Tabu Search - Examples

Petru Eles

Department of Computer and Information Science (IDA)
Linköpings universitet
http://www.ida.liu.se/~petel/

Tabu Search Examples

- **Hardware/Software Partitioning**

- **Travelling Salesman**

TS Examples: Hardware/Software Partitioning

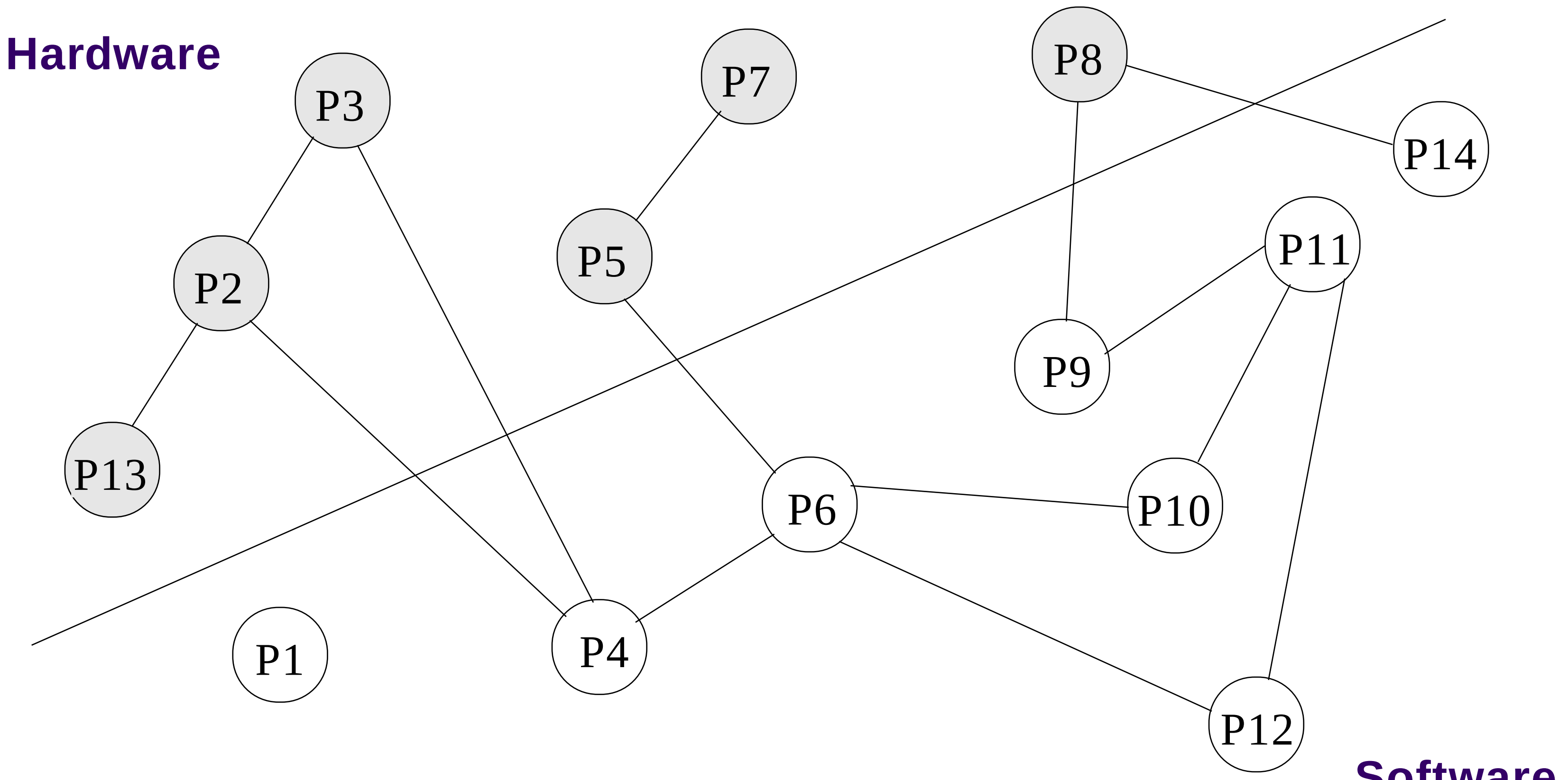
Input:

- The *process graph*: an abstract model of a system:
  - Each node corresponds to a process.
  - An edge connects two nodes if and only if there exists a direct communication channel between the corresponding processes
  - Weights are associated to each node and edge:
    - Node weights reflect the degree of suitability for hardware implementation of the corresponding process.
    - Edge weights measure the amount of communication between processes

Output:

- Two subgraphs containing nodes assigned to hardware and software respectively.

