TABU search and Iterated Local Search – *classical OR methods*

Thomas Stidsen

tks@imm.dtu.dk

Informatics and Mathematical Modeling

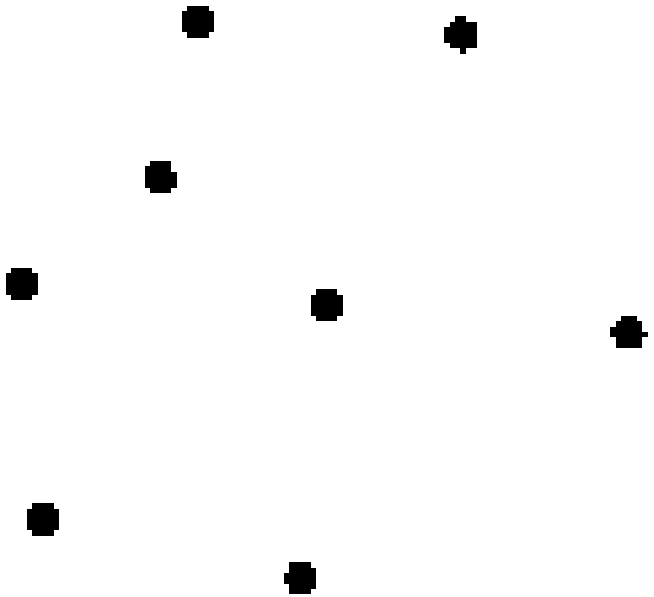
Technical University of Denmark

DTU

Outline

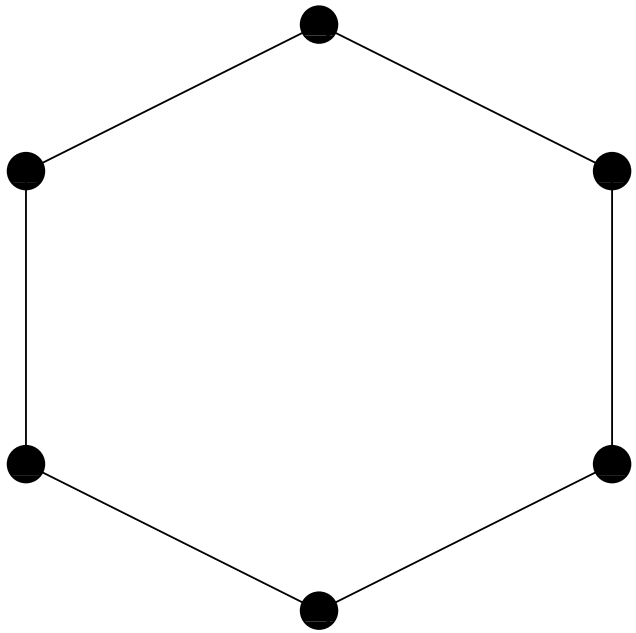
- TSP optimization problem
- Tabu Search (TS) (most important)
- Iterated Local Search (ILS)

Traveling Salesman Problem (TSP)



Objective: Find a tour of minimal length that visits all cities exactly once.

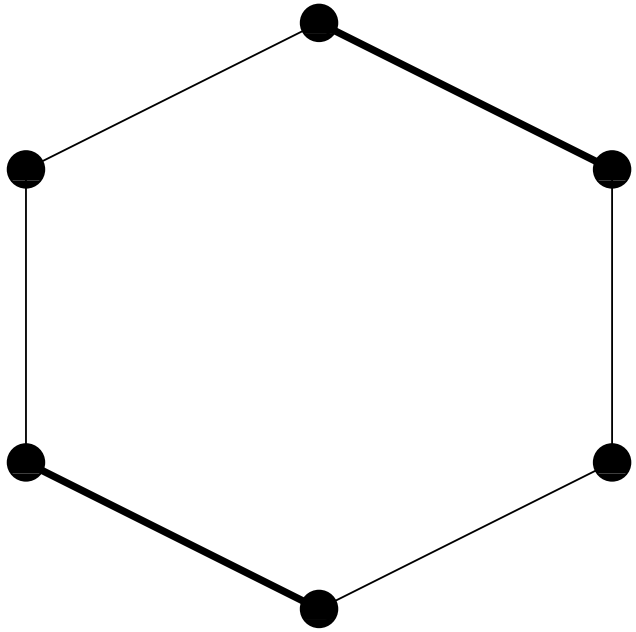
2-opt



What is the neighborhood ?

