U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Artificial Intelligence

Lecture 3b: Meta-Heuristics – Simulated Annealing and Tabu Search

(based on Løkketangen, 2019)

Luís Paulo Reis

lpreis@fe.up.pt

Director of LIACC – Artificial Intelligence and Computer Science Lab.
Associate Professor at DEI/FEUP – Informatics Engineering Department,
Faculty of Engineering of the University of Porto, Portugal
President of APPIA – Portuguese Association for Artificial Intelligence

Agenda

- Simulated Annealing
- Iterative Local Search
- Variable Neighbourhood Search
- Guided Local Search
- Tabu Search
- Summary and Conclusions

Simulated Annealing (Arrefecimento Simulado)

- A metaheuristic inspired by statistical thermodynamics
  - Based on an analogy with the cooling of material in a heat bath
- Used in optimization for over 30 years
- Very simple to implement
- A lot of literature
- Converges to the global optimum under weak assumptions (- usually slowly)
- Simulated annealing can be used for hard computational optimization problems where exact algorithms fail
- Usually only achieves an approximate solution to the global optima but it could be enough for many practical problems

Simulated Annealing

- **Name** of the algorithm comes from **annealing in metallurgy**, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects
- Both are attributes of the material that depend on their thermodynamic free energy
- Heating and cooling the material affects both the temperature and the thermodynamic free energy

Metropolis' Algorithm (1953)

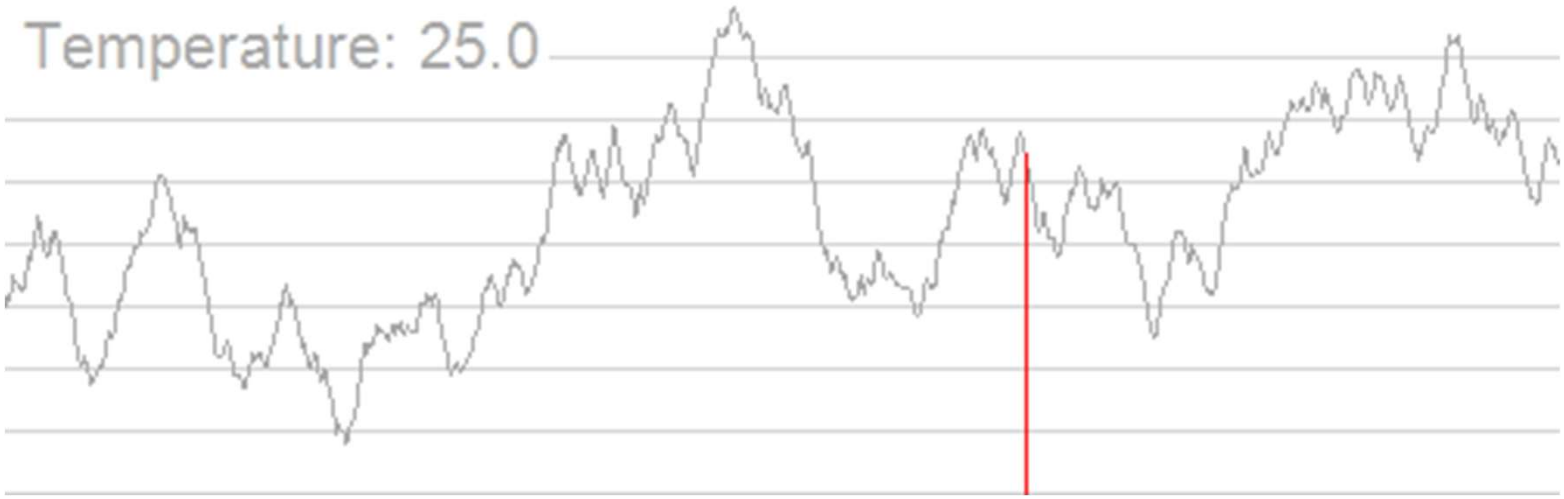
Algorithm to simulate energy changes in physical systems when cooling

Kirkpatrick, Gelatt and Vecchi (1983)

Suggested to use the same type of simulation to look for solutions in a COP

Simulated Annealing

- The state of some physical systems, and the function E(s) to be minimized, is analogous to the internal energy of the system in that state.
- The goal is to bring the system, from an arbitrary initial state, to a state with the minimum possible energy.

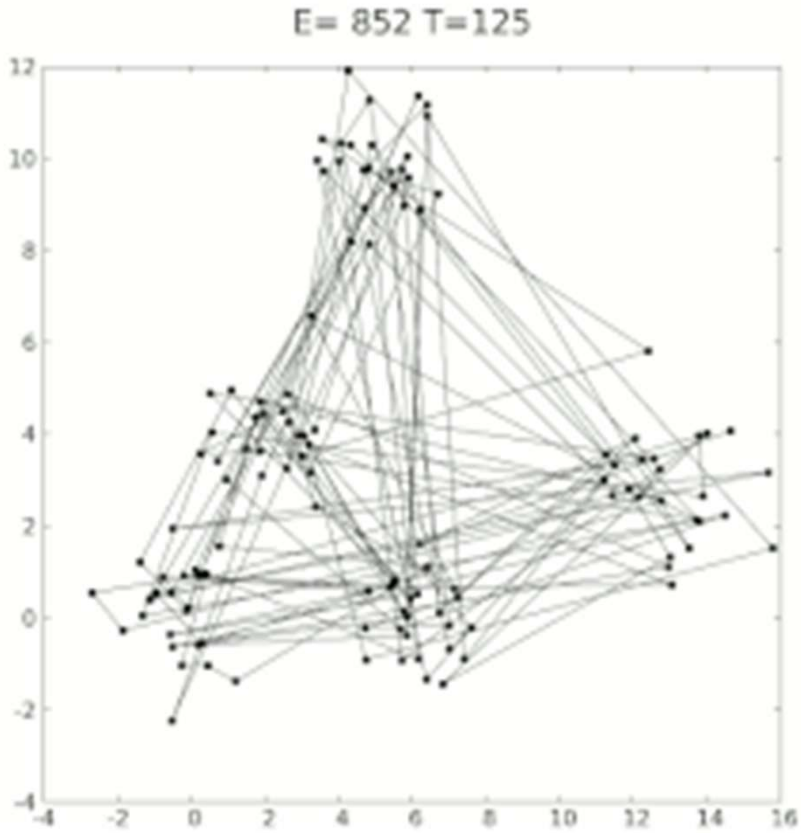


Simulated Annealing

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



TSP (125 points) Solved with SA

Based on Hill-Climbing (Subir a Colina)

- **Basic "Hill climbing":** Generates one by one the successors of the current state and if founds one better evaluated than the Current State, selects it and applies it
- **Basic "Hill climbing" (random):** Generates a random successor of the current state and if it is better evaluated than the Current State, selects it and applies it
- **"Steepest ascent":** Generates all possible successors and selects the best evaluated one

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

Simulated Annealing – Basic Algorithm

- **Basic Algorithm for Simulated Annealing:**
  - Start with an initial solution (random or other method)
  - For a given number of cycles or until achieving a given stopping criteria
  - Change the temperature, reducing it following the schedule

$s = s_0$; $T = T_0$
for $Iter = 0$ to $Iter_{max}$
    $T \leftarrow$ temp_sched(T)
    $s_{new} \leftarrow$ random_neighb(s)
    if $Prob(Eval(s), Eval(s_{new}), T) \geq$ rand(0, 1)
    then $s \leftarrow s_{new}$

  - Calculate a possible new solution neighbour from the current one
  - If the probability of accepting the new solution given its performance change from the current one and the current temperature is greater or equal to a random value between o and 1, accept the new solution as the current solution

Choice of Move in Simulated Annealing

- **Modified ”Random Descent”**
- **Select a random solution in the neighbourhood**
- **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**

Move Acceptance in Simulated Annealing

Assuming a maximization problem:

