Tabu Search origin

General problem solver proposed by Fred GLOVER in 1986 (Uty Colorado)

- Glover et alter, *Future paths for integer programming and links to artificial intelligence*, Computer and OR, 13: 533-549, 1986
- Glover and Laguna, *Modern heuristics for CO problems*, Mac Graw Hill Publisher, London, 1995

Tabu search: forbidden to reuse recently visited solutions or movements (from one solution to another) recently used

Introduce a memory in the search strategy with the aim to diversify the search in the solution space

- Diversify means looking for other solutions or movements than those already used

Use a deterministic heuristic as opposed to most mechanisms implemented in meta solvers rather stochastic

- Because determinism (or usefull knowledge) if faster than random for problem solving
- Examine a large sample of solutions, eventually all, in the neighborhood of the current solution

Tabu Search origin

F. Glover worked on very big (lot of combinations) and difficult problems (lot of local optima)

- A deterministic algorithm hangs in local optima
- A random algorithm does not find good solutions

Third way to solve those problems could be: 1/ find a local optimum, 2/ go away from it, and 3/ dot it again ; he called it the “Tabu Search method (TS)”

- The concept is to use a deterministic procedure to get out of the local optimum and manage the loop which causes the return after some movements
- The deterministic procedure manages a memory (called Tabu List) which prohibits the reuse of neighborhoods already visited: the algorithm is forced to use for a given time a sub-neighborhood that is not yet explored
- He defined two ways to do it: strict Tabu List or non-strict Tabu List

TS strict process

1.Define a Tabu List that memorizes the solutions already visited (this list is called a "strict list").

2.Explore the neighboring solutions, non tabu, of the current solution.

3.Choose the best of these neighbors: it is not necessarily better than the current solution => TS is never blocked when it finds a local optimum.

4.Update the Tabu List: the previous solution becomes tabu

Algorithm1*: *Tabu Search with prohibited solutions

1.Initialisation

s_0 starting solution

$s \leftarrow s_0,\ s* \leftarrow s_0, c* \leftarrow f(s_0)$

$T = \emptyset$ $*$ *The Tabu List*

2.Generate a subset in the neighborhood of s

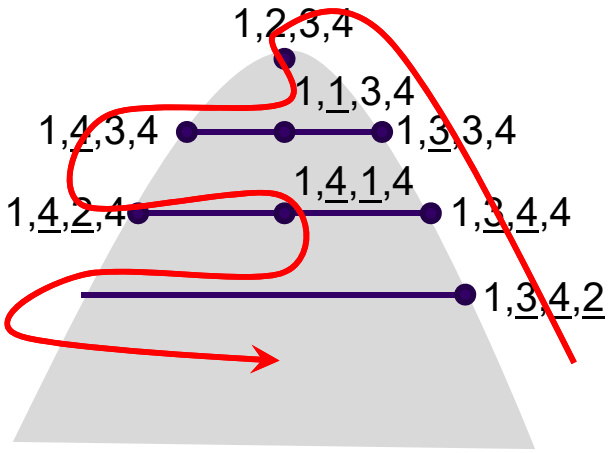
$s' \in N(s)/\forall s" \in N(s), f(s') \leq f(s")$ and $s' \notin T$

$If\ f(s') < c*\ then\ s* \leftarrow s', c* \leftarrow f(s')$

$Update\ the\ tabu\ list\ T: T \leftarrow T \cup \{s\}$

$s \leftarrow s'$

3.Loop in 2 if stopping condition not reached



Strict-list settings

Neighborhood definition: what can we change inside a solution

- It is dedicated for each specific problem as in local search general solver

Tabu Duration (TD) : the list of stored tabu solutions is finite of length TD to fix => the algorithm prevents any cycle of length smaller than TD

- Get out of the Tabu List: FIFO
- Long term => long list => good memory (few risk of loop) but time consuming
- Short term => short list => fast to run but bad memory (high risk of loop) => are you far enough away from the local optimum to avoid going back ?
- Parameter settings: calculate the Hamming distance with the local optima and set a threshold for the distance e.g. “if 10% of the variables are different the algorithm is far enough”
- This threshold must be set for each problem => it needs a **learning mechanism by data analytics**

Tabu Search « intelligence »

1. Strict restriction of possible choices: all movements leading to a tabu solution are forbidden => the neighborhood changes and is more and more restricted with the progress of the search
2. Intensive search: it allows a complete exploration around a local optimum, this is good if it is in the path to the global optima

KPI on Tabu Search performance

If a local optimum is high or isolated, the finite size Tabu List will not prevent the algorithm from looping around this local optimum and returning to it.