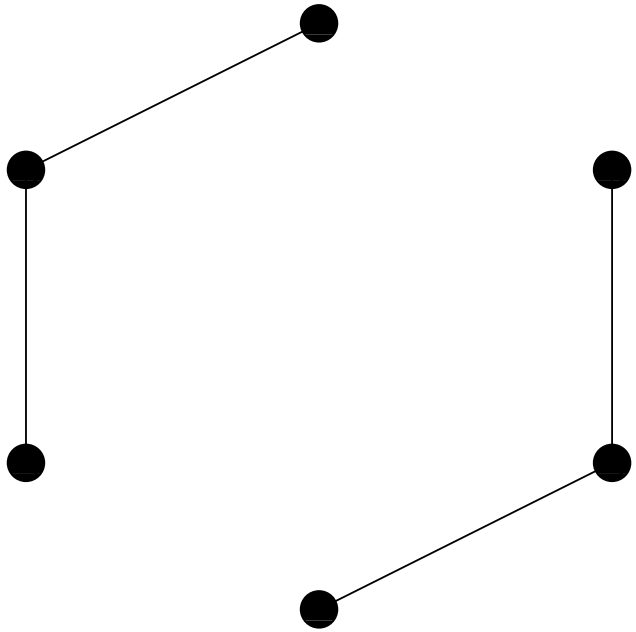
2-opt

Select two non-consecutive links:



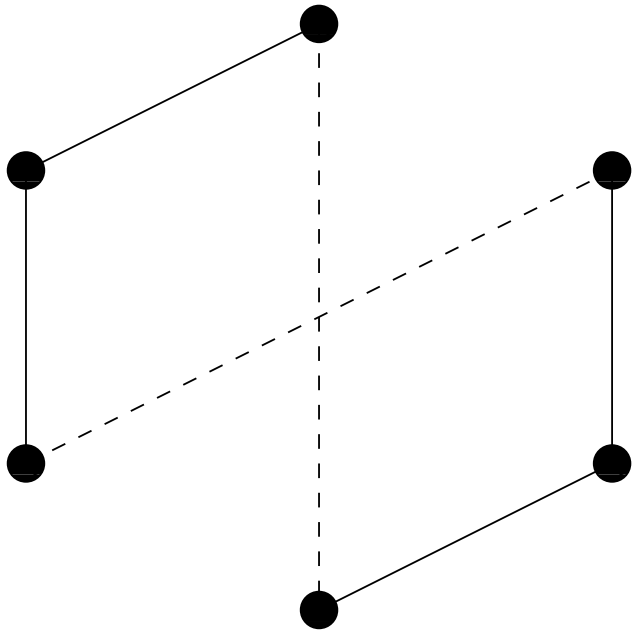
2-opt

Delete the links:



2-opt

Reconnect (only one possibility):



Local search for TSP

- Concepts:
  Solutions space S, TSP: All feasible tours visiting all cities
  Objective function $f : S \to \mathbb{R}$, TSP: Length of tour
  Neighborhood $N(s) \subset S$, TSP: All feasible tours that can be obtained by remove two edges in the tour s and replacing them with two new edges (2-opt).

TABU search

- When humans solve problems they use experience and **memory**. Try to do the same in a local search.
- Random Restart + Simulated annealing (SA) + simple ILS : No memory.
- TS: Incorporates memory.

Mountain Climbing using memory

Remember our drunk mountain climber ? Now he is sober ...