- Set: Δ = Eval(random neighbour) – Eval(current solution)
- If Δ > 0 → accept (we have an improving neighbour)
- Else accept it with probability $e^{\frac{\Delta}{t}}$
- If the move is not accepted: Try another random neighbor

```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T = 0 then return current
       next ← a randomly selected successor of current
       ΔE ← VALUE[next] – VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(Δ E/T)
```

Simulated Annealing - Structure

- **Initial temperature t_0 high**
  - (if $\infty \rightarrow$ random walk)
- **Reduce temperature *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 very important**

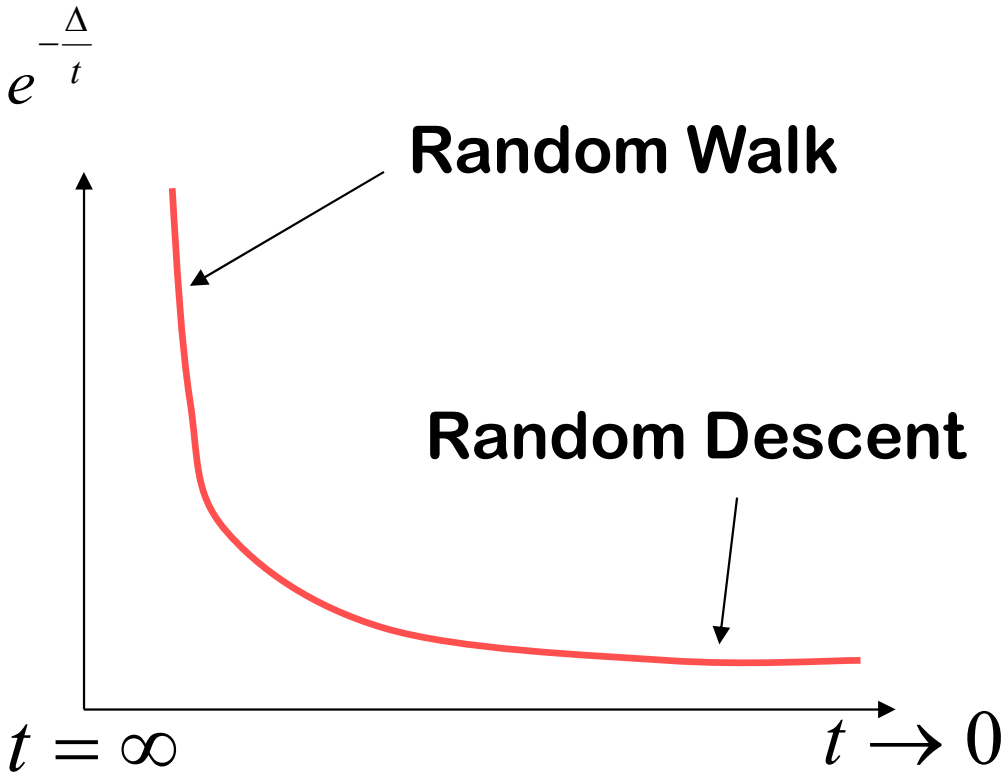
Simulated Anealling – Overall Structure

- 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

Simulated Annealing – Cooling Schedule

Cooling schedule is vitally important

- Much research on this
- Static schedules:
  - specified in advance
- Adaptive schedules:
  - react to information from the search
- Depending on the cooling schedule it may be good to perform some iterations at low (or zero) temperature to converge to local optima although not necessary if algorithm well applied



Simulated Annealing in Practice

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

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**

Problem Specific Decisons

- **Important goals**
  - Time to get the solution
  - 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

Choice of Neighbourhoods

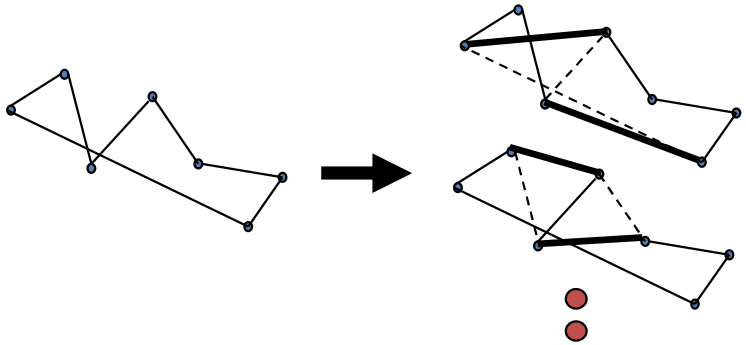
- **Size**
- **Variation in size**
- **Topologi**
  - Symmetry
  - Connectivity
    - Every solution can be reached from all the others
- **Topography**
  - Spikes, Plateaus, Deep local optima
- **Move evaluation function**
  - How expensive is it to calculate ?

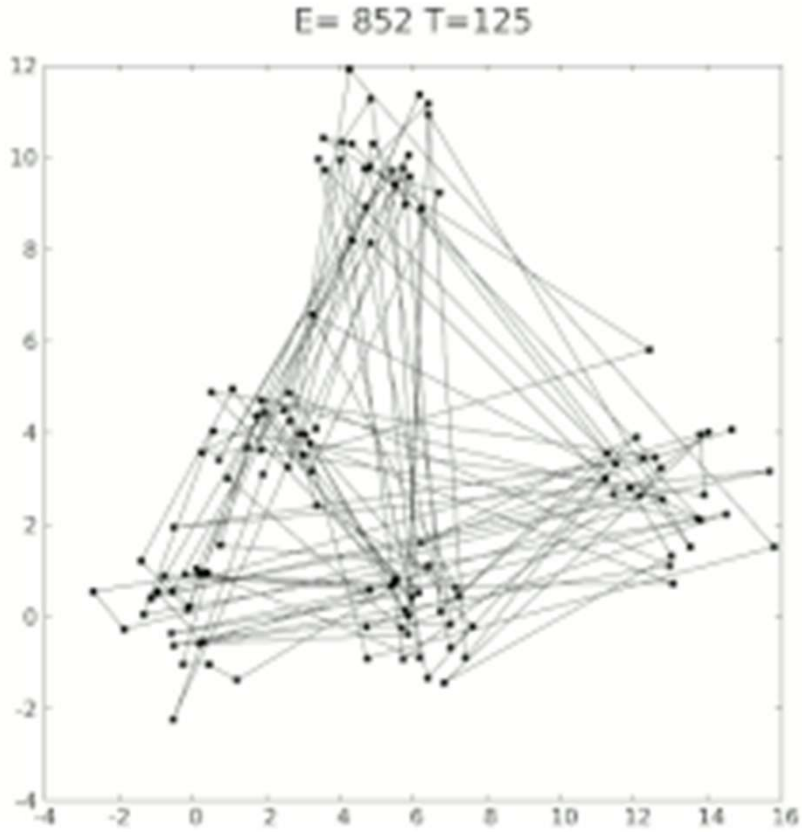
Simulated Annealing – Speed

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

Simulated Annealing – 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





Simulated Annealing – 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**

Simulated Annealing – Summary

- **Inspired by statistical mechanics - cooling**
- **Metaheuristic**
  - Local search
  - Random descent
  - Use randomness to escape local optima
- **Simple and robust method**
  - Easy to get started
- **In practice:**
  - Computationally expensive
  - Needs well devised neighboorhoods and cooling schedules
  - Fine tuning can give good results
  - SA can be good where robust heuristics based on problem structure are difficult to make

Other LS-based Metaheuristics

- **Main metaheuristics:**
  - Simulated Annealing
  - Tabu Search
- **Some other LS-based methods:**
  - Threshold Accepting (variation of SA)
  - Generalized Hill-Climbing Algorithm (generalization of SA)
  - Iterated Local Search (better than random restarts)
  - Variable Neighborhood Search (using a set of neighborhoods)
  - Guided Local Search (closer to the idea of Tabu Search)

Local Search - Restarts

- **Given a Local Search procedure:**
  - After the algorithm stops (in or near a local optimum), what to do then?
- **Restarts**
  - Repeat (iterate) the same procedure over and over again with different starting solutions
- **If the search is deterministic (no randomization), no interest to restart**
- **Change:**
  - Starting solution
  - Random neighbor selection
  - Controlling parameters (e.g., the temperature)
- **Restarting can lead to a different (possibly better) solution**

