

On Variable Neighborhood Search, Iterated Local Search, and Further Metaheuristics

Olivier Martin

University of Paris-Sud at Orsay, France

Combinatorial Optimization

Formulation of the Problem:

- Consists of (R, C) , where
 - R is a set of (feasible) « solutions »
 - $C : R \rightarrow \mathfrak{R}$, is a cost function

❖ Given (R, C) , find $s^* \in R$, such that

$$C(s^*) = \min_{s \in R} \{ C(s) \}$$

Example: Traveling Salesman Problem (TSP)

Given n cities, and distance matrix $[d_{ij}]$

To find: shortest tour of n cities (visit each city exactly *once*)

$R = \{ \text{all cyclic permutations } \pi \text{ of the } n \text{ cities} \}$

$$C(\pi) = \sum_{j=1}^n d_{j\pi(j)}$$

Coping With NP-Hardness

Q. Suppose I need to solve an NP-hard problem. What should I do?

A. Theory says you're unlikely to find poly-time algorithm.

Must sacrifice at least one of three desired features:

- Solve problem to optimality.
- Solve problem in polynomial time.
- Solve arbitrary instances of the problem.

Resort to (meta-) heuristic algorithms

Incomplete (meta)heuristics

- Set variables one at a time (« construction »)
- *Iterative improvement (of a single solution)*
 - Typically by proposing small changes
- Iterative improvement (population based)
 - May propose larger changes
- Going beyond working with feasible solutions
 - Ant colony optimization
 - Message passing techniques (belief propagation)
 - Neural networks
 - Elastic band and other analogies

Example of Ant Colony Optimisation

(Main contributors: Dorigo, Colorni, Maniezzo)

- A broad term for a number of population-based meta-heuristics such as Ant Colony System (ACS)
- A relatively popular meta-heuristic based on the foraging behavior of ant colonies
- Agents (ants) construct solutions, depositing *pheromone* on elements they choose to indicate the utility of those elements
 - Each decision is based on simple heuristic information as well as the pheromone deposited by earlier ants
- From ants and their pheromone levels, one constructs a solution; ants are then killed off and new ones brought in



Our focus from now on:
*Neighborhood Based Search
to improve a feasible solution*

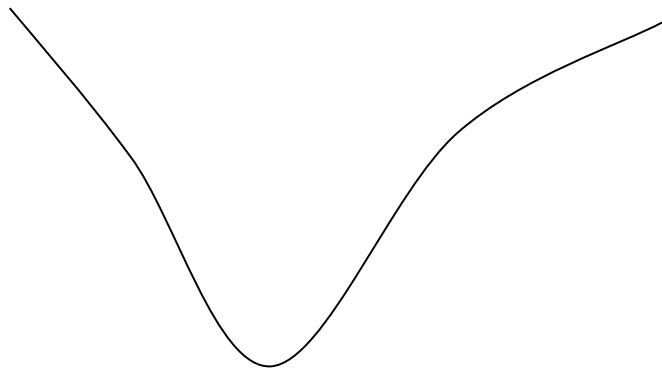
```
Generic Neighborhood Search;  
begin  
  s ← initial solution s0;  
  repeat  
    choose  $s' \in N(s)$  ;  
    s ← s' ;  
  until (Terminating condition) ;  
end;
```

Local Search: using a local neighborhood

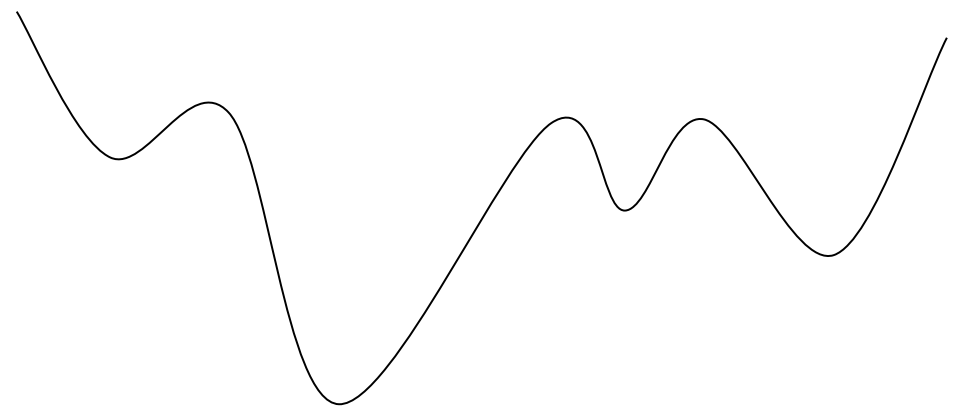
Local search. Algorithm that explores the space of possible solutions in sequential fashion, moving from a current solution to a "nearby" one.

Neighbor relation. Let $S \sim S'$ be a neighbor relation for the problem.

Descent. Let S denote current solution. If there is a neighbor S' of S with strictly lower cost, replace S by S' . Otherwise, terminate the algorithm.



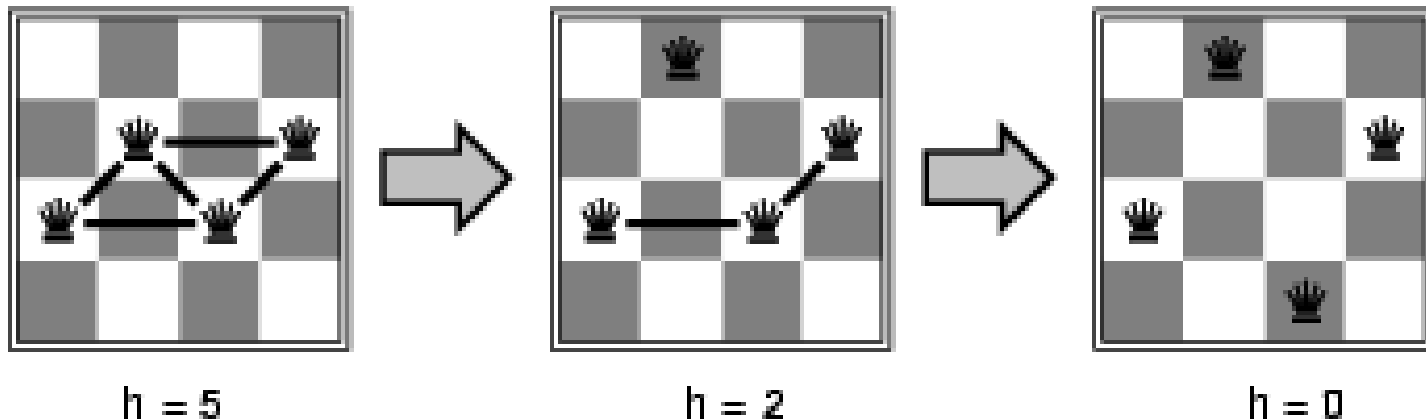
A funnel



A jagged funnel

Example: four queens

- States: 4 queens in 4 columns (256 states)
- Operators: move queen in column
- Goal test: no threats
- Cost function: = number of threats



Neighborhood Search (history...)