- Look all around you ...
- Choose the next step which increases your height as much as possible **not going back to a place where you were before !**

TABU search features

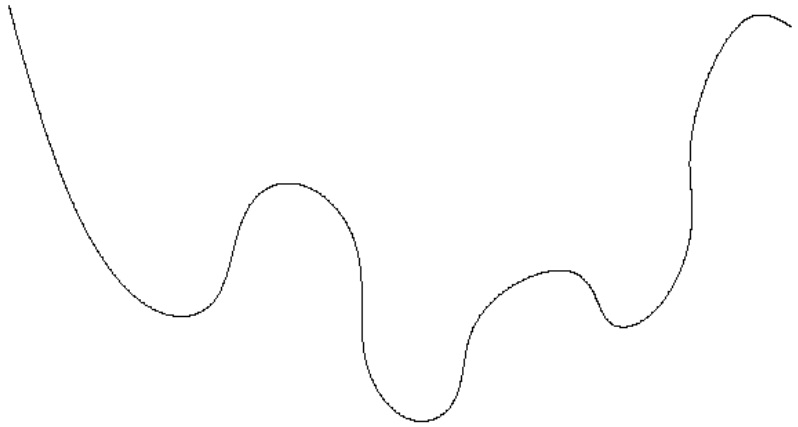
- Is difficult to analyze mathematically.
- More work is needed for designing and implementing a TS heuristic compared to a SA heuristic.
- Has been applied with success to a wide range of combinatorial optimization problems.

Tabu search history

- Roots go back to the seventies.
- Proposed in its current form by Fred Glover in 1986.
- An enourmous amount of applications and variations of TS have been proposed since then.

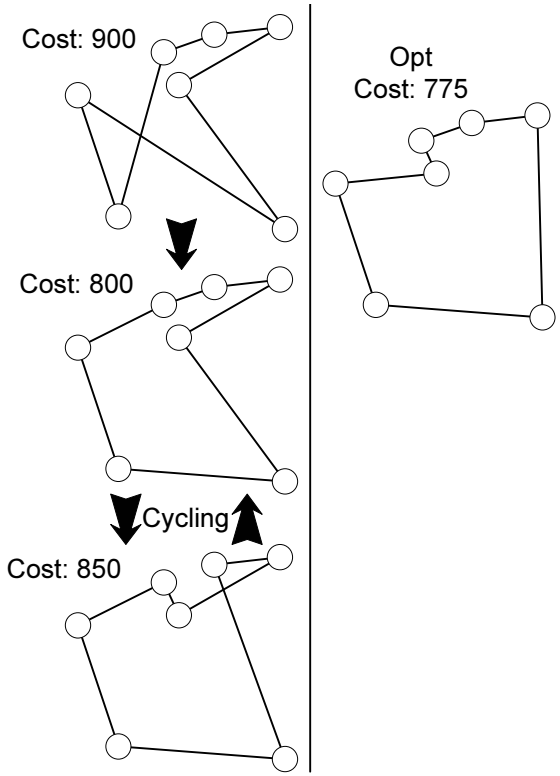
From Hill climbing (decenting) to Tabu search

- Problem with hill climbing: We are stuck in a local optimum.



- Possible solution: Allow moves that do not improve the solution or perhaps even make the solution worse.
- New problem introduced: Cycling (next slide).

Cycling



- The tabu search way of handling cycling: Remember previously visited solutions somehow.
- Why is Tabu Search called Tabu Search?

Tabu Search

Tabu search Strategies for avoiding cycling:

- Remember all solutions encountered so far (strict Tabu Search).
- Remember the last j solutions (cycles of length $\geq j+1$ are possible).
- Remember the last j moves that have been performed.
- Remember changing attributes of the last j solutions.

Basic Tabu Search algorithm

Choose initial solution s
$s^* = s$; $k = 1$;
repeat
  Generate $V \subseteq N(s,k) \subseteq N(s)$
  Choose the best s' in V
  $s = s'$
  if $f(s) < f(s^*)$ **then** $s^* = s$
  $k = k + 1$
until stopping condition is met
return s^*

Application of TABU

Points to consider when applying TS to a specific problem:

- Initial solution.
- Definition of tabu-based neighborhood $N(s,k) \subseteq N(s)$.
- Definition of candidate set $V \subseteq N(s,k)$.
- Stopping condition.