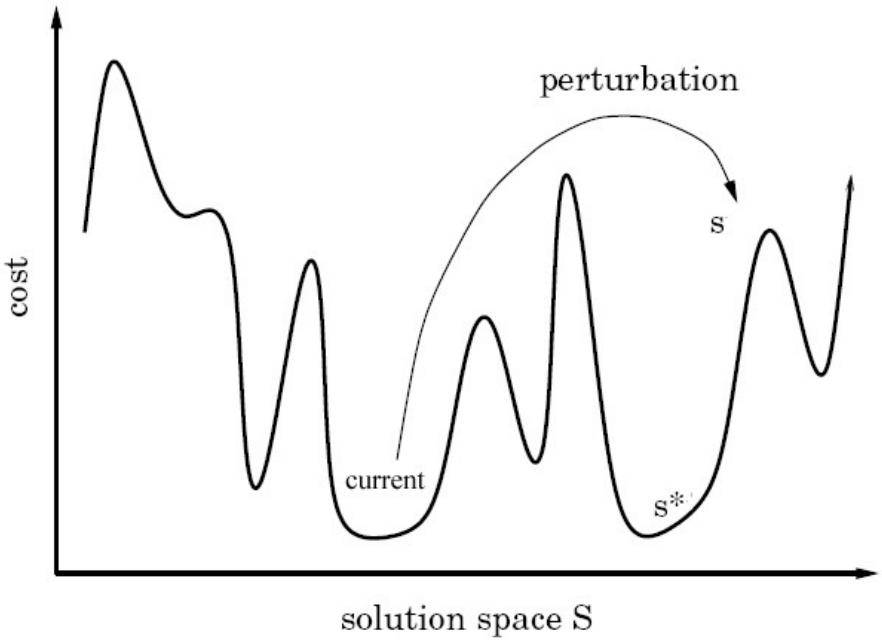
Iterated Local Search

- **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**
- **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)
  - 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

Iterated Local Search

```
1: input: starting solution, s_0
2: input: Local Search procedure, LS
3: current ⇐ LS(s_0)
4: while stopping criterion not met do
5:    s ⇐ perturbation of current based on search history
6:    s* ⇐ LS(s)
7:    if s* is accepted as the new current solution then
8:       current ⇐ s*
9:    end if
10: end while
```



- **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)

Iterated Local Search

- **Initial Solution: Random solution or 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: Random move in a ”higher order neighborhood”**
  - If returning to the same solution (*s*=current*), then increase the strength of the perturbation?
  - Strength of the perturbation is important (may vary at run-time):
    - Too strong: close to random restart
    - Too weak: Local Search may undo perturbation
  - Perturbation should be complementary to the Local Search
- **Acceptance:**
  - Accept s* only if f(s*)<f(current) – better local optimum:
    - Extreme intensification / Random Descent in space of local optima
  - Accept s* always:
    - Extreme diversification / Random Walk in space of local optima
  - Intermediate choices possible

ILS and Metaheuristics

- **Iterated Local Search makes it 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”

Variable Neighborhood Search

- **Based on the ideas:**
  - Local minimum w.r.t. one neighborhood structure is not necessarily locally minimal w.r.t. another neighborhood structure
  - A global minimum is locally optimal w.r.t. *all* neighborhood structures
  - For many problems, local minima with respect to one or several neighborhoods are relatively close to each other
- **Basic principle: change the neighborhood during the search:**
  - Variable Neighborhood Descent
  - Basic Variable Neighborhood Search
  - Reduced Variable Neighborhood Search
  - Variable Neighborhood Decomposition Search

Variable Neighborhood Search

- Generate different neighborhood structures
  - E.g., for the VRP: 2-Opt, Cross, Swap, Exchange,…
- Find neighborhoods that depend on some parameter
  - k-Opt (k=2,3,…)
  - Flip-neighborhoods can be extended to double-flip, triple-flip, etc.
- Some neighborhoods are associated with distance measures: increase the distance

Variable Neighborhood Descent

```
1: input: starting solution, s_0
2: input: neighborhood operators, {N_k}, k = 1,...,k_max
3: input: evaluation function, f
4: current ⇐ s_0
5: k ⇐ 1;
6: while k ≤ k_max do
7:    s ⇐ the best neighbor in N_k(current)
8:    if f(s) < f(current) then
9:       current ⇐ s
10:      k ⇐ 1
11:   else
12:      k ⇐ k + 1
13:   end if
14: end while
```

- Use neighborhood structures N_1, N_2,…, $N_{k\text{-max}}$
- Standard (”Best Improvement”) LS applied in N_1
- Other neighborhoods are explored only randomly
- Exploration of the other neighborhoods are perturbations as in ILS

Guided Local Search

- **A general strategy, metaheuristic, used to guide a neighborhood search**
- **Tries to overcome local optima by ”removing” them:**
  - Changes the ”topography” of the search space
  - Uses an extended move evaluation function:

    Evaluation function = Original objective function + Penalties
- **Focuses on promising parts of the search space**
- **GLS assumes that we can find some features of a solution that we can penalize**
- **Features are:**
  - Something that is characteristic for the solution
  - Something that might be different in another solution
  - Problem dependent
- **Examples of Features:**
  - TSP: whether city A follows immediately after city B in the tour or not
  - Knapsack: whether item 5 is included or not

Features & Extended Move Evaluation

- **Modification of the move evaluation is based on features**
- **Features have a cost:**
  - Represents (directly or indirectly) the influence of a solution on the (extended) move evaluation function
  - Constant or variable (dependent on other features)
  - GLS tries to avoid costly features
- **We use an indicator function as follows:**
  - $I_i(s)$ = 1 if solution *s* has feature *i (0 otherwise)*
- **Let the set of features be denoted by F = {1,…,G}**
- **Penalty vector p=$[p_i]$, *i*=1…*G***
  - p_i is the number of times feature *i* have been penalized until now
- **The extended move evaluation function becomes**

$$f^*(s) = f(s) + \lambda \sum_{i=1}^{G} I_i(s) p_i$$

parameter λ adjusts the influence of the penalties

GLS - Penalties

- **Penalties are initially equal to 0**
- **When the search has reached a local optimum (with respect to the extended move evaluation function):**
  - The penalty is increased for some of the features of the current (locally optimal) solution
  - This makes current solution worse in comparison to some neighboring solutions (which do not have the penalized features)

$$f^*(s) = f(s) + \lambda \sum_{i=1}^{G} I_i(s) p_i$$

- **To select which feature to penalize, define the *utility* of a feature *i* in solution *s* as follows:**
  - $u_i(s) = I_i(s) * c_i / (1+p_i)$ (c_i is the cost of the feature, in objective function and p_i is the previous penalty
- **In a local optimum, *s*, increase the penalty for the feature that has the highest utility value, $u_i(s)$**
- **Penalties are only adjusted when the search has reached a local optimum, and only for features included in the local optimum**

Guided Local Search - Pseudocode

Guided Local Search

```
1: input: starting solution, s_0
2: input: neighborhood operator, N
3: input: evaluation function, f
4: input: a set of features, F
5: input: a penalty factor, λ
6: current ⇐ s_0
7: best ⇐ s_0
8: p_i ⇐ 0 (for all i ∈ F)
9: while stopping criterion not met do
10:    Define f*(s) = f(s) + λ Σ_{i∈F} I_i(s) p_i
11:    s* ⇐ the best solution in N(current), according to f*
12:    if f*(s*) < f*(current) then
13:       current ⇐ s*
14:       if f(current) < f(best) then
15:          best ⇐ current
16:       end if
17:    else
18:       Define the utility, u_i(current) = I_i(current) c_i/(1+p_i), for all i ∈ F
19:       p_i ⇐ p_i + 1 for each feature i ∈ F having the maximum utility in
          solution current
20:    end if
21: end while
```