- Neighborhood Search dates back a long way
 - Simplex Algorithm for LP
 - Viewed as a local search technique
 - 2 and 3-opt for TSP (Lin, 1965)
- Some bells and whistles
 - Variable depth search (Lin-Kernighan)
 - Tabu (Kernighan-Lin, Cvijovic et al., Glover et al.)
 - stochasticity (Simulated Annealing etc.)
 - Including learning heuristics etc.
- Use of hierarchical neighborhoods
 - Variable neighborhood descent/search
 - Iterated local search

Local search design issues

- Solution Representation
- Initial Solution (*where to start*)
- Neighborhood(s) (*strength & size*)
- Cost Function (*viz-a-viz objective function*)
- Search strategy (*how to move*)
- Refinements
- Termination (*when to stop*)

How to move downhill?

Even if getting the global optimum is beyond reach, try to be state of the art...

- ◆ Descend the landscape greedily?
- ◆ Descend the landscape stochastically?
- ◆ What to do with *plateaus*?
- ◆ How to overcome ruggedness? The neighborhood definition is essential!

Local Search and improvements

- A local search procedure looks for a “*better or as good solution*” that is “*near*” another solution
 - by repeatedly making local changes to current soln
 - until no further candidate solutions are proposed
- Local search procedures introduce a **given** neighborhood structure that is generally problem specific (otherwise poor performance)
- To improve beyond this basic scheme, one can use a modified procedure to get out of a local minimum (allow for noise in move acceptance criteria, tabu rules, adaptive neighborhood, etc).
These can be thought of as ways to enlargen the neighborhood.

Example of Simulated Annealing

- Noise model based on statistical mechanics
 - Introduced as analogue to physical process of growing crystals
 - Cerny 1985, Kirkpatrick et al. 1983; Metropolis et al. 1953
- Convergence:
 - With appropriate slow schedule, will converge to global optimum
 - In practice, settle for « effective » approach using an exponential schedule and get good local optimum.
- Key aspect: upwards / sideways moves
 - Expensive, but (if have enough time) can be near best
- Hundreds of papers/year;
 - Many applications: VLSI layout, factory scheduling, . . .
- Quoting S. Kirkpatrick: « SA is the second best method for almost all problems », and it works well even if no problem specific information has been incorporated.

Example of an algorithmic definition of an extended neighborhood for Max Cut

1-flip neighborhood. (A, B) and (A', B') differ in **exactly one** node.

k-flip neighborhood. (A, B) and (A', B') differ in **at most k** nodes.

- $\Theta(n^k)$ neighbors.

cut value of (A_1, B_1) may be worse than (A, B)



KL-neighborhood. [Kernighan-Lin 1970]

- To form neighborhood of (A, B) :
 - Iteration 1: flip node from (A, B) that results in best cut value (A_1, B_1) , and mark that node.
 - Iteration i : flip node from (A_{i-1}, B_{i-1}) that results in best cut value (A_i, B_i) among all nodes not yet marked.
- Neighborhood of $(A, B) = (A_1, B_1), \dots, (A_{n-1}, B_{n-1})$.
- Neighborhood includes some very long sequences of flips, but without the computational overhead of a k-flip neighborhood.
- Practice: powerful and useful framework.
- Theory: explain and understand its success in practice.

Example of Tabu Search

(Main contributors: Fred Glover & Manuel Laguna)

- Tabu Search (TS) is a collection of techniques designed to provide a robust yet widely applicable criteria to modify the move rules in a local search algorithm
- The key aspect is ***adaptive memory***
- Simplest form of this:
 - If the search has already visited a point in the search space, why waste computation time revisiting and re-evaluating that point?
 - Such a point should really be made “tabu” (or taboo)
 - In high dimensional spaces, must consider features of the points rather than the points themselves and use a dynamic tabu table with erasable memory

Next: meta-heuristics using local search as an inside search engine

Think of local searches as black box procedures that you cannot touch (they are too complex, e.g., Lin-Kernighan).

How can one use one or more local searches to obtain a « high-level » general purpose meta-heuristic?

Or more bluntly, how to « assemble » better search tools with almost no conceptual or time effort?

We want general principles for assembling such meta-heuristics and can assume that they should perform well if the local search is well adapted to the problem of interest

Next, two such meta-heuristics, **VNS** and **ILS** which can also be thought of as exploiting hierarchical levels of the cost landscape

The VNS Meta-heuristic

Variable Neighbourhood Search (VNS) is a metaheuristic based on systematically changing of neighborhood set.

Usual heuristic searches are based on transformations of solutions that determine **one neighborhood structure** on the solution space.

- A **neighborhood structure** in a solution space Z is a mapping from a point to a set:

$$N: Z \rightarrow 2^Z. \quad x \rightarrow N(x).$$

- The elements $y \in N(x)$ are the **neighbor** solutions of x and constitute its neighborhood $N(x)$.
- VNS uses a **series of neighborhoods** N_k , $k = 1, \dots, k_{max}$.
- A local search takes a better neighbor while possible.
- **The basic idea of the VNS is to change the neighborhood used when the local search is trapped at a local minimum.**

A simple case: VND

(Variable Neighborhood Descent)

- Given a neighborhood structure, an improving **local search** iteratively seeks for a better solution in the neighborhood of the current solution; therefore it is *trapped* in a local optimum with respect to the current neighborhood structure.
- The **VND** (**Variable Neighborhood Descent**) method changes to using N_{k+1} each time the current N_k gets stuck.
- It ends when no further improvement (thus at k_{max})
- The final solution provided by the algorithm will be a local optimum with respect to all k_{max} neighborhoods.