Neighbors and moves

- A neighbour $s' \in N(s)$ is obtained by applying a move m.
- A move m identifies the changes needed in order to obtain $s' = s \oplus m$.
- $M(s)$ is the set of moves that can be applied to s.

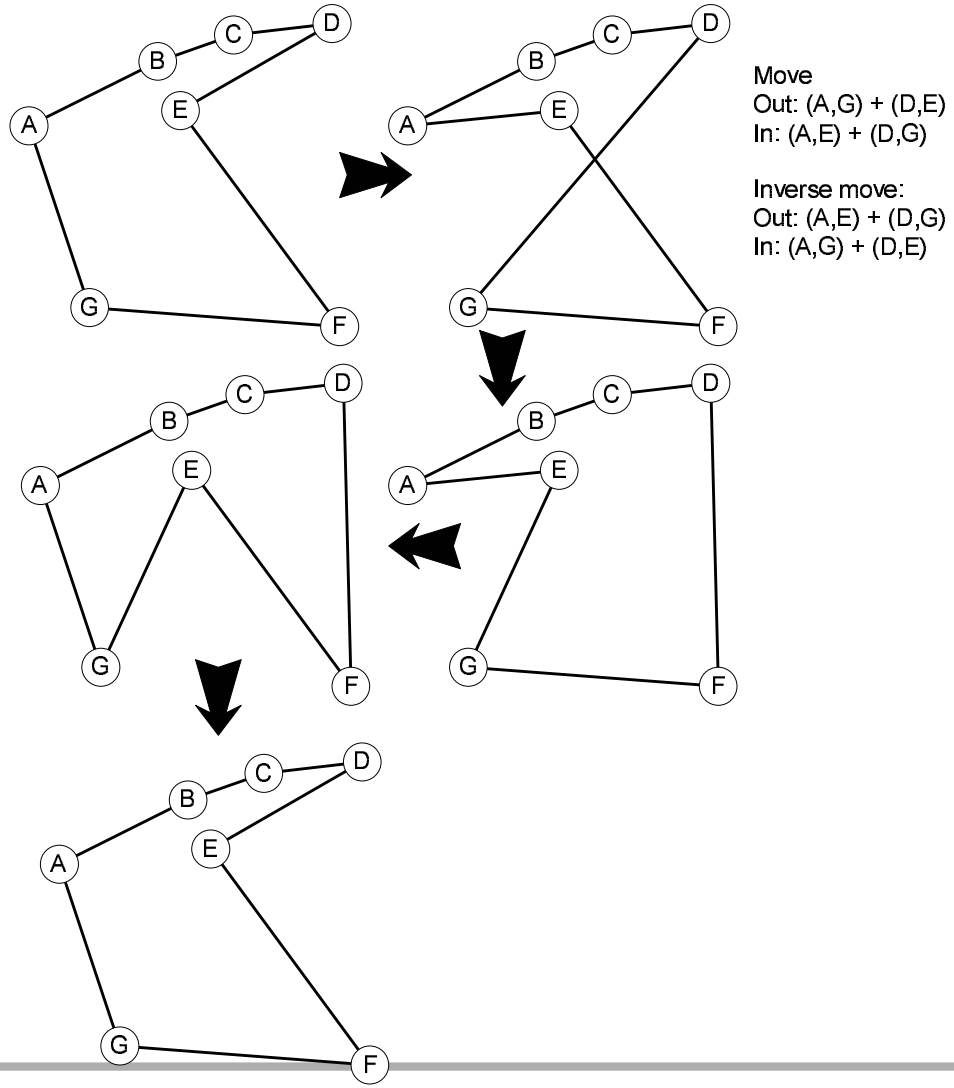
Neighbors and moves

- There is a one-to-one relationship between neighbours and moves:

$$N(s) = \{s' | \exists m \in M(s) \text{ with } s' = s \oplus m\}$$

- If every move has an inverse then we can define tabus using moves. When a move has been performed, its inverse is tabu in the following iterations.
- Even if the inverse move is tabu for j iterations cycling with length less than j can occur (see next slide).