GLS versus SA and TS

- **In SA is difficult to find good cooling schedule (problem dependent)**
  - High temperature gives bad solutions
  - Low temperature gives convergence to a local minimum
  - SA is non-deterministic
- **GLS visits local minima, but can escape**
  - Not random up-hill moves as in SA
  - GLS is deterministic
  - Penalties are added until the search escapes from local minimum
- **GLS is closely related to Tabu Search since both have mechanisms to guide the Local Search away from local optima**
  - GLS penalizes features in the solutions
  - TS bans (makes taboo) features in the solutions
- **Both incorporate memory structures**
  - GLS has the accumulated penalties
  - TS has different memory structures: Short term, long term, frequency, recency, …

ILS, VNS and GLS

- **ILS and VNS are based on different underlying ”philosophies”**
  - ILS: Perturb and do Local Search
  - VNS: Exploit different neighborhoods
- **GLS**
  - Idea: Remove local optima by changing the evaluation of solutions
  - Has some similarities with Tabu Search
- **Based on simple principles**
- **Easy to understand**
- **Basic versions are easy to implement**
- **Robust**
- **Highly effective**

Tabu Search

- **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
  - *"Loaded with a dangerous, unnatural force"*
  - *"Banned due to moral, taste or risk"*
- **Tabu Search (Fred Glover 1986):**
  - 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

Tabu Search – ideas and Pseudocode

- **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)**

Tabu Search

1: $current \Leftarrow$ a starting solution
2: Initialize tabu memory
3: **while** stopping criterion not met **do**
4: Find a list of candidate moves, a subset of $N(current)$
5: Select the solution, s, in the candidate list that minimizes an extended cost function
6: Update tabu memory and perform the move: $current \Leftarrow s$
7: **end while**

Tabu Search - Critical Choices

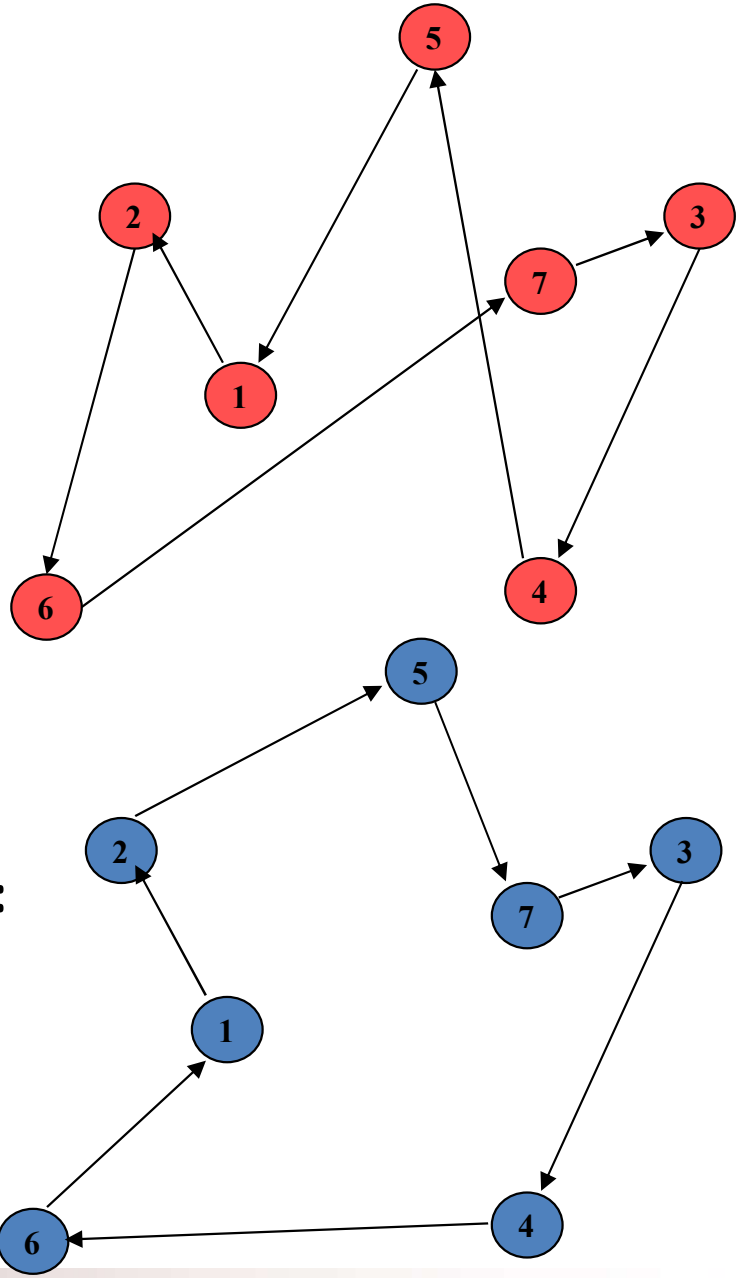
- **Choice of neighborhood, *N***
- **Definition of the tabu memory**
- **How to select the candidate list**
- **Definition of the evaluation function**
  - Improvement in solution values
  - Tabu criteria
  - Aspiration criteria
  - Long term strategies: Diversification, Intensification, …
- **Basic Tabu Search:**
  - Local Search with "Best Improvement" strategy (always select the best move)
  - 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

Tabu Criterion

- **In Tabu Search, we allow moving to a worse solution**
- **Since we (in basic TS) always select the ”Best Improvement”, how to 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!
- **Job of the tabu criterion is to avoid visiting again the same solution**
- **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/memory 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: Property of a solution or a move**
- **Can be based on any aspect of the solution that is changed by a move**
- **Attributes are the basis for tabu constraints**
  - Used to represent the solutions visited recently
- **Move can change more than one attribute**
  - e.g. a 2-opt move in TSP involves 4 cities and 4 edges
- **Similar to "Features" in GLS, but TS don't require attributes to have costs**
- **Attributes based on the edges**
  - A1: Edges added to the tour
  - A2: Edges removed from the tour
- **TSP Move example of type: Exchange two cities:**
  - 4 edges removed and 4 edges added
  - Exchange(5,6):
    - A1: (2,5),(5,7),(4,6),(6,1)
    - A2: (2,6),(6,7),(4,5),(5,1)



Tabu Criterion

- **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**
- **Attribute is tabu for a certain amount of time/iterations**
  - *Tabu Tenure* (TT)
- **Tabu criterion usually avoids immediate move reversal (or repetition)**
- **It also avoids the other (later) moves containing the tabu attribute.**
- **Cuts off a much larger part of the search space**
- **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 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: $x_{2,5}$