The VNS algorithm

- Initially it takes $k \leftarrow 1$ and a random initial solution x .
 - A local search from x with the neighborhood N_1 returns x^* .
 - These steps are repeated until the stopping condition is met.
 - « Shaking »: take $k \leftarrow k+1$ and a random neighbor y of the local optimum x^* using the neighborhood N_k ; i.e., $y \in N_k(x^*)$.
 - Apply the local search from y using the neighborhood N_1 .
 - If the local minimum found y^* is better than x^* then the process is iterated with $x^* \leftarrow y^*$ and set $k = 1$.
 - Otherwise, iterate the process with the current solution x^* .
- When $k = k_{max}$, stop, or just restart with a new initial solution.

[Hansen & Maldenovic 2001,2003,2004]

General VNS

- The GVNS (*General Variable Neighborhood Search*) method applies two (possibly different) series of neighborhoods:
 - one for the shaking and one for the descent.
- The neighborhoods are often nested and based on a single standard move.

Given a base neighborhood structure $N: Z \rightarrow 2^Z$, the series of nested neighborhood structure is defined as follows.

- The first neighborhoods are the basic ones; $N_1(x) = N(x)$.
- and the next neighborhoods are defined recursively by $N_{k+1}(x) = N(N_k(x))$.

$N_k(x)$ consists of the solutions that can be obtained from x by k base moves.

The Nested VNS

- **Initialization:**

- Find an initial solution x .
- Set $x^* \leftarrow x$ and $k \leftarrow 1$.

- **Iterations:**

Repeat the following until the stopping condition is met:

- **Shake:**

Apply k random moves to the solution x to get x' .

- **Local Search:**

Apply an improving move to the solution x' until a local minimum x'' is found.

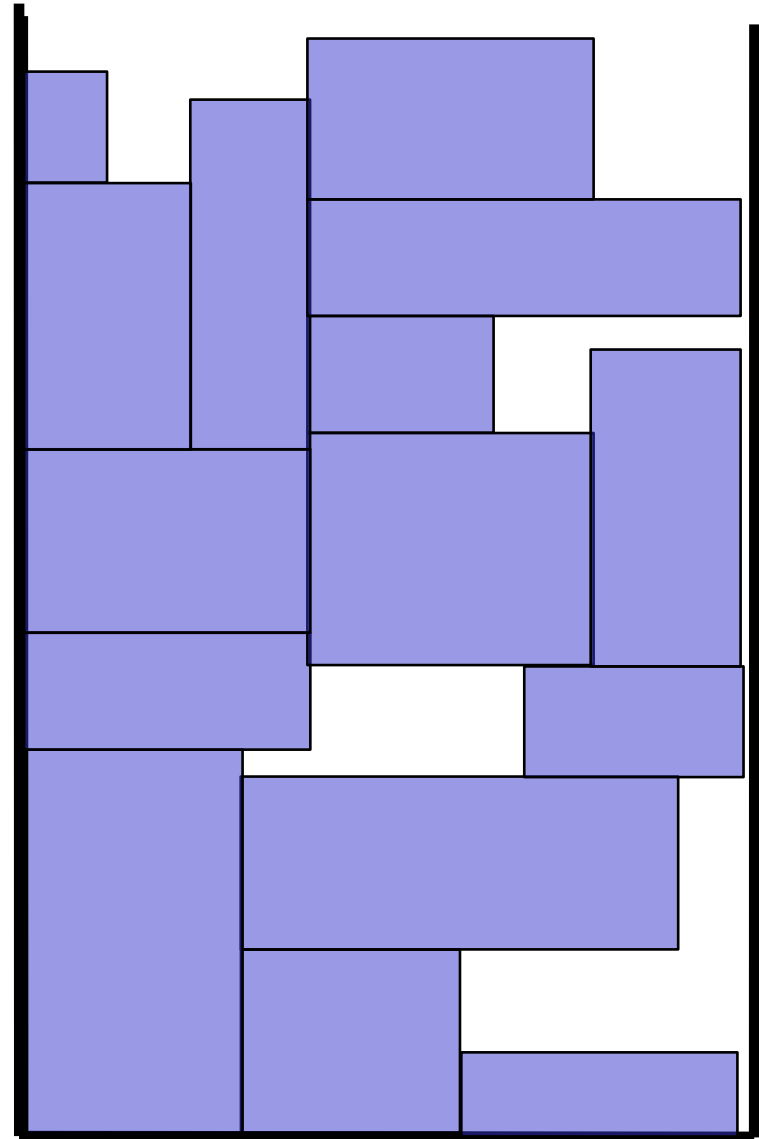
- **Improve or not:**

If x'' is better than x^* , do $x^* \leftarrow x''$ and $k \leftarrow 1$.
Otherwise do $k \leftarrow k+1$. Set $x \leftarrow x^*$.

VNS study on strip packing (J.D. Beltrán et al. 2004)

The Strip Packing Problem (SPP) consists of packing a set of rectangles in a strip so as to minimize the height of the packing

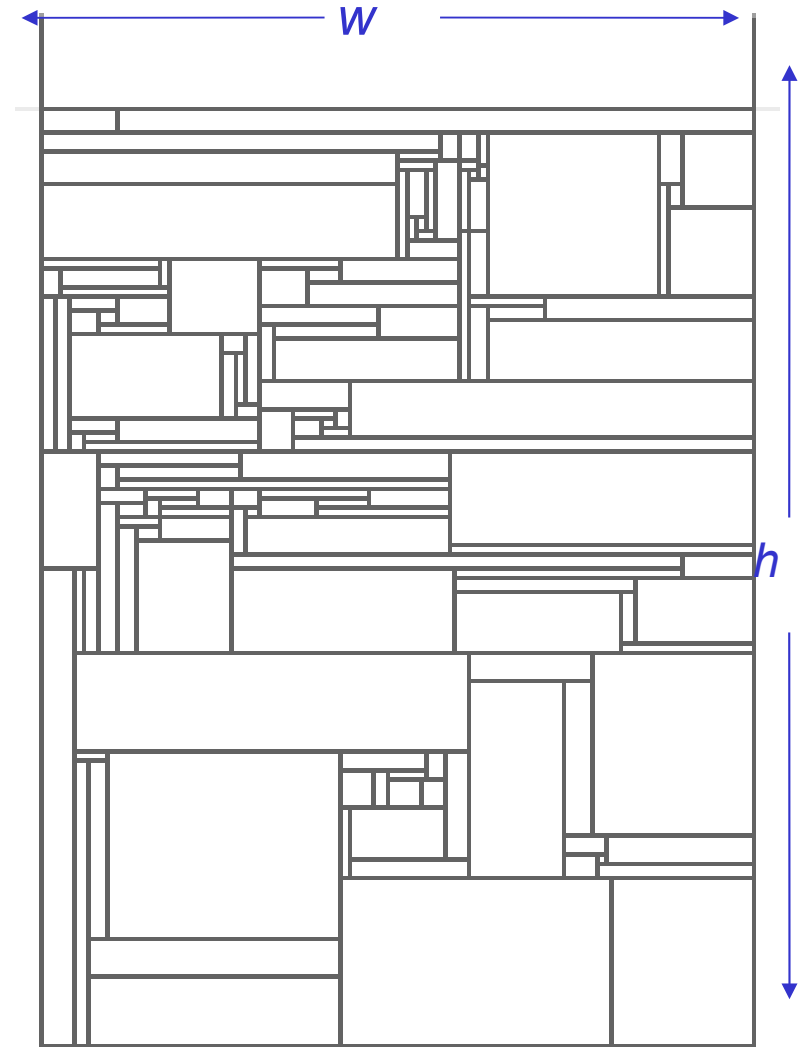
- the rectangles can be rotated by 90° .
- the cuttings can be non guillotine.
(a cutting is guillotine if it goes from a side of the object to the front side; in a non-guillotine cut this can not be true)



