

Tabu Search, Simulated Annealing, and Guided Local Search

Christian Blum

Artificial Intelligence Research Institute (IIIA-CSIC)
Bellaterra, Spain

Outline

Subjects

- ▶ **Tabu search (TS)**
 - ▶ Basic idea, framework, examples
 - ▶ Reactive Search
 - ▶ Variant for continuous optimization
- ▶ **Simulated annealing**
 - ▶ Basic idea, framework and examples
 - ▶ Threshold accepting algorithms
 - ▶ Variant for continuous optimization
- ▶ **Guided local search**
 - ▶ Basic idea, framework and examples

Historical note

1. **Introduced** by [Glover, 1986], based on ideas formulated in [Glover, 1977]
2. Similar ideas were labelled **steepest ascent mildest descent** by [Hansen, 1986]

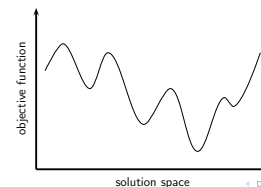
Basic references

- ▶ [Glover, 1977] Heuristics for integer programming using surrogate constraints, *Decision Sciences*, 8:156–166, 1977
- ▶ [Glover, 1986] F. Glover, 1986. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13, 533–549
- ▶ [Hansen, 1986] P. Hansen, 1986. The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy, 1986

Main idea

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 1) and 2) necessary?



Tabu lists (1)

Tabu lists ...

- ▶ are generally **FIFO** (first-in-first-out) lists
- ▶ they may have a **fixed or variable** length
- ▶ generally store **solution features** instead of whole solutions
- ▶ might store whole solutions. **Disadvantage:**
 1. Storage space requirements
 2. Comparing solutions might be expensive

Note

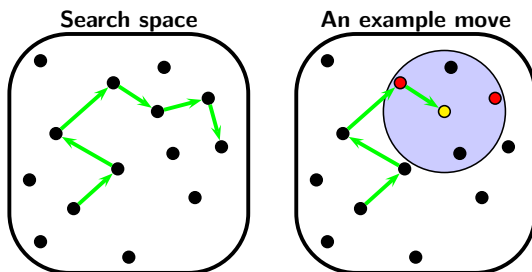
Generally we have for each (selected) solution feature one tabu list

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 the 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 from $\mathcal{N}^{nt}(s)$
4. **Update** the tabu list according to the move $s \mapsto s'$

Tabu lists: graphically

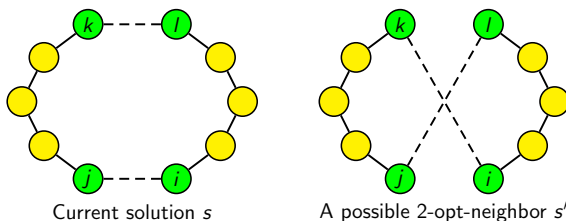


Tabu lists: TSP example (1)

2-opt neighborhood

- ▶ **Solution features:**
 1. Edges that are **removed** from a tour
 2. Edges that are **added** to a tour
- ▶ **Use of 2 tabu lists:**
 1. **OutList** stores the edges that are removed from a tour
 2. **InList** stores the edges that are added to a tour

Tabu lists: TSP example (2)



Tabu lists: TSP example (3)

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?

This move is unfeasible (that is, s' is **tabu**), iff

- ▶ $e_{k,i}$ **OR** $e_{l,j}$ are in the **OutList** **OR**
- ▶ $e_{k,l}$ **OR** $e_{j,i}$ are in the **InList**

Tabu lists: Problem

Attention

- ▶ By only storing **features of solutions** we might **forbid unvisited solutions** (**Example on the board!!**)
- ▶ This is only a problem if the forbidden unvisited solutions are very good

Solution

- ▶ **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 (4)

Tabu list update

- ▶ Tabu list are generally managed as **FIFO** lists
- ▶ **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 ← GenerateInitialSolution()
InitializeTabuLists( $TL_1, \dots, TL_r$ )
while termination conditions not met do
   $\mathcal{N}_a(s) \leftarrow \{s' \in \mathcal{N}(s) \mid s' \text{ does not violate a tabu condition, or}$ 
     $\text{it satisfies at least one aspiration condition}\}$ 
   $s' \leftarrow \operatorname{argmin}\{f(s'') \mid s'' \in \mathcal{N}_a(s)\}$ 
  UpdateTabuLists( $TL_1, \dots, TL_r, s, s'$ )
   $s \leftarrow s'$ 
end while

```

Additional examples

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

On the board!!

Tabu search: advanced topics (1)

Tabu list length

- ▶ **Problem:** Which is a good length of the tabu list?
 1. **Small tabu list length:** The search process will focus on small areas of the search space (**good intensification**)
 2. **Large tabu list length:** The search process is forced to explore larger areas of the search process (**good diversification**)
- ▶ **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 these problems

- ▶ **Simple:** 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)

Addition of long term memory

- ▶ Tabu lists are generally regarded as short term memory
- ▶ **Observation:** The second-best feasible neighbor would sometimes be a better choice
- ▶ **Therefore:**
 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 for continuous optimization

TS for cont. optimization

Best known approaches are based on a discretization of the search space:

- ▶ Exists a version of **reactive tabu search**
- ▶ **Enhanced continuous tabu search (on the board!!)**

Simulated annealing

Historical note

- ▶ Commonly accepted to be the **oldest metaheuristic**
- ▶ Origins in **statistical mechanics** (Annealing process of glass and metal)
- ▶ **Introduced** by [Kirkpatrick et al., 1983] and [Cerny, 1985]
- ▶ Search process produces a **Markov chain**

Important references

- ▶ [Cerny, 1985] V. Cerny. A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985
- ▶ [Kirkpatrick et al., 1983] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 13 May 1983, 220(4598):671–680, 1983

Reminder: basic local search pseudo-code