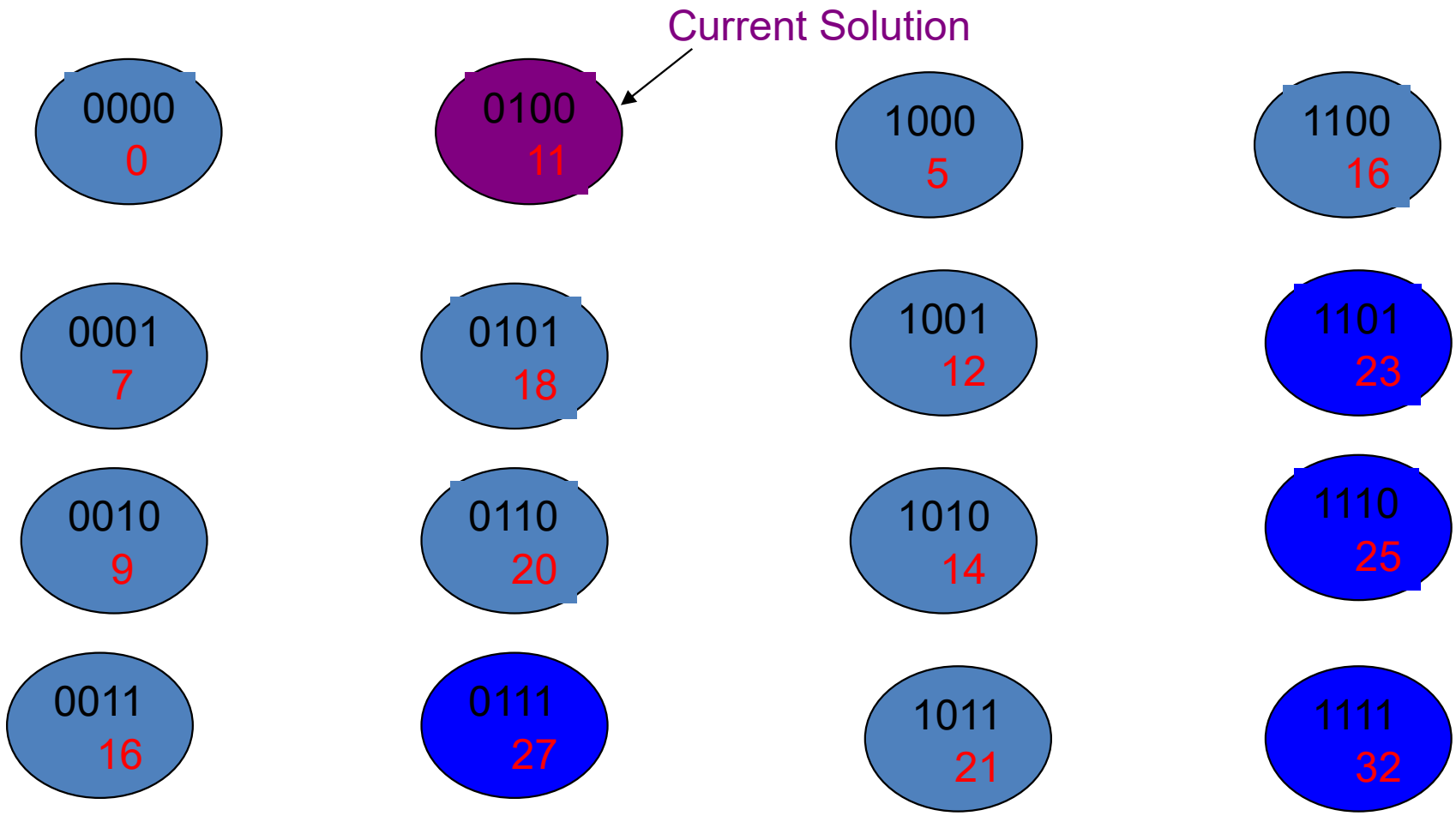
  The tabu criterion could disallow:
  - Moves involving 2 and 5
  - Moves involving 2 or 5
  - Moves involving 2
  - Moves involving 5

Tabu Tenure

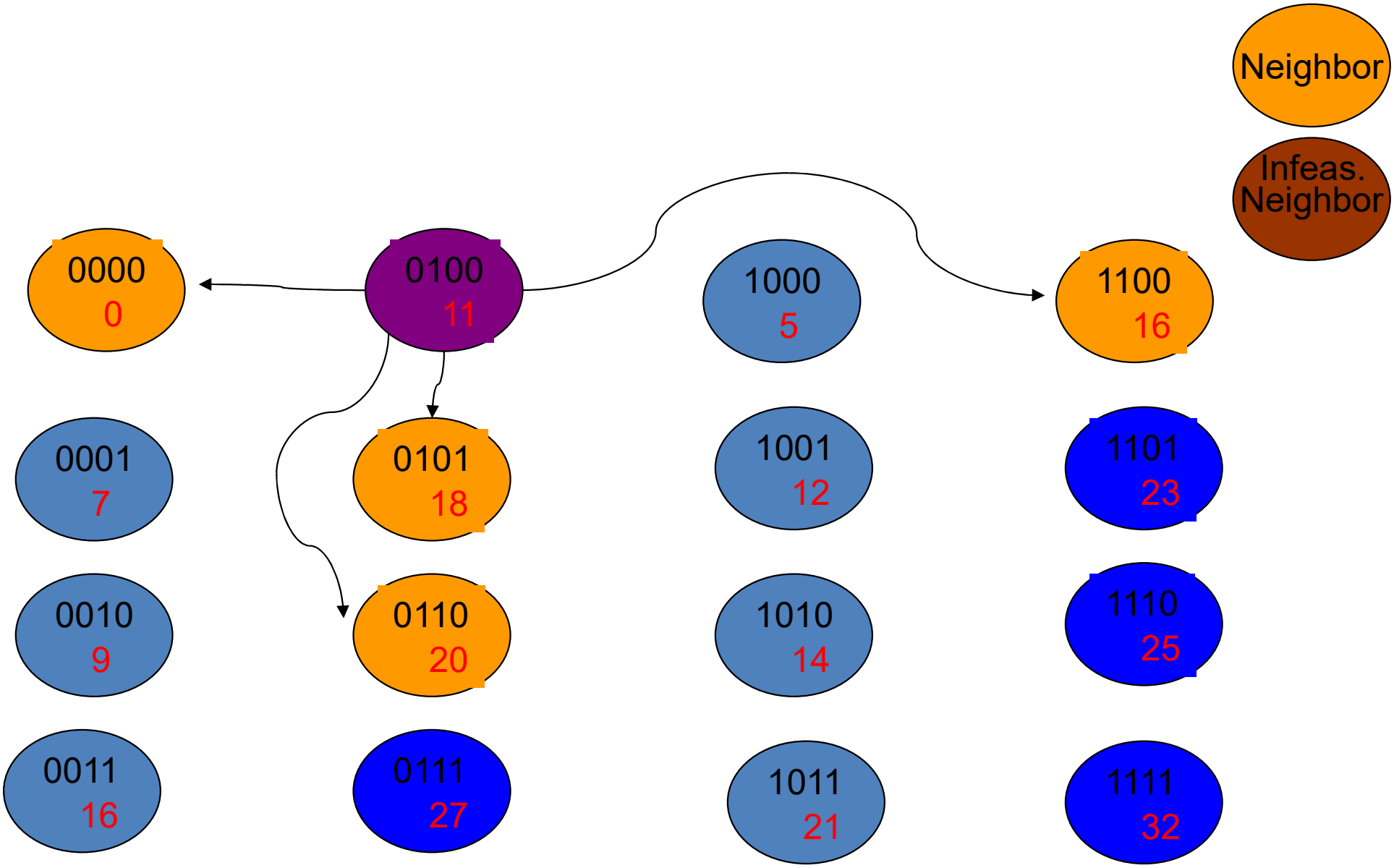
- **The tabu criterion will disallow moves that change back the value of some attribute(s)**
- **For how long do we need to enforce this rule?**
  - For ever: the search stops because no changes are allowed
  - For too long: the search might become too limited (too much of the search space is cut off due to the tabu criterion)
  - For too short: the search will still cycle, but the length of the cycle can be more than 2
- **The number of iterations for which the value of the attribute remains tabu is called the *Tabu Tenure***
  - Earlier: The magical number 7, plus or minus 2
  - Sometimes: in relation to problem size: $n^{1/2}$
  - Static (fixed): not recommended (search gets more easily stuck in loops)
- **Dynamic tabu tenure is highly recommended**
- **Dependent on the Tabu attributes**
- **Reactive Tabu Search**
  - Detect stagnation $\rightarrow$ increase TT
  - When escaped $\rightarrow$ reduce TT