Inverse moves and cycling



- Cycling occurs even though inverse moves never are used

Tabu tenure

- number of iterations that a solution/move/attribute is tabu
- may be static or dynamic — depends on problem type and instance size
- Dynamic tabu tenure.

Static tabu tenures are the easiest to implement.

Reactive tabu search

Vary tabu tenure according to search history: increase tabu tenure when the same solution is visited over and over again. Decrease tabu tenure when no duplicate solutions have been spotted for some time.

Aspiration criterion

- condition for overriding tabu status
- *global* best: neighbour is a new best solution (this is the most common asp. criterion).
- *region* best: neighbour is a new best among solutions having a certain property.
- *recent* best: neighbour is the best among the most recently visited solutions.
- *attribute* based: store best objective value seen for each attribute. Similar to *region best*.

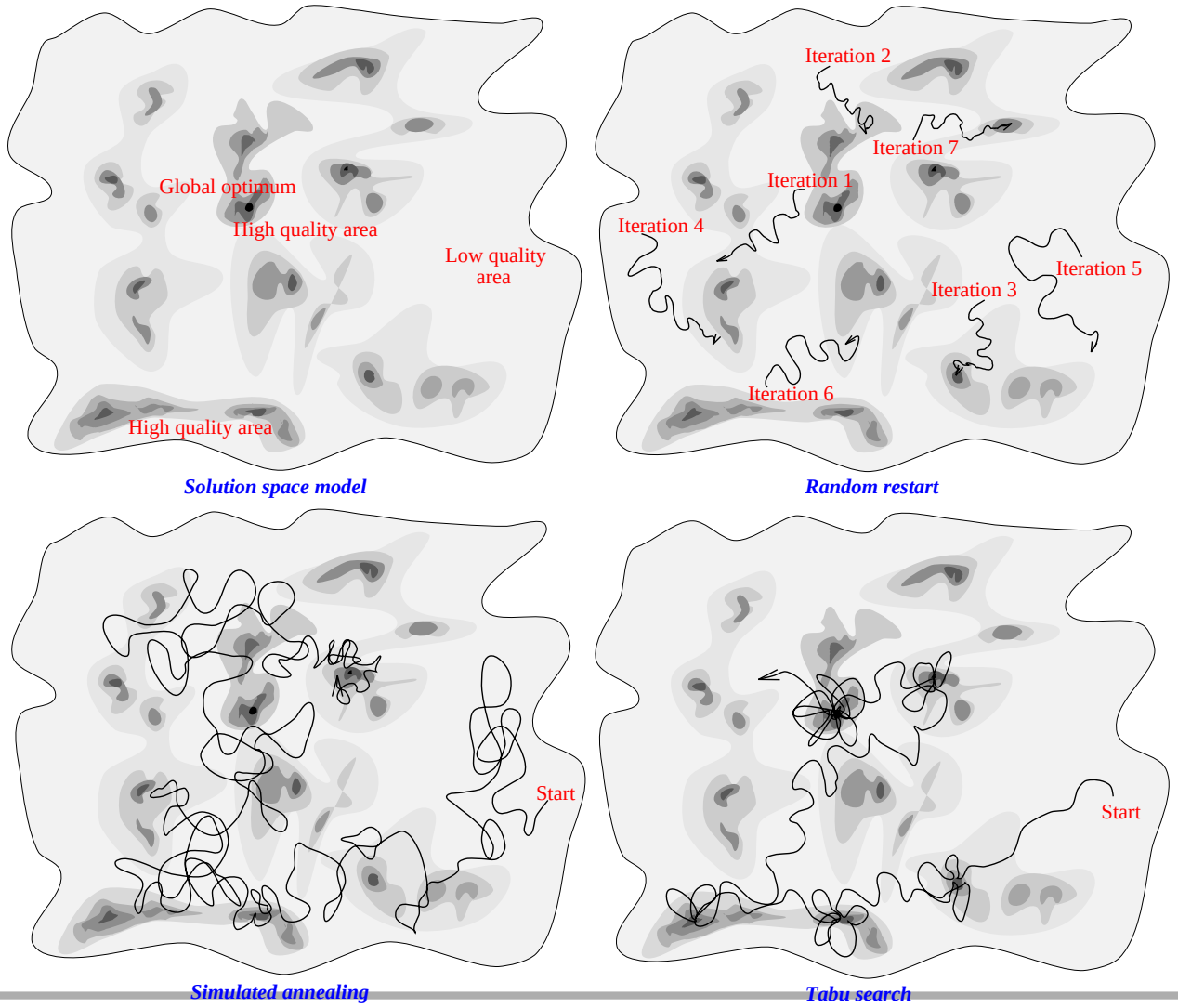
Tabu search memory: Short-term

Tabu lists which help to escape local optima and to make the search avoid recently visited solutions.

Tabu search memory: Long-term

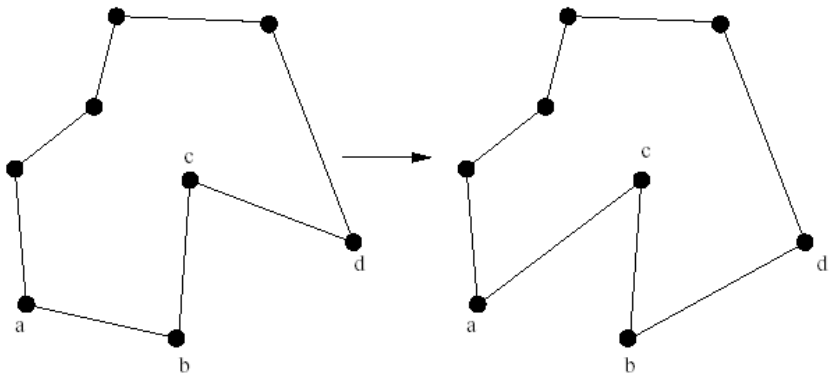
- Intensification: Information on elite solutions which extract the “core” of good solutions and direct the search according to this.
- Diversification: Information on the whole search trajectory which drives the search into new unexplored areas (diversification).
- For many problems the long term memory can be the difference between a mediocre metaheuristic and a state of the art meteheuristic. Unfortunately it is difficult to set up a general framework for how to use long-term memory - it is problem dependent.

Intuitive comparison of local search methods



Tricks for quickly searching the neighborhood

- The process of evaluating the neighborhood can often be speeded up by using Δ-computations:



Tricks for quickly searching the neighborhood