TS Examples: Hardware/Software Partitioning



TS Examples: Hardware/Software Partitioning

Weight assigned to nodes:

$$W2_i^N \quad = \quad M^{CL} \times K_i^{CL} + M^U \times K_i^U + M^P \times K_i^P - M^{SO} \times K_i^{SO}$$

K_i^{CL} **is equal to the RCL of process *i*, and thus is a measure of the computation load of that process;**

$K_i^U = \dfrac{Nr\_op_i}{Nr\_kind\_op_i}$ **;** K_i^U **is a measure of the uniformity of operations in process *i*;**

$K_i^P = \dfrac{Nr\_op_i}{L\_path_i}$ **;** K_i^P **is a measure of potential parallelism inside process *i*;**

$K_i^{SO} = \dfrac{\sum_{op_j \in SP_i} w_{op_j}}{Nr\_op_i}$ **;** K_i^{SO} **captures suitability for software implementation;**

Hw/Sw Partitioning: Cost Function

The cost function:

$$\underbrace{Q1 \times \sum_{(ij) \in cut} W1^E_{ij}}_{\substack{\text{amount of}\\ \text{Hw-Sw comm.}}} + \underbrace{Q2 \times \frac{\displaystyle\sum_{(i) \in Hw} \frac{\displaystyle\sum_{\exists(ij)} W2^E_{ij}}{W1^N_i}}{N_H}}_{\substack{\text{Ratio com/cmp}\\ \text{of Hw part.}}} - \underbrace{Q3 \times \left(\frac{\displaystyle\sum_{(i) \in Hw} W2^N_i}{N_H} - \frac{\displaystyle\sum_{(i) \in Sw} W2^N_i}{N_S} \right)}_{\substack{\text{Difference of}\\ \text{average weights}}}$$

Restrictions:

$$\sum_{i \in H} H\_cost_i \leq Max^H$$

$$\sum_{i \in H} S\_cost_i \leq Max^S$$

$$W^N_i \geq \mathrm{Lim1} \Rightarrow i \in Hw$$

$$W^N_i \leq \mathrm{Lim1} \Rightarrow i \in Sw$$

Hw/Sw Partitioning: Moves&Neighborhood

- **Moves:**
  - **The neighborhood $N(x^{now})$ of a certain solution x^{now} is the set of solutions which can be obtained by moving a node from its current partition to the other one.**

Hw/Sw Partitioning: TS Algorithm

- Construct initial configuration $x^{now} = (Hw_0, Sw_0)$

start:

for each solution $x_k \in N(x^{now})$ *do*

 - Compute change of cost function $\Delta C_k = C(x_k) - C(x^{now})$

end for

for each $\Delta C_k < 0$, in increasing order of ΔC_k *do*

  if not *tabu*(x_k) or *tabu_aspirated*(x_k) *then*

   $x^{now} = x_k$

   goto accept

  end if

end for

for each solution $x_k \in N(x^{now})$ *do*

 - Compute $\Delta C'_k = \Delta C_k + penalty(x_k)$

end for

for each $\Delta C'_k$ in increasing order of $\Delta C'_k$ *do*

  if not tabu(x_k) *then*

   $x^{now} = x_k$

   goto accept

  end if

end for

- Generate x^{now} by performing the least tabu move

accept:

if iterations since previous best solution < *Nr_w_b then*

  goto start

end if

if restarts < *Nr_r then*

 - Generate initial configuration x^{now} considering frequencies

  goto start

end if

return solution corresponding to the minimum cost function

Hw/Sw Partitioning: TS Algorithm

- ***First attempt*:**
  - **An improving non-tabu move (the best possible) is tried.**
- ***Second attempt*:**
  - **Frequency based penalties are applied and the best possible non-tabu move is tried;**
- ***Third attempt*:**
  - **The move which is closest to leave the tabu state is executed.**

Hw/Sw Partitioning: The Tabu-List

- **The last τ moves performed are stored in the *tabu-list*. Their reverse is tabu.**
  τ = *tabu tenure* (length of the tabu list)