Example: Knapsack/Flip Neighborhood

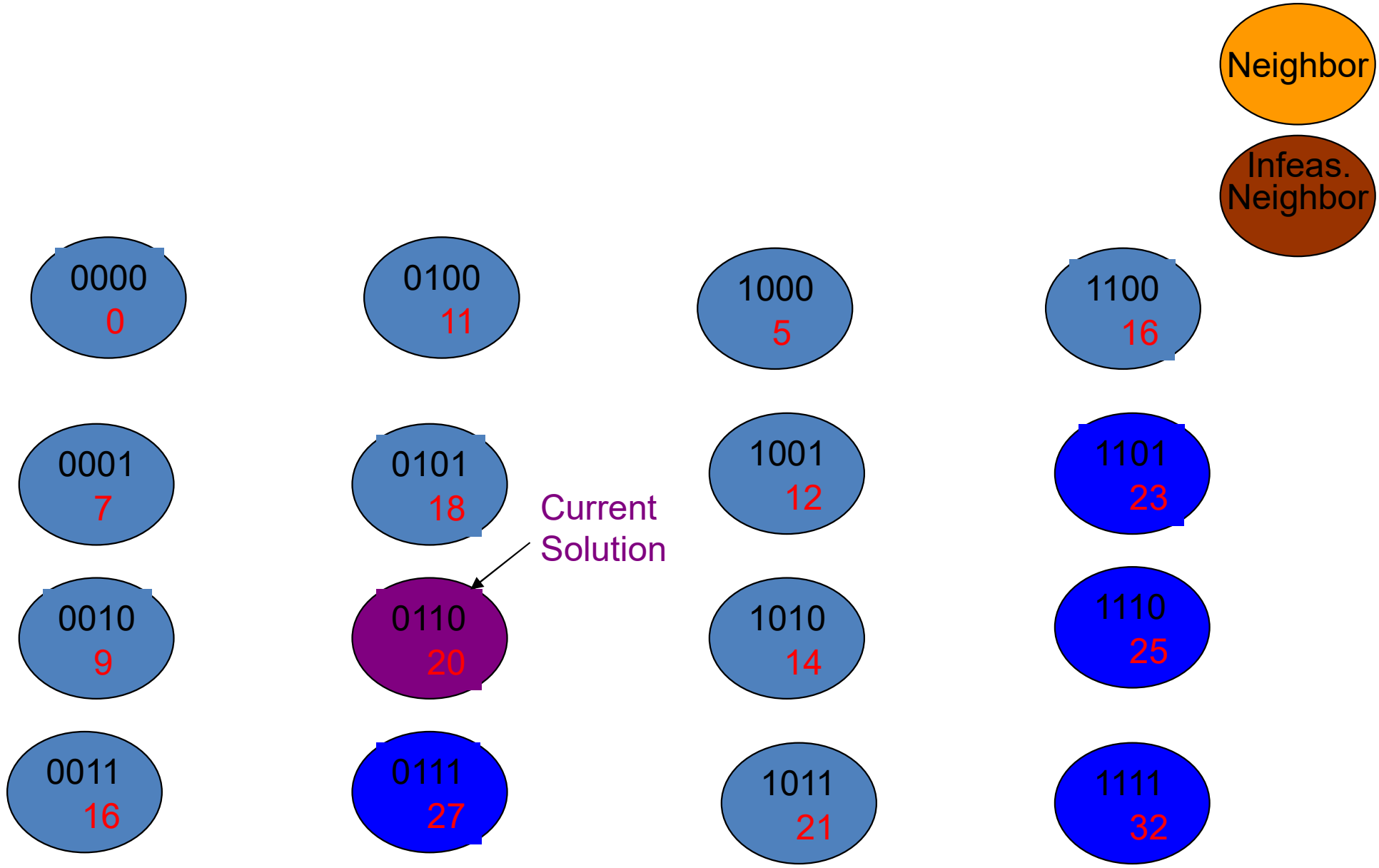
- **If the move is selecting an item to include in the solution, then any move trying to remove the same item is *tabu* for the duration of the *tabu tenure***
- **An item thrown out is not allowed in for the duration of the tabu tenure Here the attribute is the same as the whole move**