They used Grasp followed by VNS

VNS operates on a part of the solution given by GRASP.

VNS attempts to improve the packing of the last rectangles introduced into the solution.



Analysis on benchmarks

The instances are from
Hopper and Turton
(2002)

Category	n	w	h_{opt}
C_1	16	20	20
	17	20	20
	16	20	20
C_2	25	40	15
	25	40	15
	25	40	15
C_3	28	60	30
	29	60	30
	28	60	30
C_4	49	60	60
	49	60	60
	49	60	60
C_5	73	60	90
	73	60	90
	73	60	90
C_6	97	80	120
	97	80	120
	97	80	120
C_7	196	160	240
	196	160	240
	197	160	240

Iterated Local Search

Outline

1. ILS in a nutshell
2. Historical development
3. ILS Implementation
 - initial solution
 - perturbation
 - acceptance criterion
 - local search
 - optimization
4. ILS Applications
5. Conclusions

ILS — principle

local search

- given some search space \mathcal{S} of candidate solutions and a local search algorithm **LocalSearch**
- **LocalSearch** defines a mapping from a large search space \mathcal{S} to a much smaller space \mathcal{S}^* , $\mathcal{S}^* \subseteq \mathcal{S}$

central idea of ILS

- sample \mathcal{S}^* , generating a chain of candidate solutions by iterating over **LocalSearch**
 - \mathcal{S}^* is much smaller than \mathcal{S}
 - candidate solutions in \mathcal{S}^* are on average much better than those in \mathcal{S}
 - avoids disadvantages of “random restart”

ILS — principle

ILS builds this chain by looping through three steps

- 1 perturb s^* , leading to some intermediate solution s'
- 2 apply **LocalSearch** to s' , leading to some $s^{*'}$
- 3 apply an acceptance test to decide whether to continue from s^* or $s^{*'}$; go to 1

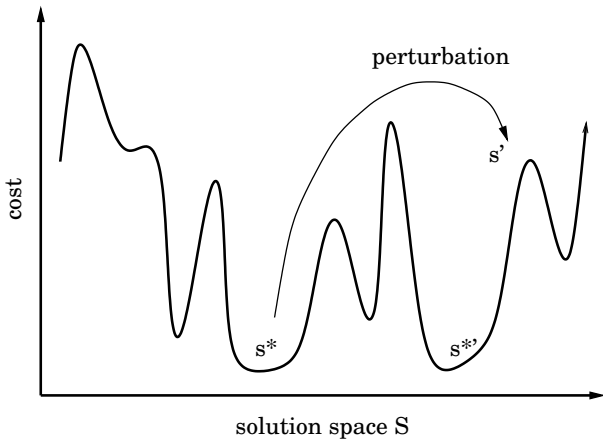
ILS — principle

ILS builds this chain by looping through three steps

- 1 perturb s^* , leading to some intermediate solution s'
- 2 apply **LocalSearch** to s' , leading to some $s^{*'}$
- 3 apply an acceptance test to decide whether to continue from s^* or $s^{*'}$; go to 1

*leads to a biased randomized walk in S^**

I LS — pictorial view



ILS — algorithmic outline

Outline

ILS in a
nutshellHistorical
developmentILS Implemen-
tationinitial solution
perturbation
acceptance
criterion
local search
optimizationILS
Applications

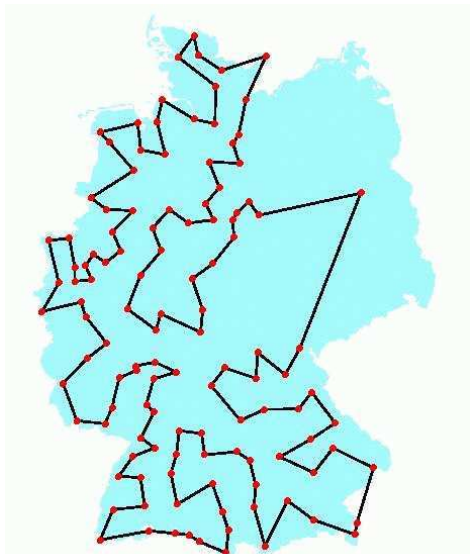
Conclusions

procedure *Iterated Local Search* $s_0 \leftarrow \text{GenerateInitialSolution}$ $s^* \leftarrow \text{LocalSearch}(s_0)$ **repeat** $s' \leftarrow \text{Perturbation}(s^*, \text{history})$ $s^{*'} \leftarrow \text{LocalSearch}(s')$ $s^* \leftarrow \text{AcceptanceCriterion}(s^*, s^{*'}, \text{history})$ **until** *termination condition met***end**

ILS — example, TSP

Traveling Salesman Problem (TSP)

- *given*: fully connected, weighted Graph
 $G = (V, E, d)$
- *goal*: find shortest Hamiltonian cycle
- *hardness*: \mathcal{NP} -hard
- *interest*: standard benchmark problem for algorithmic ideas



basic ILS algorithm for TSP

- GenerateInitialSolution: greedy heuristic
- LocalSearch: 2-opt, 3-opt, LK, (whatever available)
- Perturbation: double-bridge move (a specific 4-opt move)
- AcceptanceCriterion: accept s^{*} only if $f(s^{*}) \leq f(s^*)$

ILS — example, QAP

Quadratic Assignment Problem (QAP)

