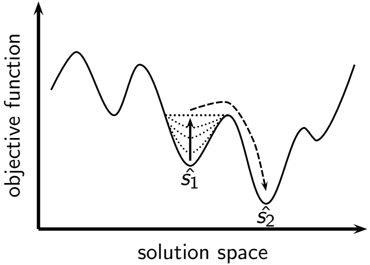
Tabu Search, Simulated Annealing, and Guided Local Search (Vienna 2022)

Christian Blum

Artificial Intelligence Research Institute (IIIA-CSIC)



Essential information

- **Introduced** by [Glover, 1986][a], based on ideas already published in [Glover, 1977][b]
- **A similar idea:** *steepest ascent mildest descent*, invented by [Hansen, 1986][c]

---

[a] F. Glover, 1986. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, *13*, 533–549

[b] F. Glover. Heuristics for integer programming using surrogate constraints, *Decision Sciences*, 8:156–166, 1977

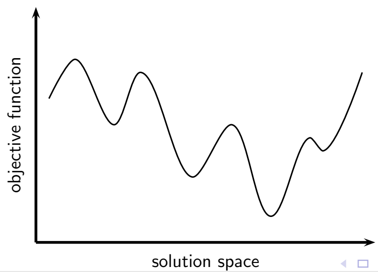
[c] P. Hansen, 1986. The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy, 1986

TS: main ideas

Ideas

1. Allow to move to solutions worse than the current one if necessary
2. Use a mechanism that prevents moves to recently visited solutions → **tabu lists**

Why are both ideas necessary?



Tabu lists (1)

Tabu lists: characteristics

- Generally implemented as FIFO (first-in-first-out) lists
- They may have a fixed or variable length
- They might store entire solutions. **Disadvantage:**
  1. Storage space requirements
  2. Comparing solutions might be expensive
- **Therefore:** tabu lists rather store solution attributes instead of complete solutions

Note

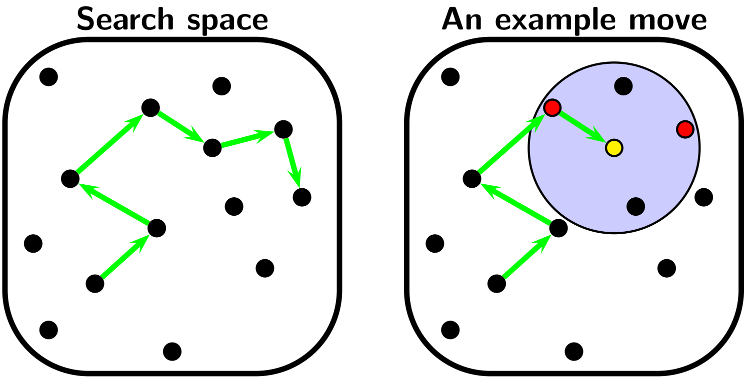
For each type of solution attribute (for example, nodes resp. edges of a graph) a tabu list is maintained

Tabu lists (2)

How are tabu lists used?

1. Tabu lists are used to classify the solutions $s' \in \mathcal{N}(s)$:
   - **tabu:** s' can not be considered for a move
   - **not tabu:** s' is a feasible neighbor
2. This results in the restricted set $\mathcal{N}^{nt}(s) \subseteq \mathcal{N}(s)$ of neighbors
3. Choose the best solution s' from $\mathcal{N}^{nt}(s)$
4. Update the tabu list according to the move $s \mapsto s'$

Tabu lists: graphical illustration

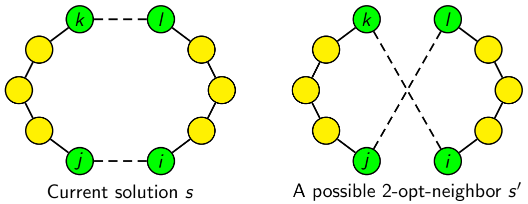


Tabu lists: TSP example (1)

2-opt neighborhood

- Solution attributes:
  1. Edges that are **removed** from a solution/tour
  2. Edges that are **added** to a solution/tour
- Therefore: use of 2 tabu lists
  1. **OutList** stores the edges that are removed from a tour
  2. **InList** stores the eges that are added to a tour

Graphical illustration



Tabu lists: TSP example (2)

For transforming s into s' ...