This loop is identifiable by measuring the frequency of visits of the variables of the problem. When detected, it is necessary to stop the process and change it: 1/ enlarge the Tabu Duration or 2/ change the neighborhood or 3/ generate a new starting solution.

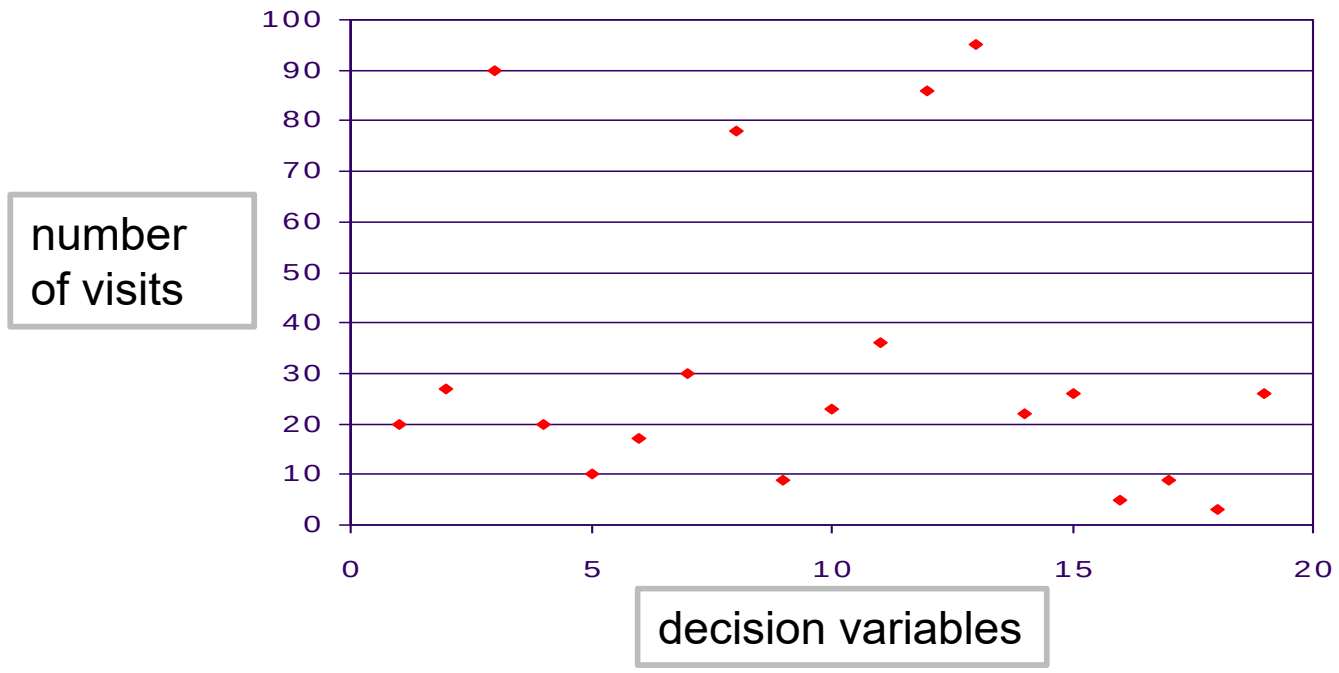
Let, n_j, the number of visit of the variable j

KPI examples:

$$\frac{n_j}{iteration\_number}$$

$$\frac{n_j}{\max\{n_j, \forall j\}}$$

$$\frac{n_j}{\bar{n}}$$



TS non-strict process

A move from one state to another is defined by all the variables who change their value from the state *s* to *s'*$\in N(s)$; *usually only one variable at a time*.

1. Define a Tabu List that memorizes the moves already applied (this list is called a "non-strict list").
2. Explore the neighboring solutions, with a non Tabu Move from the current solution: either the variable choice is prohibited, or the variable and value choices are prohibited.
3. Choose the best of these neighbors: it is not necessarily better than the current solution => TS is never blocked when it finds a local optimum.
4. Update the Tabu List: the move to go from *s* to *s'* becomes tabu i.e. the updated variable or the updated couple (variable, old value).