- *given*: n objects and n locations with
 - a_{ij} : flow from object i to object j
 - d_{rs} : distance between location r and location s
- *goal*: find an assignment (i.e. a permutation) of the n objects to the n locations that minimizes

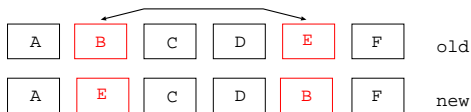
$$\min_{\pi \in \Pi(n)} \sum_{i=1}^n \sum_{j=1}^n a_{ij} d_{\pi(i)\pi(j)}$$

$\pi(i)$ gives location of object i

- *interest*: it is among the “hardest” combinatorial optimization problems; several applications

basic ILS algorithm for QAP

- GenerateInitialSolution: random initial solution
- LocalSearch: 2-opt



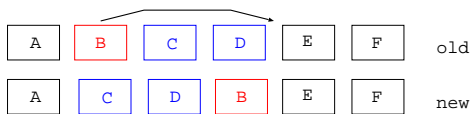
- Perturbation: random k -opt move, $k > 2$
- AcceptanceCriterion: accept s^{*} only if $f(s^{*}) \leq f(s^*)$

Permutation flow shop problem (PFSP)

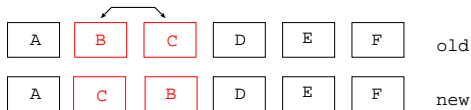
- *given*:
 - n jobs to be processed on m machines
 - processing times t_{ij} of job i on machine j
- machine order for all jobs is identical
- permutation FSP: same job order on all machines
- usual basic assumptions (available at zero, no preemption etc.)
- *goal*: minimize completion time C_{\max} of last job (makespan) — or any other
- *interest*: prototypical scheduling problem, \mathcal{NP} -hard

basic ILS algorithm for PFSP

- GenerateInitialSolution: NEH heuristic
- LocalSearch: insertion neighborhood



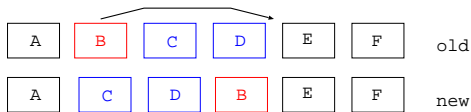
- Perturbation: a number of *swap*- or interchange moves



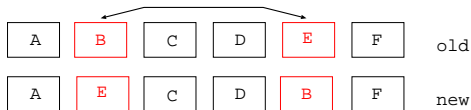
- AcceptanceCriterion: accept s^{*} ' only if $f(s^{*}') \leq f(s^{*})$

basic ILS algorithm for PFSP

- GenerateInitialSolution: NEH heuristic
- LocalSearch: insertion neighborhood



- Perturbation: a number of swap- or *interchange* moves



- AcceptanceCriterion: accept s^{*} ' only if $f(s^{*}') \leq f(s^{*})$

Iterated Local Search

*Iterated local search (ILS) is an SLS method that generates a **chain** of solutions returned by an **embedded heuristic**.*

- simple principle
- easy to implement
- state-of-the-art results
- long history

an extension of the ALAM local search

- first (?) ILS algorithm
- problem tackled is a simple depot location problem
- local search: adaptive location–allocation method (ALAM)
- perturbations by including hazards that perturb instance data (distances); various perturbation “strengths” used
- only improved solutions are accepted
- in experimental tests only few ILS iterations
- significant improvements over first local optimum

iterated descent

- first ILS algorithm for the TSP
- several variants of first-improvement type algorithms tested for the local search
- perturbation done by a random 2-exchange step
- only better quality tours are accepted
- performance: improvements of 2-opt and 3-opt algorithms, but in general poor performance probably because of poor choice for perturbation

Martin, Otto & Felten, 1991–1996

large-step Markov chains

- developed by Martin, Otto, Felten, LSMC was the first high-performing ILS algorithm for the TSP
- initially usage of a 3-opt algorithm, later LK
- perturbation by a double-bridge move that restricts the total weight of the four new edges to at most k times the average edge weight of the incumbent solution
- tours are accepted using the Metropolis criterion from simulated annealing; later variant, known as *chained local optimisation* (CLO) accepts only better quality solutions

iterated Lin-Kernighan

- an early variation on LSMC by Johnson (et al.)
- for quite some time the state-of-the-art SLS algorithm for TSP
- key differences to LSMC
 - initialisation with randomised variant of the greedy algorithm
 - acceptance criterion accepts only better quality solutions
 - double-bridge perturbation is completely random (does not include cost restrictions; cutpoints are chosen randomly)
 - uses a more efficient implementation of LK than LSMC plus some other, simple “tricks”

SAT_{x.y} algorithms

- first local search algorithms for SAT
- local search is a first-improvement algorithm in 1-flip neighbourhood
- once a local optimum is reached, some variables are randomly flipped
- every new local optimum is accepted
- performance: a bit unclear, probably similar to GSAT

Mühlenbein, 1991

iterated hill-climbing

- initially applied to some deceptive problems known from evolutionary algorithms' literature; candidate solutions encoded as bit-strings
- perturbations by flipping each bit with some probability
- local search algorithms tested are first-improvement and best-improvement algorithms
- only better solutions are accepted

Codenotti et al. 1993–1996

iterated local search

- first(?) to use name Iterated Local Search in their ICSI technical report from 1993
- another variation of ILS for TSP
- main characteristic is the perturbation scheme based on modifications of the instance data and re-optimizations
- remainder is a rather straightforward ILS algorithm

Further development

- increasingly developed later on
- however, still several times “re-discovered”
- iterated local search is now the most widely used name for this “old” general-purpose SLS method

Names for ILS

- no name — Baxter, 1981
- iterated descent — Baum, 1986
- large-step Markov chains — Martin et al., 1991
- iterated Lin-Kernighan — Johnson, 1990
- iterated hillclimbing — Mühlenbein, 1991
- simulated annealing — Osman, 1993
- iterated local search — Codenotti et al., 1993
- chained local optimization — Martin & Otto, 1996
- chained Lin-Kernighan — Applegate et al., 1999
- hybrid 1 + 1 evolutionary algorithm — Reinholz, 2005
- ...