- **The tabu tenure depends on the size of the problem and of the neighborhood: large problem sizes are coupled with large tabu tenures.**

- **The tabu tenure depends on the strength of the tabu restriction: stronger restrictions are coupled with smaller sizes.**

- **Tabu tenures are tuned experimentally or can be variable:**
  - **too small tenures $\Rightarrow$ cycling**
  - **too large tenures $\Rightarrow$ deterioration of the solution**
  - **Recommended values: 7 $\div$ 25**

- **Tabu tenures can be selected randomly from a given interval.**

Hw/Sw Partitioning: Tabu Aspiration

- The tabu status of a move is ignored if the solution produced is better than the best obtained so far.

Hw/Sw Partitioning: Long Term Memory

- **Long term memory stores the number of iterations each node has spent in the hardware partition. This information is used for *diversification*:**

  1. **Application of a penalty to the cost function, which favors the transfer of nodes that have spent a long time in their current partition.**

  2. **A move is forbidden (tabu) if the frequency of occurrences of the node in its current partition is smaller than a certain threshold.**

  3. **If the system is frozen a new search can be started from an initial configuration which is different from those encountered previously.**

Hw/Sw Partitioning: Penalty

The penalized cost function:

$$\Delta C'_k = \Delta C_k + \frac{\sum_i |\Delta C_i|}{Nr\_of\_nodes} \times pen(k)$$

Coefficients experimentally set to:

- C_H=0.4
- C_S=0.15.

where

$$pen(k) = \begin{cases} -C_H \times \dfrac{Node\_in\_Hw_k}{N_{iter}} & \textbf{\textit{if node}}_k \in \textbf{\textit{Hw}} \\ -C_S \times \left(1 - \dfrac{Node\_in\_Hw_k}{N_{iter}}\right) & \textbf{\textit{if node}}_k \in \textbf{\textit{Sw}} \end{cases}$$

- ***Node_in_Hw****$_k$* : number of iterations node *k* spent in the Hw partition.
- ***N****$_{iter}$* : total number of iterations;
- ***Nr_of_nodes*** : total number of nodes;

Hw/Sw Partitioning: Thresholds for Node Movement

- A move is forbidden (tabu) if the frequency of occurrences of the node in its current partition is smaller than the threshold:

$$\frac{Node\_in\_Hw_k}{N_{iter}} > T_H \qquad \textbf{\textit{if node}}_k \in \textbf{\textit{Hw}}$$

$$\left(1 - \frac{Node\_in\_Hw_k}{N_{iter}}\right) > T_S \qquad \textbf{\textit{if node}}_k \in \textbf{\textit{Sw}}$$

- The thresholds have been experimentally set to:
  - T_H=0.2
  - T_S=0.4.