- Remove edges $e_{k,l}$ and $e_{j,i}$ from s
- Add edges $e_{k,i}$ and $e_{l,j}$ to s

Is this move feasible according to the tabu lists?

This move is unfeasible (that is, s' is **tabu**), if and only if

- $e_{k,i}$ or $e_{l,j}$ are in **OutList** or
- $e_{k,l}$ or $e_{j,i}$ are in **InList**

Tabu lists: a well known problem

Attention with the following issue

- By only storing features of solutions we might forbid (declare as tabu) so-far unvisited solutions (*Example on the board!!*)
- This is, however, only a problem if the forbidden unvisited solutions are very good

Posible solution to this problem

- **Aspiration criteria**: Define conditions for canceling the tabu status of a move
- **Example:** If the move is better than the best solution found so far

Tabu lists: TSP example

Update of the tabu lists

- **Remember:** tabu lists normally work in a FIFO manner
- **TSP example:**
  1. Drop the 2 edges that are longest in **OutList**
  2. Drop the 2 edges that are longest in **InList**
  3. Add edges $e_{k,l}$ and $e_{j,i}$ to **Outlist**
  4. Add edges $e_{k,i}$ and $e_{l,j}$ to **InList**

Basic tabu search

Pseudo code

$s \leftarrow$ GenerateInitialSolution()
InitializeTabuLists($TL_1, \ldots, TL_r$)
while termination conditions not met **do**
    $\mathcal{N}_a(s) \leftarrow \{s' \in \mathcal{N}(s) \mid s'$ does not violate a tabu condition, or it satisfies at least one aspiration condition$\}$
    $s' \leftarrow \text{argmin}\{f(s'') \mid s'' \in \mathcal{N}_a(s)\}$
    UpdateTabuLists(TL_1, ..., TL_r,s,s')
    $s \leftarrow s'$
end while

Tabu search: additional examples

On the board!

- Tabu search for the k-cardinality tree (KCT) problem
- Tabu search for job shop scheduling (JSS)

Tabu search: advanced topics (1)

Tabu list length

- **Problem:** Which is a good length of the tabu list?
  1. Short lists: the search process will focus on small areas of the search space (**intensificación**)
  2. Long lists: the search process is forced to explore larger areas of the search process (**diversificación**)

Cycling

- **Related problem:** the search process might enter into a cycle
- **Definition of a cycle:** the repetition of the same sequence of solutions over and over again

Tabu search: advanced topics (2)

Solutions to the afore-mentioned problems

- **Simple option:** try to find a good compromise (by tuning)
- **Robust tabu search:** periodically reinitialize the tabu list length from $[l_{min}, l_{max}]$
- **Reactive tabu search:**
  1. **Increase** tabu list length when there is evidence for the repetition of solutions
  2. **Decrease** tabu list length when there are many improvements

Tabu search: advanced topics (3)

Making use of long-term memory

- Tabu lists are generally regarded as short term memory
- **Observation:** the second-best feasible neighbor (instead of the best one) would sometimes be a better choice
- **A way of taking profit from second-best neighbors:**
  1. Keep a memory of the best second-best neighbors
  2. In situations in which the search process seems stuck, re-start the search from one of the second-best neighbors from the memory

Tabu search: continuous optimization

Discretization of the search space

- Best known approaches are based on a discretization of the search space
- Exists a version of reactive tabu search
- Enhanced continuous tabú search (**on the board!!**)

Questions?