ILS — algorithmic outline

procedure *Iterated Local Search*

$s_0 \leftarrow \text{GenerateInitialSolution}$

$s^* \leftarrow \text{LocalSearch}(s_0)$

repeat

$s' \leftarrow \text{Perturbation}(s^*, \text{history})$

$s^{*'} \leftarrow \text{LocalSearch}(s')$

$s^* \leftarrow \text{AcceptanceCriterion}(s^*, s^{*'}, \text{history})$

until *termination condition met*

end

ILS — basic version

basic version of ILS

- **GenerateInitialSolution**: random or construction heuristic
- **LocalSearch**: often readily available
- **Perturbation**: random moves in higher order neighborhoods
- **AcceptanceCriterion**: force cost to decrease

basic version of ILS

- often leads to very good performance
- only requires few lines of additional code to existing local search algorithm
- state-of-the-art results with further optimizations

ILS — modules

ILS is a modular approach

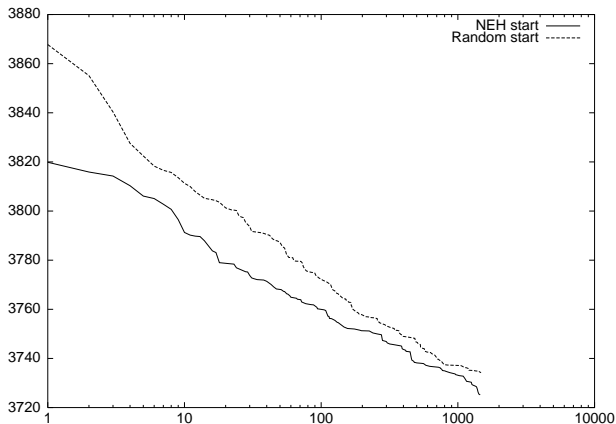
Optimization of modules

- consider different implementation possibilities for modules
 - fine-tune modules step-by-step
 - optimize single modules without considering interactions among modules
- ↪ *local optimization of ILS*

ILS — initial solution

- determines starting point s_0^* of walk in \mathcal{S}^*
- random vs. greedy initial solution
- greedy initial solutions appear to be recommendable
- for long runs dependence on s_0^* should be very low

ILS for FSP, initial solution



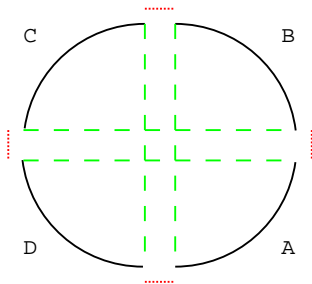
ILS — perturbation

- important: *strength* of perturbation
 - *too strong*: close to random restart
 - *too weak*: LocalSearch may undo perturbation easily
- random perturbations are simplest but not necessarily best
- perturbation should be complementary to LocalSearch

ILS — perturbation example

double-bridge move for TSP

- small perturbation
- complementary to LK local search
- low cost increase



Old:

A-B-C-D

New:

A-D-C-B

ILS — perturbation strength

sometimes large perturbations needed

- example: basic ILS for QAP

given is average deviation from best-known solutions for different sizes of the perturbation (from 3 to n); averages over 10 trials; 60 seconds on a 500MHz Pentium III.

instance	3	$n/12$	$n/6$	$n/4$	$n/3$	$n/2$	$3n/4$	n
kra30a	2.51	2.51	2.04	1.06	0.83	0.42	0.0	0.77
sko64	0.65	1.04	0.50	0.37	0.29	0.29	0.82	0.93
tai60a	2.31	2.24	1.91	1.71	1.86	2.94	3.13	3.18
tai60b	2.44	0.97	0.67	0.96	0.82	0.50	0.14	0.43

ILS — perturbation strength

Outline

ILS in a nutshell

Historical development

ILS Implementation

initial solution

perturbation

acceptance criterion

local search optimization

ILS

Applications

Conclusions

Adaptive perturbations

- single perturbation size not necessarily optimal
- perturbation size may vary at run-time; done in *basic Variable Neighborhood Search*
- perturbation size may be adapted at run-time; leads to *reactive search*

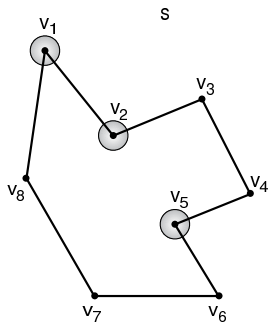
Complex perturbation schemes

- optimizations of subproblems [Lourenço, 1995]
- input data modifications
 - modify data definition of instance
 - on modified instance run LocalSearch using input s^* , output is perturbed solution s'

ILS — complex perturbation

modification of data definition

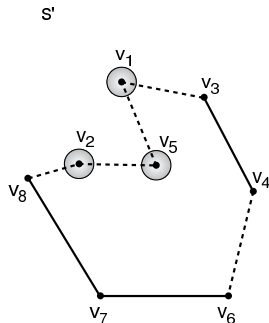
[Codonotti et al., 1993, 1996]



1. coordinate perturbation



2. local search



ILS — speed

- on many problems, small perturbations are sufficient
- LocalSearch in such a case will execute very fast; very few improvement steps
- sometimes access to LocalSearch in combination with Perturbation increases strongly speed (e.g. don't look bits)
- example: TSP

ILS — speed, example

Outline

ILS in a
nutshellHistorical
developmentILS Implemen-
tation

initial solution

perturbation

acceptance
criterionlocal search
optimization

ILS

Applications

Conclusions

- compare No. local searches of 3-opt in fixed computation time
- $\#LS_{RR}$: No. local searches with random restart
- $\#LS_{1-DB}$: No. local searches with one double bridge move as Perturbation
- time limit: 120 sec on a Pentium II 266 MHz PC

instance	$\#LS_{RR}$	$\#LS_{1-DB}$	$\#LS_{1-DB}/\#LS_{RR}$
kroA100	17507	56186	3.21
d198	7715	36849	4.78
lin318	4271	25540	5.98
pcb442	4394	40509	9.22
rat783	1340	21937	16.38
pr1002	910	17894	19.67
d1291	835	23842	28.56
f11577	742	22438	30.24
pr2392	216	15324	70.94
pcb3038	121	13323	110.1
f13795	134	14478	108.0
r15915	34	8820	259.4

ILS — acceptance criterion

Outline

ILS in a
nutshellHistorical
developmentILS Implemen-
tationinitial solution
perturbation
acceptance
criterion
local search
optimizationILS
Applications

Conclusions

- AcceptanceCriterion has strong influence on nature and effectiveness of walk in \mathcal{S}^*
- controls balance between intensification and diversification
- simplest case: Markovian acceptance criteria
- extreme intensification:
Better($s^*, s^{*'}, history$): accept $s^{*'}$ only if $f(s^{*'}) < f(s^*)$
- extreme diversification:
RW($s^*, s^{*'}, history$): accept $s^{*'}$ always
- many intermediate choices possible