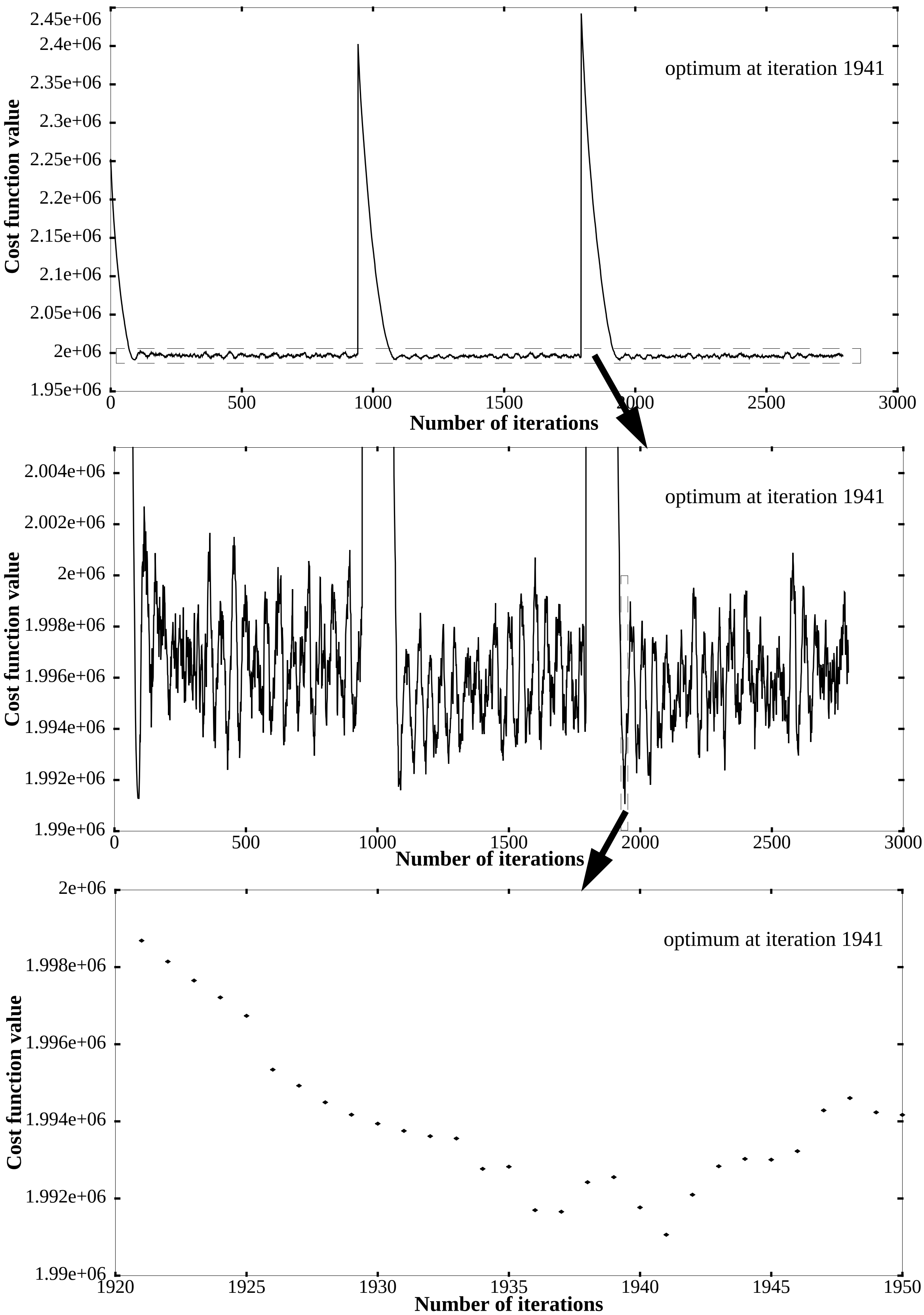
Hw/Sw Partitioning: Some Experimental Results

Parameters and CPU times for Tabu Search partitioning

(SPARCstation 10)

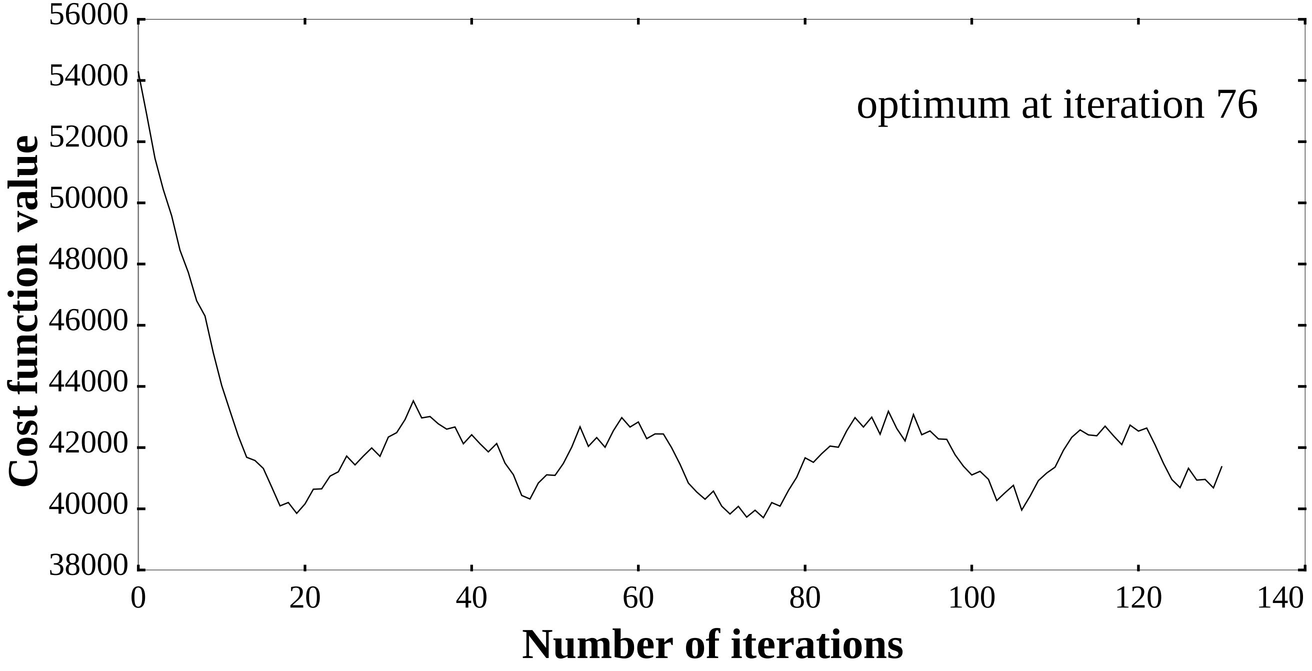
numbers of nodes	τ	Nr_w_b	Nr_r	CPU time (s) (time with SA)
20	7	30	0	0.008 (0.23)
40	7	50	0	0.04 (1.27)
100	7	50	0	0.19 (2.33)
400	18	850	2	30.5 (769)