Simulated Annealing (SA)

Essential information

- Commonly considered to be the oldest metaheuristic
- Origins in statistical mechanics (annealing process of glas and metal)
- **Introduced by** [Kirkpatrick et al., 1983][a] and [Cerny, 1985][b]
- Search process of SA produces a Markov chain

[a] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, 13 May 1983*, 220(4598):671–680, 1983

[b] V. Cerny. A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985

Reminder: basic local search

Pseudo code: local search

$s \leftarrow$ GenerateInitialSolution()
while $\exists\, s' \in \mathcal{N}(s)$ such that $f(s') < f(s)$ **do**
  $s \leftarrow$ ChooseImprovingNeighbor($\mathcal{N}(s)$)
end while

Implementation of ChooseImprovingNeighbor($\mathcal{N}(s)$)

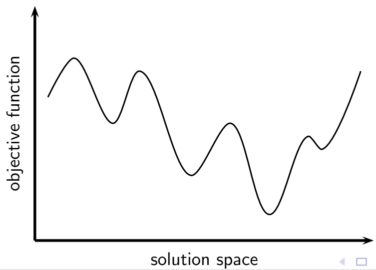
- **First improvement:** Scans the neighborhood an returns the first improving neighbor
- **Best improvement:** Returns the best neighbor of the neighborhood

SA: basic ideas

Basic ideas

1. Allow to accept solutions worse than the current one if necessary
2. Make the choice of the next solution probabilistic
3. The acceptance decision is probabilistic

Why are the first two ideas necessary?



SA: pseudo code

Pseudo code

$s \leftarrow$ GenerateInitialSolution()
$T \leftarrow$ SetInitialTemperature()
while termination conditions not met **do**
  $s' \leftarrow$ PickNeighborAtRandom($\mathcal{N}(s)$)
  if $(f(s') < f(s))$ **then**
    $s \leftarrow s'$
  else
    Accept s' as new solution with probability $\mathbf{p}(s' \mid T, s)$
  end if
  AdaptTemperature(T)
end while

SA: design choices

Initial solution

- **Generally:** randomly generated
- **Also possible:** use of a heuristic

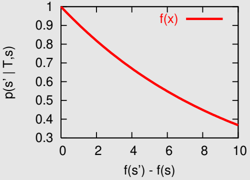
Acceptance probability

$$\mathbf{p}(s' \mid T, s) = e^{-\frac{f(s')-f(s)}{T}}$$

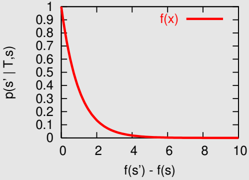
where T is the so-called temperature parameter

SA: acceptance probability

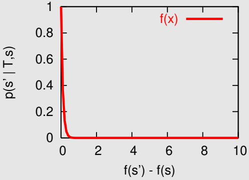
Influence of T



$T = 10$



$T = 1$



$T = 0.1$

Adaptation of T

Initial and final settings of T

- At the beginning of the search process: T should be high
  Goal: favoring the exploration of the search space
- At the end of the search process: T should be low
  Goal: find a local (possibly global) minimum

Adaptation of T: *Cooling schedule*

- **Standard:** Continuously decreasing
  - **Example:** geometric cooling, $T \leftarrow \alpha \cdot T$, where $\alpha \in (0, 1)$
- **More advanced:** Re-heating schemes (or non-monotonic cooling)

Adaptation of T

Attention

- If the initial value of T is too high: waste of computation time due to an extended random search phase
- If the initial value of T is too low: pre-mature convergence to some basin of attraction
- If the final value of T is too high: algorithm lacks the intensification phase (no good solutions will be found)

SA: animation

Theoretical result

Theorem

$\exists\, r \in \mathbb{R}^+$ such that $\lim_{k\to\infty} \mathbf{p}(\text{global min. found after } k \text{ iters.}) = 1$

iff $\sum_{k=1}^{\infty} e^{\left(\frac{r}{T_k}\right)} = \infty$

Which cooling schedule applies?

For example the **logarithmic schedule**: $T_k \leftarrow \frac{r}{\log(k+c)}$ (where c is a constant)

Is this result useful?

For practice: no! Too slow

SA variant: threshold accepting (TA)

Idea

Instead of an acceptance probability use an explicit acceptance threshold

Pseudo code

$s \leftarrow$ GenerateInitialSolution()
$T \leftarrow$ SetInitialThreshold()
while termination conditions not met **do**
  $s' \leftarrow$ PickNeighborAtRandom($\mathcal{N}(s)$)
  if $(f(s') - f(s)) \leq T$ **then**
    $s \leftarrow s'$
  end if
  ReduceThreshold(T)
end while

EXample: TSP

TSP example: design decisions

- Neighborhood: 2-opt
- Adaptation of T: geometric cooling schedule

Other examples

On the board!!