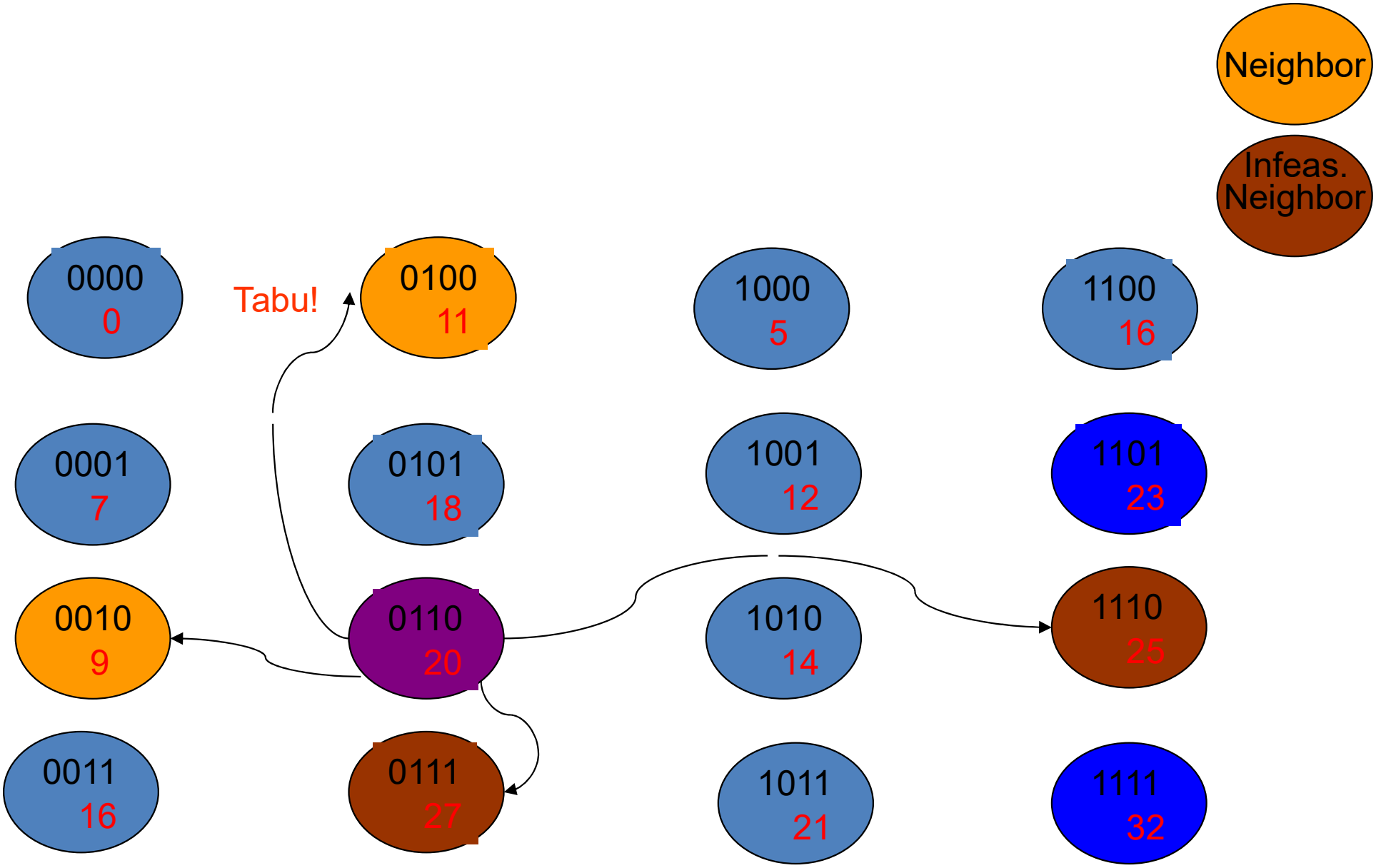
Example: Knapsack/Flip Neighborhood



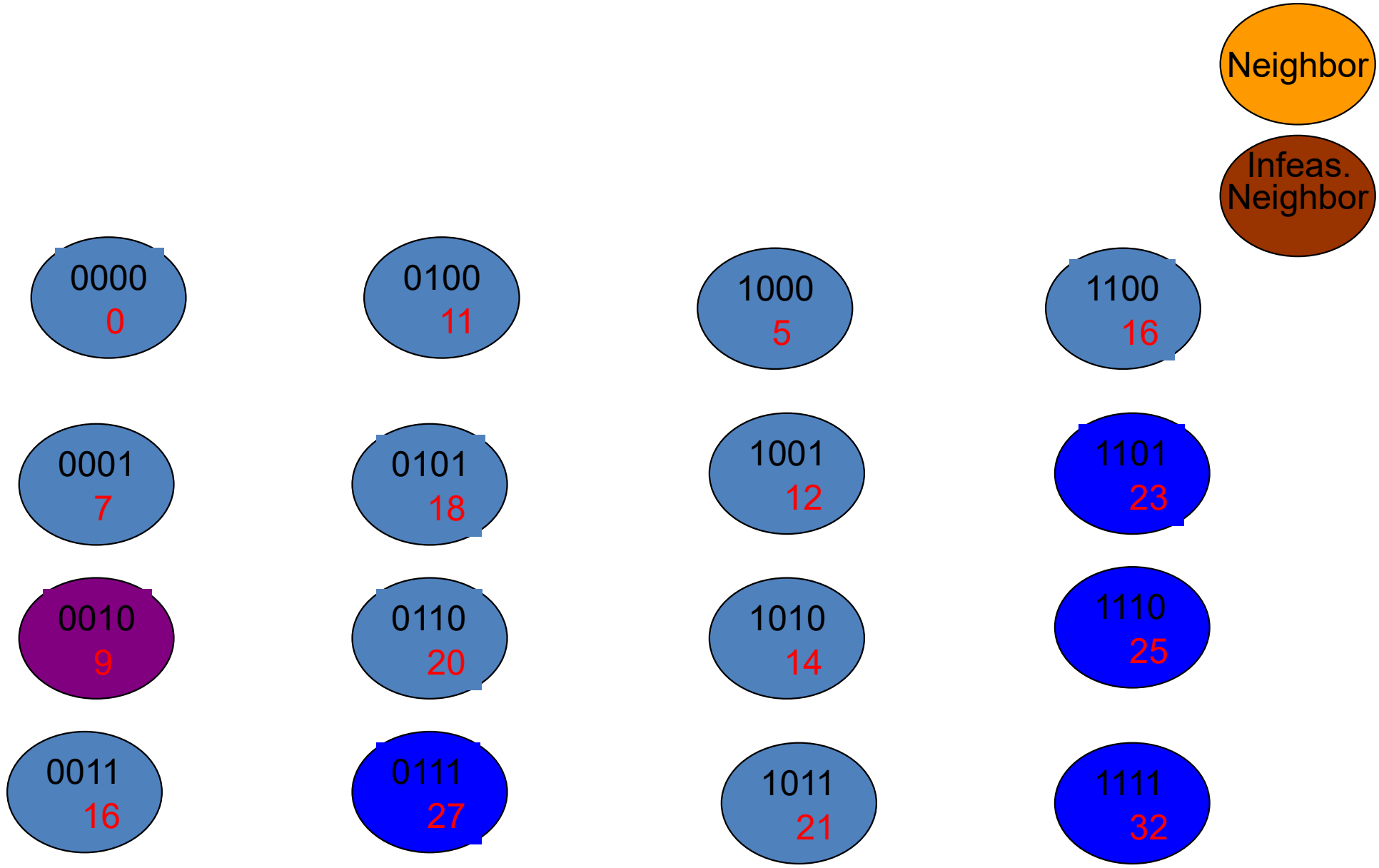
Example: Knapsack/Flip Neighborhood



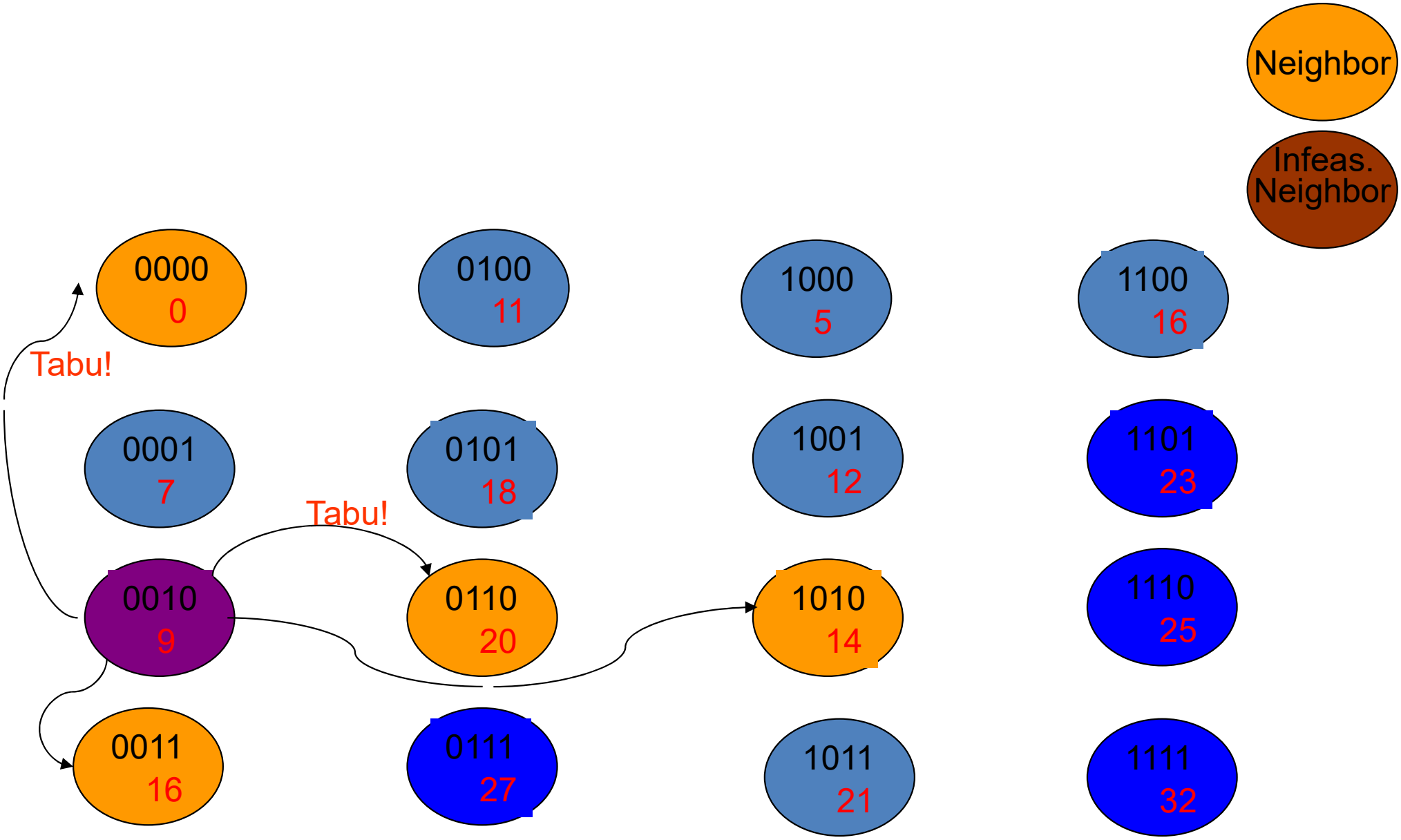
Example: Knapsack/Flip Neighborhood



Example: Knapsack/Flip Neighborhood



Example: Knapsack/Flip Neighborhood



Aspiration Criterion

- **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

- **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
  - Influence of a move: Level of structural change in a solution
- **If all moves are tabu:**
  - Choose the best move, or choose randomly (in 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/ Intensification

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
- **Active spreading of the search, by actively prioritizing moves that gives solutions with new composition of attributes**