ILS — acceptance criterion, TSP

example: influence of acceptance criterion on TSP

- small perturbations are known to be enough
- high quality solutions are known to cluster;
“big valley structure”
~> good strategy incorporates intensification

ILS — acceptance criterion, TSP

Outline

ILS in a nutshell

Historical development

ILS Implementation

initial solution
 perturbation
 acceptance criterion
 local search optimization

ILS Applications

Conclusions

- compare average dev. from optimum (Δ_{avg}) over 25 trials
- $\Delta_{avg}(\text{RR})$: random restart
- $\Delta_{avg}(\text{RW})$: random walk as AcceptanceCriterion
- $\Delta_{avg}(\text{Better})$: first descent in \mathcal{S}^* as AcceptanceCriterion
- time limit: 120 sec on a Pentium II 266 MHz PC

instance	$\Delta_{avg}(\text{RR})$	$\Delta_{avg}(\text{RW})$	$\Delta_{avg}(\text{Better})$
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
f11577	10.3	1.20	0.33
pr2392	4.38	2.29	0.54
pcb3038	4.21	2.62	0.47
f13795	38.8	1.87	0.58
r15915	6.90	2.13	0.66

ILS — search history

exploitation of search history

- many of the bells and whistles of other strategies (diversification, intensification, tabu, adaptive perturbations and acceptance criteria, etc...) are applicable

simplest usage of search history

- extremely simple use of history:
 $\text{Restart}(s^*, s^{*'}, history)$: Restart search if for a number of iterations no improved solution is found

ILS — QAP, example results

Outline

ILS in a nutshell

Historical development

ILS Implementation

initial solution perturbation acceptance criterion local search optimization

ILS

Applications

Conclusions

instance	accept	3	$n/12$	$n/6$	$n/4$	$n/3$	$n/2$	$3n/4$	n
kra30a	Better	2.51	2.51	2.04	1.06	0.83	0.42	0.0	0.77
kra30a	RW	0.0	0.0	0.0	0.0	0.0	0.02	0.47	0.77
kra30a	Restart	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.77
sko64	Better	0.65	1.04	0.50	0.37	0.29	0.29	0.82	0.93
sko64	RW	0.11	0.14	0.17	0.24	0.44	0.62	0.88	0.93
sko64	Restart	0.37	0.31	0.14	0.14	0.15	0.41	0.79	0.93
tai60a	Better	2.31	2.24	1.91	1.71	1.86	2.94	3.13	3.18
tai60a	RW	1.36	1.44	2.08	2.63	2.81	3.02	3.14	3.18
tai60a	Restart	1.83	1.74	1.45	1.73	2.29	3.01	3.10	3.18
tai60b	Better	2.44	0.97	0.67	0.96	0.82	0.50	0.14	0.43
tai60b	RW	0.79	0.80	0.52	0.21	0.08	0.14	0.28	0.43
tai60b	Restart	0.08	0.08	0.005	0.02	0.03	0.07	0.17	0.43

ILS — local search

- in the simplest case, use LocalSearch as black box
- any improvement method can be used as LocalSearch
- best performance with optimization of this choice
- often it is necessary to have direct access to LocalSearch (e.g. when using don't look bits)

ILS — local search

complex local search algorithms

- variable depth local search, ejection chains
- dynasearch
- variable neighborhood descent
- any other local search can be used within ILS, including *short* runs of
 - tabu search
 - simulated annealing
 - dynamic local search

ILS — local search

effectiveness of local search?

- *often*: the more effective the local search the better performs ILS
 - example TSP: 2-opt vs. 3-opt vs. Lin-Kernighan
- *sometimes*: preferable to have fast but less effective local search

ILS — local search

effectiveness of local search?

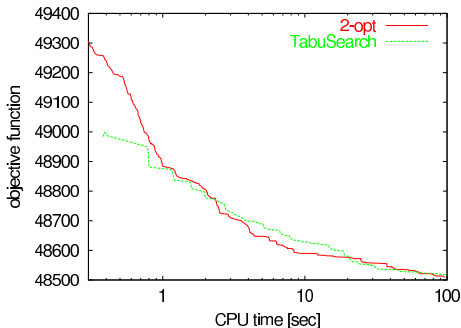
- *often*: the more effective the local search the better performs ILS
 - example TSP: 2-opt vs. 3-opt vs. Lin-Kernighan
- *sometimes*: preferable to have fast but less effective local search

the tradeoff between effectiveness and efficiency of the local search procedure is an important point to be addressed when optimizing an ILS algorithm

ILS — local search, example

tabu search vs. 2-opt, sko64

- short tabu search runs ($6n$ iterations) vs. 2-opt



ILS — optimization

optimize performance of ILS algorithms

- important to reach peak performance
- optimization goal has to be given (optimize average solution quality, etc.)
- robustness is an important issue
- *start*: basic ILS

ad-hoc optimization

- optimize single components, e.g. in the order
GenerateInitialSolution, LocalSearch, Perturbation,
AcceptanceCriterion
- iterate through this process

ILS — optimization

closer look

- optimal configuration of one component depends on other components
- complex interactions among components exist
- directly address these dependencies to perform a *global optimization of ILS performance*

needed: sound experimental methodologies

- statistical analysis!
- run-time distributions methodology [*Hoos, Stützle, 1996–*]
- racing algorithms [*Birattari et al. 2002*]
- experimental design techniques

ILS — optimization

main dependencies

- perturbation should not be easily undone by LocalSearch; if LocalSearch has obvious short-comings, a good perturbation should compensate for them.
- combination Perturbation — AcceptanceCriterion determines the relative balance of intensification and diversification; large perturbations are only useful if they can be accepted

ILS — optimization

main dependencies

- perturbation should not be easily undone by LocalSearch; if LocalSearch has obvious short-comings, a good perturbation should compensate for them.
- combination Perturbation — AcceptanceCriterion determines the relative balance of intensification and diversification; large perturbations are only useful if they can be accepted

The balance intensification—diversification is very important and achieving it is a challenging problem

ILS applications — TSP

Outline

ILS in a
nutshell

Historical
development

ILS Implemen-
tation

initial solution
perturbation
acceptance
criterion
local search
optimization

ILS
Applications

Conclusions

- iterated descent [*Baum, 1986*]
 - first approach, relatively poor results
- large step Markov chains [*Martin, Otto, Felten, 1991–1996*]
 - first effective ILS algorithm for TSP
- iterated Lin-Kernighan [*Johnson, 1990, 1997*]
 - efficient ILS implementation based on preprints of MOF91
- data perturbation [*Codenotti et.al, 1996*]
 - complex perturbation based on changing problem data
- improved LSMC [*Hong, Kahng, Moon, 1997*]
 - study of different perturbation sizes, acceptance criteria