Algorithm2*: *Tabu Search with prohibited moves

2. Generate a subset in the neighborhood of s

$s' \in N(s)/\forall s" \in N(s), f(s') \leq f(s")$ and $move(s,s') \notin T$

$If\ f(s') < c*\ then\ s* \leftarrow s', c* \leftarrow f(s')$

$Update\ the\ tabu\ list\ T: T \leftarrow T \cup \{move(s,s')\}$

$s \leftarrow s'$

Non-strict list settings

Algorithm2*: *Tabu Search with prohibited moves

2.Generate a subset in the neighborhood of s

$s' \in N(s)/\forall s" \in N(s), f(s') \leq f(s")$ and $move(s,s') \notin T$

$If\ f(s') < c*\ then\ s* \leftarrow s', c* \leftarrow f(s')$

$Update\ the\ tabu\ list\ T: T \leftarrow T \cup \{move(s,s')\}$

$s \leftarrow s'$

The Tabu Moves cannot be applied whatever the solutions *s* and *s' i.e. all solutions requiring one Tabu Move become forbidden.*

NB: the non-strict list directly impact the frequency of visits of the variables i.e. the number of variable visits will be reduce if the Tabu Move only memorizes the variable.

NB: the non-strict list avoid to return on a previous visited solution as any subset of that solution becomes prohibited (e.g. all previous solutions with $x_3 = 8$).

Tabu Search « intelligence »

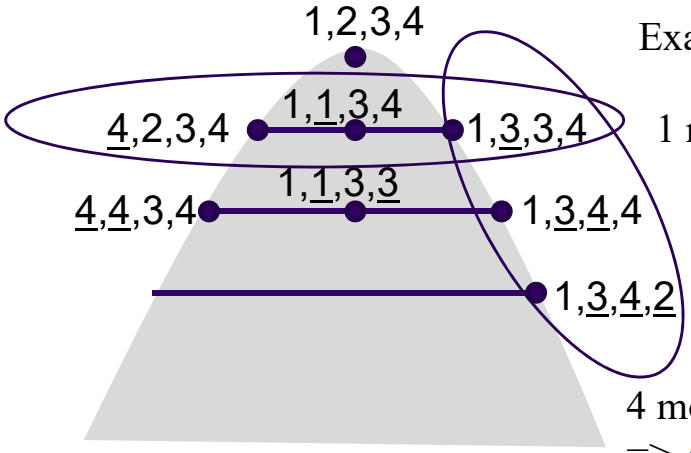
1. Loop avoidance: the algorithm forces to diversify the variable or the variable/value choice (long term memory effect) and then avoid the loops
2. Explorative search: because the local minimum is surrounded by close solutions (subset of variables with the same values), it forces to go away from it ; this is good if it is not in the path to the global optima

Non-strict list settings

Distance between two solutions *s* and *s'* : the Hamming distance is a distance in the mathematical sense which quantifies the difference between two sequences of symbols. To two series of symbols of the same length, it associates the number of positions where the two suites differ.

$$s = (x_i), s' = (x'_i), i \in [\![1, n]\!], \text{Hamming}(s, s') = \sum_{i=1}^{n} ! \, |x_i == x'_i|$$

When a variable is tabu (e.g. x_2), all solutions with a different value for x_2 are tabu => the algorithm moves faster away from the local optimum than if the couple (variable, value) is tabu (e.g. if x_2=2 is tabu, then all solutions with x_2=3 are not tabu).



Example with a 4 variables problem:

1 modified variable, we stay close to the local optimum

2 modified variables, we roll away from the local optimum

3 modified variables, we move away from the local optimum

4 modified variables, NO MORE MOVE are possible

=> **An adaptive Tabu Duration is necessary in the non-strict list too**

Non-strict list settings

Tabu Duration for a move: the Tabu Duration prohibits to reuse the move for a number of iterations. This number must be set according to the number of variables and possible values for the variables in order not to block the algorithm.

Let *n*, the number of unknown variables of the problem

td, the Tabu Duration be defined by:

- *td(n)*, *constant function* of *n* (often $td(\mathrm{n}) = \sqrt{n}$); not good results usually.
- *td(n), dynamic function* of *n*

1. $td_{min} < td < td_{max}$ with $td_{min}=0{,}5\ \sqrt{n}$ and $td_{max}=1{,}5\ \sqrt{n}$
2. $td(n_j) = \alpha * n_j * n^{-1}$ and n_j, the number of visit of the variable *j*

   // the more the variable is used, the longer its Tabu Duration
3. $td(iteration\_number) = \alpha * LOG(iteration\_max + 1 - iteration\_number) * n^{-1}$

   // long Tabu Duration first to diversify, then shorter and shorter
4. a rule that depends on the quality of the move e.g. shorter duration for moves that have improved because the landscape quality may be very sensitive to that variable.