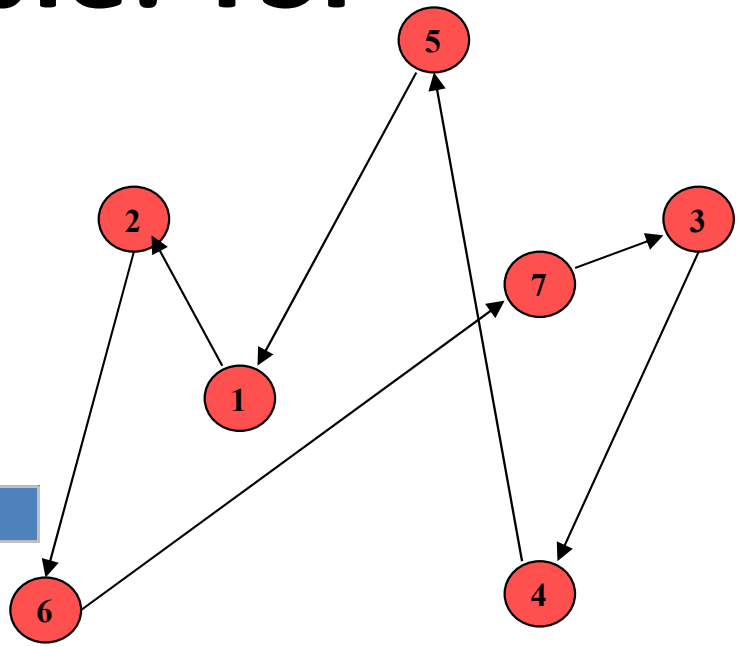
Intensification:

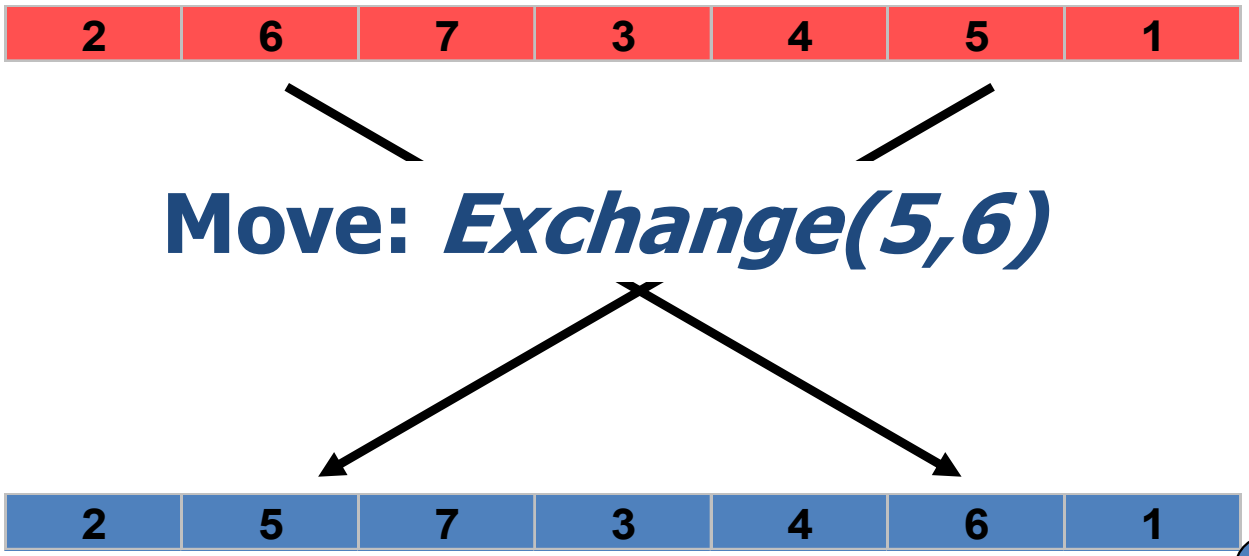
- **Aggressively prioritize good solution attributes in a new solution**
- **Usually based on frequency but may be based on elite solutions**
  - Short term: based directly on the attributes
  - Longer term: use of elite solutions, or parts of elite solutions (vocabulary building)

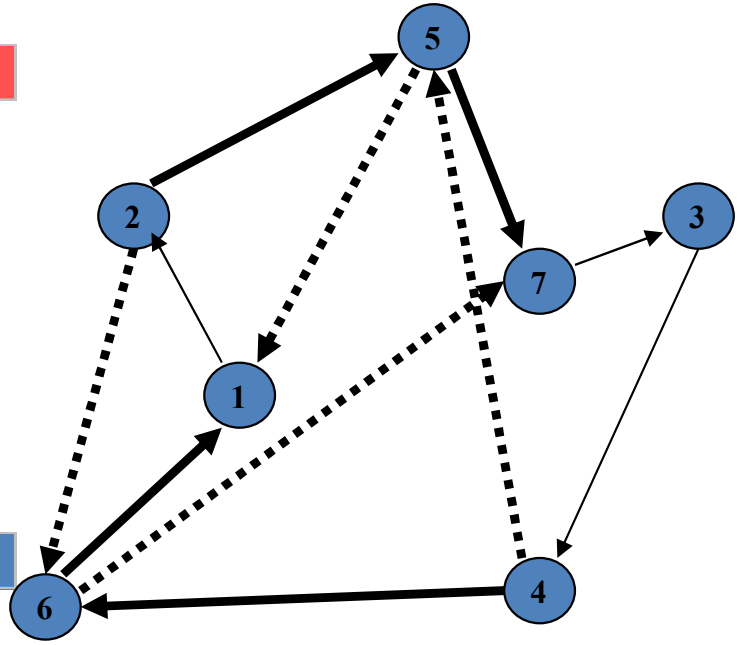
Tabu Search Example: TSP

- **Representation: permutation vector**
- **Move: pairwise exchange**

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







Tabu Search - 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

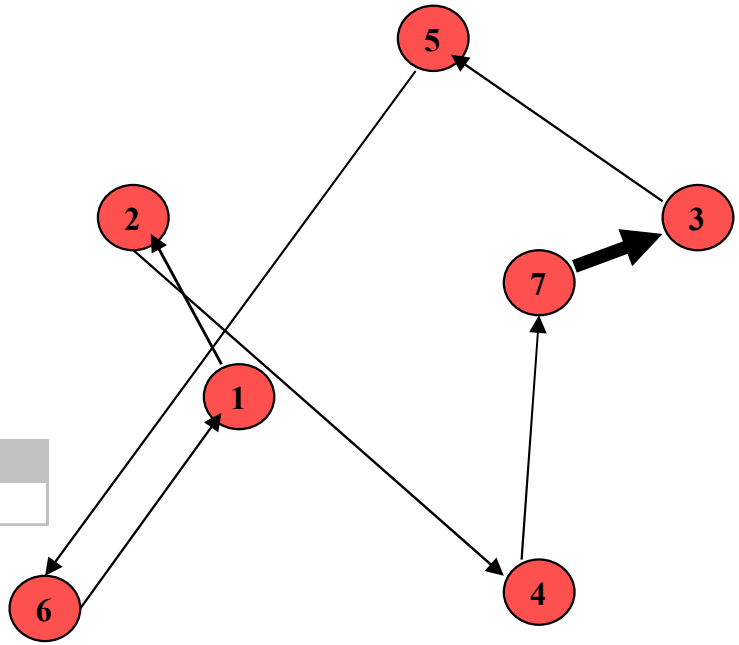
- Data structure: triangular table, storing the number of iterations until moves are legal
- Updated for every move

	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)

NB: 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)

NB: 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

3 out of 21 moves are prohibited

More restrictive tabu effect can be achieved by

- Increasing the tabu tenure
- Using stronger tabu-restrictions such as 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

Tabu Search - Summary

- **Introduction to Tabu Search**
  - Basic Ideas and Algorithm
  - Tabu Criterion
    - Tabu Tenure
  - Aspiration Criterion
  - Examples of use in the Knapsack and TSP
  - Intensification and Diversification
  - Frequency Based Long Term Memory
- **Advanced Topics:**
  - Probabilistic Move Acceptance
  - Strategic Oscillation
  - Exploiting infeasible solutions
  - Path Relinking
  - Ejection Chains

U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Artificial Intelligence

Lecture 3b: Meta-Heuristics – Simulated Annealing and Tabu Search

(based on Løkketangen, 2019)

Luís Paulo Reis

lpreis@fe.up.pt

Director of LIACC – Artificial Intelligence and Computer Science Lab.
Associate Professor at DEI/FEUP – Informatics Engineering Department,
Faculty of Engineering of the University of Porto, Portugal
President of APPIA – Portuguese Association for Artificial Intelligence