- Sometimes the cost of a move in iteration t can be computed faster by using the cost of the move in iteration $t-1$ and modifying this cost according to the move performed in iteration $t-1$.
- It can occasionally be profitable to search the neighborhood in a special order and using information from the last move evaluation to calculate the cost of the current move

TS Overview

- Pros:
  - Generates good solutions quickly.
  - Many examples of TS algorithms yielding state of the art results for many problems.
  - The heuristic has been applied to many problems areas. When faced with a new problem it is easy to find a paper that describes a tabu search algorithm for a similar problem (to get inspired).
  - There are many ways to twist and tune the heuristic - helps you to get good results.

TS Overview

- Cons:
  - There are many ways to twist and tune the heuristic - this is time consuming.
  - For many applications you have to come up with some special tricks to ensure that the tabu search does enough diversification (probably using long term memory). The tabu list is not always enough.
  - May not be the ideal metaheuristic when the neighbourhood is to large to be searched exhaustively.
  - Enormous amount of literature about TS. The quality of some papers is questionable.

TS for Christmas Lunch I

So how does TS work for the Christmas Lunch Problem ? In TABU.java:

- The neighborhood: Run through a triple for loop: For every table and every seat at that table, for every other table, for every seat at that other table: Try to swap the two persons. If new value is better and non-tabu, save this swap.
- Add tabu
- Check if best has been improved.
- This version is SLOW: DeepClone and tabu solutions are also evaluated ...

TS for Christmas Lunch II

DataObject.java:

- The matrix representation is chosen. This makes the swap operation very fast.
- The evaluation function takes time ...