ILS applications — TSP

Outline

ILS in a
nutshell

Historical
development

ILS Implemen-
tation

initial solution
perturbation
acceptance
criterion
local search
optimization

ILS
Applications

Conclusions

- CLO implementation in Concorde [*Applegate, Bixby, Chvatal, Cook, Rohe, 199?-today*]
 - very fast LK implementation, publicly available, applied to extremely large instances (25 million cities!)
- ILS with fitness-distance based diversification [*Stützle, Hoos 1999*]
 - diversification mechanism in ILS for long run times
- ILS with genetic transformation [*Katayama, Narisha, 1999*]
 - perturbation guided by a second solution
- plus more recent variants like Iterated Helsgaun, ILS with Helsgaun's LK variant etc.

ILS applications — routing

- vehicle routing
 - capacitated VRP [*Osman, 1993*]
 - VRPs with convex time penalty functions [*Ibaraki et al., 2003*]
 - various VRPs [*Reinholz, 199?–2005*]
- inventory routing
 - with time windows [*Lau, Liu, Ono, 2001*]
 - multi-period inventory routing [*Ribeiro, Lourençou, 2005*]

ILS applications — scheduling

Outline

ILS in a
nutshell

Historical
development

ILS Implemen-
tation

initial solution
perturbation
acceptance
criterion
local search
optimization

ILS
Applications

Conclusions

- single machine total weighted tardiness problem
 - iterated dynasearch [*Congram, Potts, Van de Velde, 1998*]
 - ILS with VND local search [*den Besten et al., 2000*]
 - ILS with dynasearch VND [*Grosso, Della Croce, Tadei, 2004*]
- single and parallel machine scheduling
 - several problems [*Brucker, Hurink, Werner 1996, 1997*]
 - parallel machine total weighted completion time [*Agarwal et al. 2004*]
- flow shop scheduling
 - permutation flow shop problem [*Stützle, 1998*]
 - flow shop problem with stages in series [*Yang et al., 2000*]
 - two-stage flow shop, secondary criterion [*Gupta et al., 2002*]

ILS applications — scheduling

Outline

ILS in a
nutshell

Historical
development

ILS Implemen-
tation

initial solution
perturbation
acceptance
criterion
local search
optimization

ILS
Applications

Conclusions

- job shop scheduling (JSP)
 - ILS with optimization of subproblems [*Lourenço 1995, Lourenço, Zwijnenburg, 1996*]
 - guided local search extensions for JSP [*Balas, Vazacopoulos 1998*]
 - total weighted tardiness job shop problem [*Kreipl, 2000*]
- timetabling
 - course timetabling, Di Gaspero and Schaerf, 2002
 - examination timetabling, Di Gaspero, 2002
 - nurse rostering, Bellanti et al., 2004

ILS applications — bioinformatics

Outline

ILS in a
nutshell

Historical
development

ILS Implemen-
tation

initial solution
perturbation
acceptance
criterion
local search
optimization

ILS
Applications

Conclusions

- phylogenetic trees, [*Nixon 1999*]
- phylogenetic trees, [*Roshan et al. 2004*]
- DNA code design, [*Tulpan, Hoos, 2004*]
- multiple sequence alignment, [*Auer, 2004*]
- protein–ligand docking, [*Korb et al. 2005*]

ILS applications — other

- graph partitioning [Martin, Otto, 1995; Fukunaga et al., 1996]
- unweighted MAX-SAT
 - reactive search [Battiti, Protasi, 1997]
 - ILS with 2- and 3-flip local search [Yagiura, Ibaraki, 2001]
- weighted MAX-SAT [Smith, Hoos, Stützle, 2002]
- quadratic assignment problem [Stützle, 1999]
- 1-dim cutting stock [Umetani, Yagiura, Ibaraki, 2003]
- graph colouring [Morgenstern, 1996], [Chiarandini, Paquete, Stützle, 2001, 2002], [Glass, Prügel-Bennett, 2005]
- learning Bayesian networks [de Campos et al., 2003]
- minimum sum-of-squares clustering [Merz, 2003]
- ...

ILS applications — more

- Various VRPs, Reinholz
- Packing, Costa et al.
- Time dep. VRPTW, Hashimoto, Yagiura & Ibaraki
- Error correcting codes, Blum & Blesa
- TSP, Fischer & Merz
- Mirrored TSPm, Araújo et al.
- FSP with seq. dep. setup times, Ruiz & Stützle (*IG*)

ILS — applications summary

world class performance

- TSP, QAP, FSP and variants, JSP, SMTWTP, MAX-SAT, phylogenetic trees, ...

performance, general

- basic version often results in quite good performance
- often, excellent performance with further optimizations

main advantage

- ease of application and its understandability
- flexibility for further optimizations

ILS — related ideas

- iterated greedy
 - underlying heuristic is a greedy construction heuristics
 - iterate over construction heuristic by (partially) destructing and then re-constructing solutions
 - often performance improves with additional local search
 - very good results for set covering, flow shop scheduling, ..
- reactive tabu search
 - one essential part of reactive tabu search are escapes from “attractors”
- basic VNS, skewed VNS
 - multiple hierarchical neighborhoods, slightly different framework
- memetic algorithms, many ACO algorithms
 - shares idea of sampling in \mathcal{S}^*

Conclusions

ILS is ..

- based on simple principles
- easy to understand
- basic versions are easy to implement
- flexible, allowing for many additional optimizations if needed
- highly effective in many applications

- application to new types of problems
 - multi-objective, dynamic, stochastic, etc.
- understanding the interaction between the ILS modules
- understanding of the reasons of success for ILS and where it it may fail
- development of a *systematic methodology* towards the bottom-up *design* and *implementation* of *high-performing SLS algorithms*
 - ILS can be a central element of it

A pedagogical reference for getting started in ILS

Helena R. Lourenço, Olivier Martin, and Thomas Stützle.
Iterated Local Search. In F. Glover and G. Kochenberger,
editors, *Handbook of Metaheuristics*, volume 57 of *International
Series in Operations Research & Management Science*, pages
321-353, Kluwer Academic Publishers, Norwell, MA, 2002.

[download](#) technical report version via my webpage

Grazie...
and a special thanks to
Thomas Stützle...