SA for continuous optimization

Two options! On the board!

1. Discretization of the search space $\rightarrow$ apply discrete SA
2. SA for continuous spaces

Questions?

Guided local search (GLS)

Essential information

- One of the most recent metaheuristic methods
- **Introduced by** [Voudouris, 1997][a] y [Voudouris and Tsang, 1999][b].

[a] C. Voudouris. *Guided Local Search for Combinatorial Optimization Problems*. PhD thesis, Department of Computer Science, University of Essex, 1997

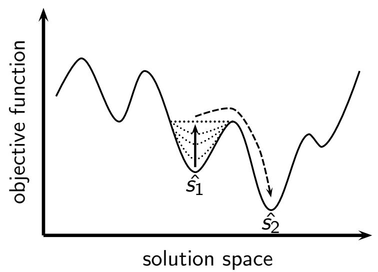
[b] C. Voudouris and E. Tsang. Guided Local Search. *European Journal of Operational Research*, 113(2):469–499, 1999

GLS: basic ideas

Basic ideas

1. Dynamically change the objective function depending on the search history
2. Changing the objective function means: changing the search landscape

Why are both ideas necessary?



GLS: objective function

Change of the objective function (based on search history)

1. First, define a set $\mathcal{I} = \{1, \ldots, n\}$ of solution features
2. A function $\delta : \mathcal{I} \times \mathcal{S} \mapsto \{0, 1\}$ indicates if a feature $i \in \mathcal{I}$ is present in a solution $s \in \mathcal{S}$
3. Second, introduce a penalty value $p_i \geq 0$ for each solution feature $i \in \mathcal{I}$
4. Third, add the penalty values to the objective function:

$$f'(s) \leftarrow f(s) + \lambda \cdot \sum_{i=1}^{n} p_i \cdot \delta(i, s)$$

where $\lambda > 0$ is a parameter to adjust the strength of the penalty term

Guided local search (GLS)

Pseudo code

$s \leftarrow$ GenerateInitialSolution()
$\mathbf{p} \leftarrow (0, \ldots, 0)$ {initialization of penalties}
while termination conditions not met **do**
  $\hat{s} \leftarrow$ LocalSearch(s, f')
  UpdatePenalityVector($\mathbf{p}$,$\hat{s}$)
  $s \leftarrow \hat{s}$
end while

GLS: penality vector update

- **Generally:** each feature i has an associated cost c_i, $i = 1, \dots, n$
- Punish all features $i \in \hat{s}$ that maximize the so-called utility function:

$$u(\hat{s}, i) = \frac{c_i}{1 + p_i}$$

- **Punishment:** $p_i \leftarrow p_i + 1$

Design guidelines

- Carefully tune the setting of λ
- Test different penalty update procedures. They largely determine the success of the algorithm

EB-GLS: a variant of GLS

Problem

- Not only **bad features** are punished, also good ones.
- **Possible solution:** do not punish the features of the best-so-far solution (s^{bsf})
- **GLS variant:** Elite biased GLS[a]

[a] Shi, J., Zhang, Q., & Tsang, E. EB-GLS: an improved guided local search based on the big valley structure. *Memetic Computing*, 10(3), 333-350, 2018.

Main difference between EB-GLS and GLS: function $u(\hat{s}, i)$

- **Original:** $u(\hat{s}, i) = \frac{c_i}{1+p_i}$
- **EB-GLS:** $u(\hat{s}, i) = \frac{c_i \cdot \Delta(i, s^{\text{bsf}})}{1+p_i}$, where $\Delta(i, s^{\text{bsf}}) = w > 1$ in case i is not present in s^{bsf}; $\Delta(i, s^{\text{bsf}}) = 1$ otherwise.

GLS: examples

GLS for the TSP

- Each edge $e \in E$ is a solution feature i_e
- The cost of an edge e is its distance d_e
- **Effect:** making often-used edges less desirable over time
- **Local search:** 2-opt neighborhood, best-improvement

Other examples: on the board!

- GLS for the *quadratic assignment problem* (QAP)
- GLS for continuous optimization

Questions?