Iterated Local Search (ILS)

- Descent algorithm maps from the set of solutions S to the set $S^* \subseteq S$ of solutions localy optimal w.r.t. the chosen neighborhood.
- The set S^* is usually much smaller than the set S.
- It seems profitable to search S^* instead of S.
- We would expect to find better solutions by randomly sampling S^* compared to randomly sampling S.

How to search S^*?

- Simplest approach: Use random restart:
- Problem: we tend to get solutions close to the mean of S^*.
- Better approach: ILS:
  1. Generate initial solution.
  2. Perform local search to reach local optima.
  3. Modify the current solution somehow.
  4. Iterate step 2 - 3 as desired.

ILS

```
proc IteratedLocalSearch

   s_0 = GenerateInitialSolution;
   s* = LocalSearch(s_0);
   repeat

      s_current = Perturb(s*,history);
      s*_current = LocalSearch(s_current);
      s* = Accept(s*, s*_current, history);
   until termination condition met
end
```

ILS comments

- Note: *Accept* operates on the solution obtained after pertubation AND local search, not right after the permutation.
- First used on a location problem by Baxter in 1981 and on the TSP by Baum in 1986 according to Thomas Stützle.
- The name Iterated Local Search was first invented much later (in the 90s).

Important points

- *Perturb* (Diversification)
  - Make changes that no easily are undone by local search (perturbation should be stronger or different from local search moves).
  - Perhaps use history to determine the strength of the pertubation.
- Pertubation can sometimes be seen as ruining part of the solution (dropping bombs!)

Important points

- *LocalSearch* (Intensification)
  - Tradeoff between speed and strength of local search.
  - Use succesfull local search procedure from scientific literature if available.
- *Accept*
  - Simple: only accept improvements or accept all solutions.
  - More advanced: accept certain deteriorating solutions, use history.

Example ILS applied to TSP

- Local seach: 3-opt
- Perturb: A special 4-opt move called *double-bridged move*. Is not easy to undo using simpler moves.
- Accept: Accept only improving solutions

ILS applied to TSP

- *Accept* = *Better :* Accept only improving solutions.
- *Accept* = *RW* : Random walk (in S^*), accept all solutions.

instance	Δ_{avg}(RR)	Δ_{avg}(RW)	Δ_{avg}(Better)
kroA100	0.0	0.0	0.0
d198	0.003	0.0	0.0
lin318	0.66	0.30	0.12
pcb442	0.83	0.42	0.11
rat783	2.46	1.37	0.12
pr1002	2.72	1.55	0.14
pcb1173	3.12	1.63	0.40
d1291	2.21	0.59	0.28
fl1577	10.3	1.20	0.33
pr2392	4.38	2.29	0.54
pcb3038	4.21	2.62	0.47
fl3795	38.8	1.87	0.58
rl5915	6.90	2.13	0.66

Time limit: 120 sec on PII-266 Mhz (experiments by Lorenco, Martin and Stützle)

Example ILS applied to TSP

Speed:

instance	$\#LS_{RR}$	$\#LS_{1\text{-DB}}$	$\#LS_{5\text{-DB}}$
kroA100	17507	56186	34451
d198	7715	36849	16454
lin318	4271	25540	9430
pcb442	4394	40509	12880
rat783	1340	21937	4631
pr1002	910	17894	3345
pcb1173	712	18999	3229
d1291	835	23842	4312
fl1577	742	22438	3915
pr2392	216	15324	1777
pcb3038	121	13323	1232
fl3795	134	14478	1773
rl5915	34	8820	556

Time limit: 120 sec on PII-266 Mhz (experiments by Lorenco, Martin and Stützle)

ILS Overview

Pros:

- Easy to implement a simple ILS. Usually good descent algorithms already exists.
- Compared to SA it quickly gives you good solutions and gradually improves upon it.
- For some problems it yields state of the art results.

ILS Overview

Cons:

- It may require a deep understanding of the problem and a lot of trial and error to come up with a good pertubation method.
- If we are using a bad pertubation method we might either keep returning to the same local optima or our metaheuristic may resemble random restart.
- May not be the ideal metaheuristic when the neighbourhood is to large to be searched exhaustively.
- No usefull theory