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
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

![Diagram showing hardware and software partitioning with nodes P1 to P14 connected with lines indicating dependencies or connections between hardware and software components.]

Hardware

Software

**Heuristic Algorithms for Combinatorial Optimization Problems**

Tabu Search

Petru Eles, 2010
Weight assigned to nodes:

\[ W_{2i}^N = M^{CL} \times K_{i}^{CL} + M^{U} \times K_{i}^{U} + M^{P} \times K_{i}^{P} - M^{SO} \times K_{i}^{SO} \]

\( K_{i}^{CL} \) is equal to the RCL of process \( i \), and thus is a measure of the computation load of that process;

\[ K_{i}^{U} = \frac{N_{r \_o p_{i}}}{N_{r \_kind \_o p_{i}}} ; \quad K_{i}^{U} \text{ is a measure of the uniformity of operations in process } i; \]

\[ K_{i}^{P} = \frac{N_{r \_o p_{i}}}{L_{\_path_{i}}} ; \quad K_{i}^{P} \text{ is a measure of potential parallelism inside process } i; \]

\[ K_{i}^{SO} = \frac{\sum_{o_{p_{i}} \in S_{P_{i}}} w_{o_{p_{i}}}}{N_{r \_o p_{i}}} ; \quad K_{i}^{SO} \text{ captures suitability for software implementation}; \]
Hw/Sw Partitioning: Cost Function

The cost function:

\[
Q_1 \times \sum_{(i,j) \in cut} W_{1ij} + Q_2 \times \left( \sum_{i \in Hw} \frac{W_{1N_i}}{N_H} \right) - Q_3 \times \left( \sum_{i \in Hw} \frac{W_{2^N_i}}{N_H} - \sum_{i \in Sw} \frac{W_{2^N_i}}{N_S} \right)
\]

amount of Hw-Sw comm.  Ratio com/cmp of Hw part.  Difference of average weights

Restrictions:

\[
\sum_{i \in H} H_{cost_i} \leq Max^H
\]

\[
\sum_{i \in H} S_{cost_i} \leq Max^S
\]

\[W_{iN}^H \geq Lim1 \Rightarrow i \in Hw\]

\[W_{iN}^S \leq Lim1 \Rightarrow i \in Sw\]
Hw/Sw Partitioning: Moves&Neighborhood

- Moves:

  - The neighborhood $N(x^{\text{now}})$ of a certain solution $x^{\text{now}}$ 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 $x^{\text{now}} = (Hw_0, Sw_0)$

\textbf{start:}
\begin{itemize}
  \item for each solution $x_k \in N(x^{\text{now}})$ do
    \begin{itemize}
      \item Compute change of cost function $\Delta C_k = C(x_k) - C(x^{\text{now}})$
    \end{itemize}
\end{itemize}

\textbf{end for}

\begin{itemize}
  \item for each $\Delta C_k < 0$, in increasing order of $\Delta C_k$ do
    \begin{itemize}
      \item if not tabu$(x_k)$ or tabu\_aspirated$ (x_k)$ then
        \begin{itemize}
          \item $x^{\text{now}} = x_k$
          \item goto accept
        \end{itemize}
    \end{itemize}
\end{itemize}

\textbf{end for}

\begin{itemize}
  \item for each solution $x_k \in N(x^{\text{now}})$ do
    \begin{itemize}
      \item Compute $\Delta C'_k = \Delta C_k + \text{penalty}(x_k)$
    \end{itemize}
\end{itemize}

\textbf{end for}

\begin{itemize}
  \item for each $\Delta C'_k$ in increasing order of $\Delta C'_k$ do
    \begin{itemize}
      \item if not tabu$(x_k)$ then
        \begin{itemize}
          \item $x^{\text{now}} = x_k$
          \item goto accept
        \end{itemize}
    \end{itemize}
\end{itemize}

\textbf{end for}

• Generate $x^{\text{now}}$ by performing the least tabu move

\textbf{accept:}
\begin{itemize}
  \item if iterations since previous best solution $< Nr\_w\_b$ then
    \begin{itemize}
      \item goto start
    \end{itemize}
  \end{itemize}

\textbf{end if}

\begin{itemize}
  \item if restarts $< Nr\_r$ then
    \begin{itemize}
      \item Generate initial configuration $x^{\text{now}}$ considering frequencies
      \item goto start
    \end{itemize}
  \end{itemize}

\textbf{end if}

\textbf{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 $\tau$ moves performed are stored in the \textit{tabu-list}. Their reverse is tabu.
  \[ \tau = \text{tabu tenure \ (length of the tabu list)} \]

