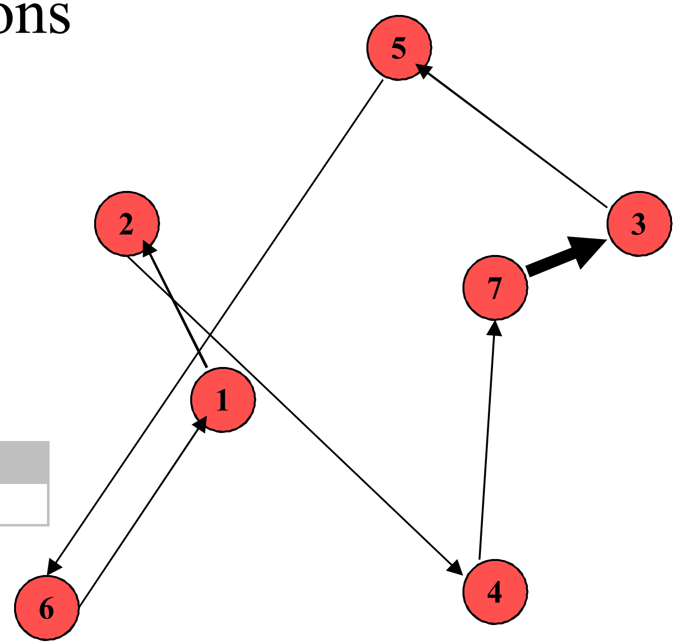
- Data structure: triangular table, storing the number of iterations until moves are legal
- Updated for every move

	2	3	4	5	6	7
1	0	2	0	0	0	0
	2	0	3	0	0	0
		3	0	0	0	0
			4	1	0	0
				5	0	0
					6	0

TSP Example: Tabu Criteria/Attributes

- Illegal to operate on given cities
- Illegal to change the city in position *k* in the vector
- Criteria on edges
  - Links often present in good solutions
  - Length of links w.r.t. the average
- For permutation problems
  - Attributes related to previous/next often work well

1	2	3	4	5	6	7
2	4	7	3	5	6	1



TSP Example: Iteration 0

Starting solution: Value = 234

1	2	3	4	5	6	7
2	5	7	3	4	6	1

Tabu list:

	2	3	4	5	6	7
1	0	0	0	0	0	0
	2	0	0	0	0	0
		3	0	0	0	0
			4	0	0	0
				5	0	0
					6	0

TSP Example: Iteration 1

Current solution: Value = 234

1	2	3	4	5	6	7
2	5	7	3	4	6	1

Candidate list:

Exchange	Value
5.4	-34
7.4	-4
3.6	-2
2.3	0
4.1	4

1	2	3	4	5	6	7
2	4	7	3	5	6	1

After move: Value = 200

Tabu list:

	2	3	4	5	6	7
1	0	0	0	0	0	0
	2	0	0	0	0	0
		3	0	0	0	0
			4	3	0	0
				5	0	0
					6	0

TSP Example: Iteration 2

Current solution: Value = 200

1	2	3	4	5	6	7
2	4	7	3	5	6	1

Candidate list:

Exchange	Value
3.1	-2
2.3	-1
3.6	1
7.1	2
6.1	4

Choose move (3,1)

Tabu list:

	2	3	4	5	6	7
1	0	0	0	0	0	0
	2	0	0	0	0	0
		3	0	0	0	0
			4	3	0	0
				5	0	0
					6	0

TSP Example: Iteration 2

Current solution: Value = 200

1	2	3	4	5	6	7
2	4	7	3	5	6	1

Candidate list:

Exchange	Value
3.1	-2
2.3	-1
3.6	1
7.1	2
6.1	4

Choose move (3,1)

Update tabu list

Tabu list:

	2	3	4	5	6	7
1	0	3	0	0	0	0
2		0	0	0	0	0
3			0	0	0	0
4				2	0	0
5					0	0
6						0

TSP Example: Iteration 3

Current solution: Value = 198

1	2	3	4	5	6	7
2	4	7	1	5	6	3

Candidate list:

Exchange	Value
1.3	2
2.4	4
7.6	6
4.5	7
5.3	9

Tabu!

Choose move (2,4)

Worsening move!

Tabu list:

	2	3	4	5	6	7
1	0	3	0	0	0	0
	2	0	0	0	0	0
		3	0	0	0	0
			4	2	0	0
				5	0	0
					6	0

TSP Example: Iteration 3

Current solution: Value = 198

1	2	3	4	5	6	7
2	4	7	1	5	6	3

Candidate list:

Exchange	Value
1.3	2
2.4	4
7.6	6
4.5	7
5.3	9

Tabu!

Choose move (2,4)

Worsening move!

Tabu list:

	2	3	4	5	6	7
1	0	2	0	0	0	0
	2	0	3	0	0	0
		3	0	0	0	0
			4	1	0	0
				5	0	0
					6	0

Update tabu list

TSP Example: Iteration 4

Current solution: Value = 202

1	2	3	4	5	6	7
4	2	7	1	5	6	3

Candidate list:

Exchange	Value
4,5	-6
5,3	-2
7,1	0
1,3	3
2,6	6

Tabu!

Choose move (4,5)

Aspiration!

Tabu list:

	2	3	4	5	6	7
1	0	2	0	0	0	0
	2	0	3	0	0	0
		3	0	0	0	0
			4	1	0	0
				5	0	0
					6	0

Observations

- In the example 3 out of 21 moves are prohibited
- More restrictive tabu effect can be achieved by
  - Increasing the tabu tenure
  - Using stronger tabu-restrictions
    - Using OR instead of AND for the 2 cities in a move

TSP Example: Frequency Based Long Term Memory

- Typically used to diversify the search
- Can be activated after a period with no improvement
- Often penalize attributes of moves that have been selected often

Tabu-status (closeness in time)

	1	2	3	4	5	6	7	
1			2					
2				3				
3	3							
4	1	5			1			
5		4		4				
6			1		2			
7	4			3				

Frequency of moves

References

- Book
  - Modern heuristic techniques for combinatorial problems/ by Colin R. Reeves
- Papers:
  - The theory and practice of simulated annealing/ by Darrall Henderson, Sheldon H. Jacobson, and Alan W. Johnson
  - An introduction to Tabu Search/ by Michel Gendreau