Variation of cost function for TS partitioning with 400 nodes



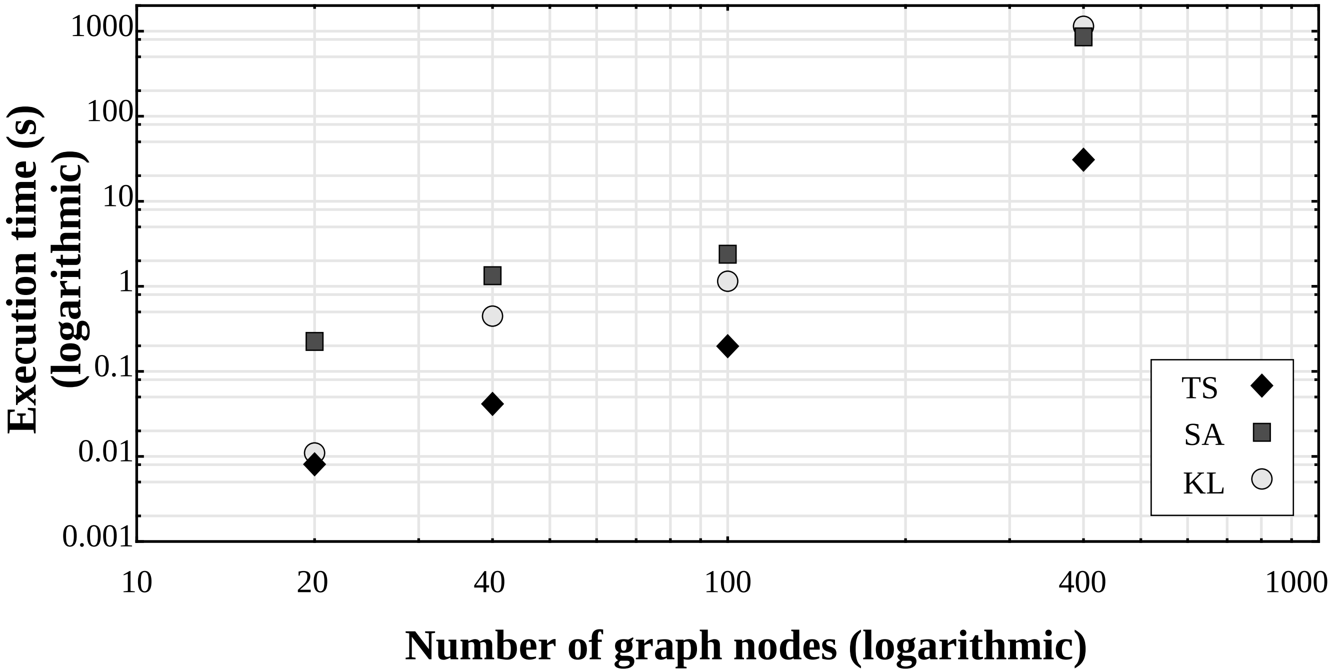
Hw/Sw Partitioning: Some Experimental Results

- Variation of cost function for TS partitioning with 100 nodes



Hw/Sw Partitioning: Some Experimental Results

- **Partitioning times with SA, TS, and KL**



TS Examples: Travelling Salesman Problem

A salesman has to travel to a number of cities and then return to the initial city; each city has to be visited once. The objective is to find the tour with minimum distance.