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

\[ \Delta C'_k = \Delta C_k + \frac{\sum_i |\Delta C_i|}{Nr_{of\_nodes}} \times pen(k) \]

where

\[ pen(k) = \begin{cases} 
-CH \times \frac{Node_{in\_Hw_k}}{N_{iter}} & \text{if } node_k \in Hw \\
-CS \times \left(1 - \frac{Node_{in\_Hw_k}}{N_{iter}}\right) & \text{if } node_k \in Sw 
\end{cases} \]

- **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;

Coefficients experimentally set to:
- \( C_H = 0.4 \)
- \( C_S = 0.15 \).
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:

\[
\frac{\text{Node}_in_{-}\text{Hw}_k}{N_{iter}} > T_H \quad \text{if node}_k \in \text{Hw}
\]

\[
\left(1 - \frac{\text{Node}_in_{-}\text{Hw}_k}{N_{iter}}\right) > T_S \quad \text{if node}_k \in \text{Sw}
\]

- The thresholds have been experimentally set to:
  - \(T_H=0.2\)
  - \(T_S=0.4\).
### Parameters and CPU times for Tabu Search partitioning

(SPARCstation 10)

<table>
<thead>
<tr>
<th>numbers of nodes</th>
<th>( \tau )</th>
<th>Nr_w_b</th>
<th>Nr_r</th>
<th>CPU time (s) (time with SA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>7</td>
<td>30</td>
<td>0</td>
<td>0.008 (0.23)</td>
</tr>
<tr>
<td>40</td>
<td>7</td>
<td>50</td>
<td>0</td>
<td>0.04 (1.27)</td>
</tr>
<tr>
<td>100</td>
<td>7</td>
<td>50</td>
<td>0</td>
<td>0.19 (2.33)</td>
</tr>
<tr>
<td>400</td>
<td>18</td>
<td>850</td>
<td>2</td>
<td>30.5 (769)</td>
</tr>
</tbody>
</table>
Variation of cost function for TS partitioning with 400 nodes

optimum at iteration 1941

optimum at iteration 1941

optimum at iteration 1941
Variation of cost function for TS partitioning with 100 nodes

optimum at iteration 76
Hw/Sw Partitioning: Some Experimental Results

- Partitioning times with SA, TS, and KL

![Graph showing execution time vs. number of graph nodes for SA, TS, and KL methods.](image-url)
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.

**In graph theoretical formulation:**

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.
If the problem consists of $n$ cities $c_i$, $i = 1, \ldots, n$, 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 $\pi$ of the $n$ cities, $v_i$ and $v_{i+1}$ are adjacent cities in the permutation. The permutation $\pi$ 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: \( \frac{n(n - 1)}{2} \)
  - 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.
Permutation: [0 2 4 6 7 5 3 1]
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):
  
  \[
  \begin{align*}
  &\text{remove } (v_i, v_{i+1}) \text{ and } (v_j, v_{j+1}) \\
  &\text{connect } v_i \text{ to } v_j \text{ and } v_{i+1} \text{ to } v_{j+1}
  \end{align*}
  \]

- 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:
  
  \[
  \Delta C = d(v_i, v_j) + d(v_{i+1}, v_{j+1}) - d(v_i, v_{i+1}) - d(v_j, v_{j+1})
  \]
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;
  - $\text{position}(x)$ and $\text{position}(y)$: positions in the tour before the swap.
  - $\text{position}(x) < \text{position}(y)$.

Questions:

1. What information do we store (move attributes)?
2. Using this information, which moves are becoming tabu?
1. Vector\((x, y, \text{position}(x), \text{position}(y))\)
   - To prevent any swap from resulting in a tour with city \(x\) and city \(y\) occupying position\((x)\) and position\((y)\) respectively.

2. Vector\((x, y, \text{position}(x), \text{position}(y))\)
   - To prevent any swap from resulting in a tour with city \(x\) occupying position\((x)\) or city \(y\) occupying position\((y)\).

3. Vector\((x, \text{position}(x))\)
   - To prevent city \(x\) from returning to position\((x)\).
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

- $\tau$ has to be experimentally tuned.
  - For highly restrictive tabu conditions $\tau$ can be relatively small.
  - For less restrictive tabu conditions $\tau$ has to be larger.
  - $\tau$ too small $\Rightarrow$ cycling
  - $\tau$ too large $\Rightarrow$ exploration driven away from possibly good vicinity.
TSP: Tabu List Size

- **Tabu list size:**
  - for conditions 4 and 7: \( nr\_cities/4 \div nr\_cities/3 \)
  - for conditions 5, 8, 9: \( nr\_cities/5 \)
  - for conditions 1, 2, 3, 6: \( \approx nr\_cities \)

- Best results for conditions 4 and 7
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)