```

s ← GenerateInitialSolution()
while ∃ s' ∈ N(s) such that f(s') < f(s) do
  s ← ChooseImprovingNeighbor(N(s))
end while

```

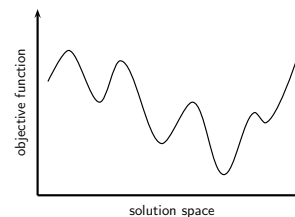
Implementations of ChooseImprovingNeighbor(N(s))

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

Main idea

1. Allow to **accept solutions worse than the current one**
2. Make the **choice of the next solution probabilistic**

Why are 1) and 2) necessary?



Simulated annealing: pseudo code

```

s ← GenerateInitialSolution()
T ← SetInitialTemperature()
while termination conditions not met do
  s' ← PickNeighborAtRandom(N(s))
  if (f(s') < f(s)) then
    s ← s'
  else
    Accept s' as new solution with probability p(s' | T, s)
  end if
  AdaptTemperature(T)
end while

```

Design choices

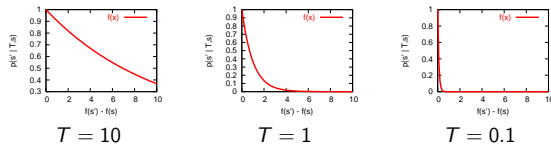
1. **Initial solution:**
 - ▶ **Generally:** randomly chosen
 - ▶ **Also possible:** heuristically generated
2. **Acceptance probability:**

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

where T is a so-called **temperature parameter**

Acceptance probability

Influence of T



Design choice: Management of T

Initial and final settings

- At the beginning of the search process: T should be **high**
Goal: favouring 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
 - Geometric cooling:** $T \leftarrow \alpha \cdot T$, where $\alpha \in (0, 1)$
- More advanced:** Re-heating schemes (or non-monotonic cooling)

Design choice: Management of T

Attention

- If **initial T too high:** Waste of computation time due to an extended random search
- If **initial T too low:** Pre-mature convergence to some basin of attraction
- If **final T too high:** Algorithm lacks the intensification phase (no good solutions will be found)

Design choice: Management of T

How to set initial and final T ?

- Determine the **distribution of the objective function values**
 - Upper and lower bounds of the objective function
 - By randomly sampling the search space
- Example:**
 - Set **initial T** such that a 5%-worsening move is accepted with probability 95%
 - Set **final T** such that a 5%-worsening move is accepted with probability 5%

Theoretical result

Theorem

$$\exists r \in \mathbb{R}^+ \quad \text{s.t.} \quad \lim_{k \rightarrow \infty} \mathbf{p}(\text{global minimum found after } k \text{ steps}) = 1$$

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

Which cooling schedule applies?

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

Is it useful? **No!** Too slow for practical purposes

SA variant: Threshold accepting (TA)

Idea

Instead of an **acceptance probability** use an **explicit acceptance threshold**

Pseudo-code

```

s ← GenerateInitialSolution()
T ← SetInitialThreshold()
while termination conditions not met do
  s' ← PickNeighborAtRandom(N(s))
  if (f(s') - f(s)) ≤ T then
    s ← s'
  end if
  ReduceThreshold(T)
end while
  
```

Example: TSP

- ▶ 2-opt neighborhood structure
- ▶ geometric cooling schedule

Other examples

On the board!!

Simulated annealing for continuous optimization

As usual: 2 possibilities

1. **Discretization** of the search space → apply discrete SA
2. SA for real continuous spaces

On the board!!

Guided local search

Historical note

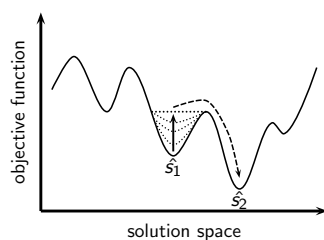
- ▶ One of the **most recent** metaheuristic methods
- ▶ **Introduced** by [Voudouris, 1997] and [Voudouris and Tsang, 1999].

Relevant literature

- ▶ [Voudouris, 1997] C. Voudouris. *Guided Local Search for Combinatorial Optimization Problems*. PhD thesis, Department of Computer Science, University of Essex, 1997
- ▶ [Voudouris and Tsang, 1999] C. Voudouris and E. Tsang. Guided Local Search. *European Journal of Operational Research*, 113(2):469–499, 1999

Main idea

- ▶ Dynamically **change the objective function** depending on the search history
- ▶ Changing the objective function **means**: changing the search landscape



How to change the objective function?

1. Define a set $\mathcal{I} = \{1, \dots, n\}$ 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. Introduce a **penalty value** $p_i \geq 0$ for each solution feature $i \in \mathcal{I}$
4. 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

GLS: pseudo-code

```

s ← GenerateInitialSolution()
p ← (0, ..., 0)
while termination conditions not met do
    ŝ ← LocalSearch(s, f')
    UpdatePenaltyVector(p, ŝ)
    s ← ŝ
end while

```

Penalty 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 **utility function**:

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

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

Guidelines

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

GLS: Examples

GLS applied to the TSP

- ▶ Each **edge** $e \in E$ is considered 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 best-improvement local search

Other examples (on the board)

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

Questions?

Questions?