<u>In graph theoretical formulation</u>:

Find the shortest Hamiltonian circuit in a complete graph where the nodes represent cities. The weights on the edges represent the distance between cities. The cost of the tour is the total distance covered in traversing all cities.

TSP: Cost Function

- If the problem consists of n cities c_i, i = 1, .., n, any tour can be represented as a permutation of numbers 1 to n.
  $d(c_i,c_j) = d(c_j,c_i)$ is the distance between c_i and c_j.

- Given a permutation π of the n cities, v_i and v_{i+1} are adjacent cities in the permutation. The permutation π has to be found that minimizes:

$$\sum_{i=1}^{n-1} d(v_i, v_{i+1}) + d(v_n, v_1)$$

- The size of the solution space is $(n-1)!/2$

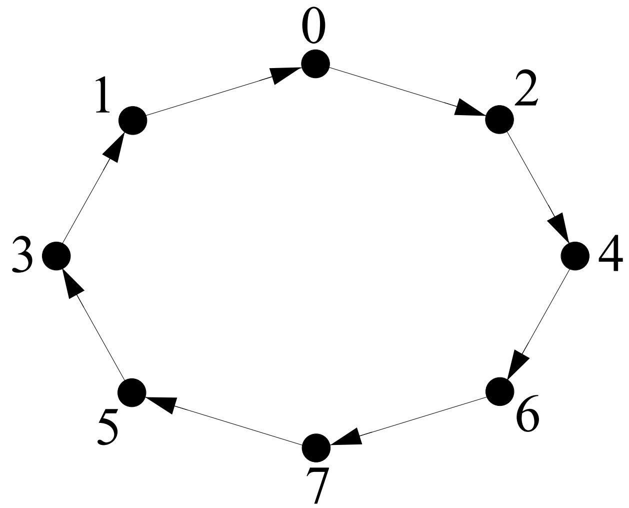
TSP: Moves&Neighborhood

- *k-neighborhood* of a given tour is defined by those tours obtained by removing *k* links and replacing them by a different set of *k* links, in a way that maintains feasibility.

- For *k* = 2, there is only one way of reconnecting the tour after two links have been removed.
  - Size of the neighborhood: $n(n - 1) / 2$
  - As opposed to SA, all alternatives are estimated in order to select the appropriate move.
  - Any tour can be obtained from any other by a sequence of such moves.

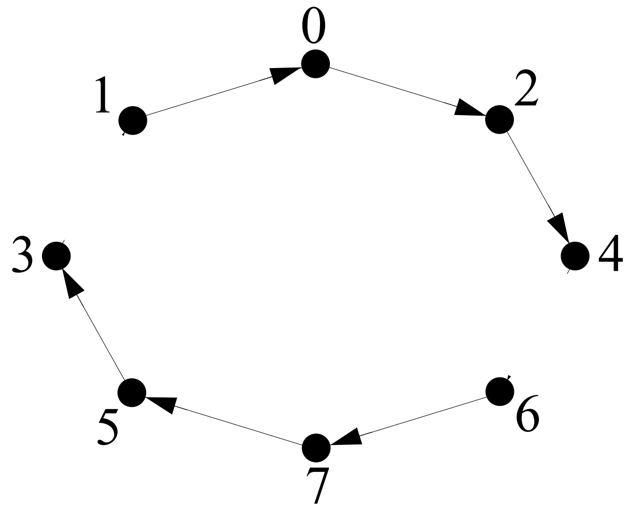
TSP: Moves&Neighborhood

Permutation:
[0 2 4 6 7 5 3 1]

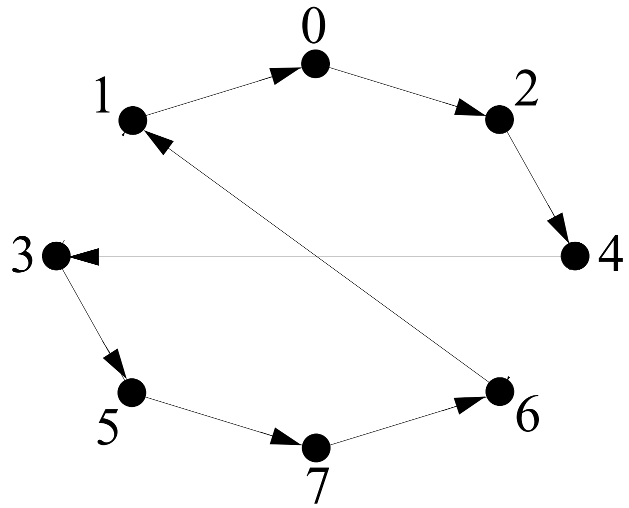


TSP: Moves&Neighborhood

links (v_3,v_1), (v_4,v_6) are removed



Permutation:
[0 2 4 3 5 7 6 1]



TSP: Moves&Neighborhood

- v_i is the city in position i of the tour (i^{th} position in the permutation):

  remove **(v_i, v_{i+1}) and (v_j, v_{j+1})**

  connect **v_i to v_j and v_{i+1} to v_{j+1}**

- **All 2-neighbors of a certain solution are defined by the pair i, j so that $i < j$.**

- **The change of the cost function can be computed incrementally:**

  $\Delta C = d(v_i,v_j) + d(v_{i+1},v_{j+1}) - d(v_i,v_{i+1}) - d(v_j,v_{j+1})$

TSP: Move Attributes for Tabu-Classification

- **We have performed a move as result of which several pairs of cities have swapped their position in the tour:**
  - **Cities *x* and *y* are such a pair;**

    position(*x*) and position(*y*): positions in the tour before the swap.

    position(*x*) < position(*y*).

Questions:

1. **What information do we store (*move attributes*)?**
2. **Using this information, which moves are becoming tabu?**

TSP: Move Attributes for Tabu-Classification

1. Vector(x, y, position(x), position(y))

   - To prevent any swap from resulting in a tour with city x and city y occupying position(x) and position(y) respectively.

2. Vector(x, y, position(x), position(y))

   - To prevent any swap from resulting in a tour with city x occupying position(x) or city y occupying position(y).

3. Vector(x, position(x))

   - To prevent city x from returning to position(x).

TSP: Move Attributes for Tabu-Classification

4. City *x*

   - To prevent city *x* from moving LEFT.

5. City *x*

   - To prevent city *x* from moving.

6. Vector(*y*, position(*y*))

   - To prevent city *y* from returning to position(*y*).

TSP: Move Attributes for Tabu-Classification

7. City *y*
   - To prevent city *y* from moving RIGHT.

8. City *y*
   - To prevent city *y* from moving.

9. City *x and y*
   - To prevent both cities from moving.

TSP: Move Attributes for Tabu-Classification

- Condition 1 is the least restrictive (prevents the smallest amount of moves).

- Condition 9 is the most restrictive (prevents a large amount of moves).

- Conditions 3, 4, 5 have increasing restrictiveness.

TSP: Tabu List Size

- τ has to be experimentally tuned.

  - For highly restrictive tabu conditions τ can be relatively small.
  - For less restrictive tabu conditions τ has to be larger.

  - τ too small $\Rightarrow$ cycling
  - τ too large $\Rightarrow$ exploration driven away from possibly good vicinity.

TSP: Tabu List Size

- *Tabu list size*:
  - for conditions 4 and 7: *nr_cities*/4 ÷ *nr_cities*/3
  - for conditions 5, 8, 9: *nr_cities*/5
  - for conditions 1, 2, 3, 6: ≈*nr_cities*

- Best results for conditions 4 and 7

TSP: Long Term Memory

- Long term memory maintains the number of times an edge is visited.

- After a certain number of iterations a new starting tour is generated consisting of edges that have been visited less frequently.

TSP: Some Experimental Results

- 100 city problem; optimal solution: *C* = 21247.

    - Best solution *C* = 21352 (21255 for SA)
    - Time = 210 s (Sun4/75) - (1340 s for SA)
    - Standard deviation over 10 trials: 30.3;
      (randomly generated starting tour!)
    - Average cost: 21372

- 57 city problem; optimal solution: *C* = 12955

    - Optimal solution in 109 s (673 s for SA).
      (Sequent Balance 8000)