

Iterated Local Search and Variable Neighborhood Search

Christian Blum

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

Outline

Subjects

- ▶ Iterated local search (ILS)
 - ▶ Framework
 - ▶ Implementation
 - ▶ Practice
- ▶ Variable Neighborhood Search (VNS)
 - ▶ Framework
 - ▶ Variants

Thanks

... to **Thomas Stützle** who provided many of these slides

Historical note

- ▶ **First** approaches by Baxter, 1981 to a location problem and Baum, 1986 to TSP
- ▶ First formulated as a **metaheuristic framework** by Stützle in 1998
- ▶ Like iterated-greedy (IG) techniques, ILS methods are state-of-the-art for many problems

Literature:

1. [Baxter, 1981] Local optima avoidance in depot location. *Journal of the Operational Research Society*, 32, 815–819, 1981.
2. [Stützle, 1998] Local search algorithms for combinatorial problems – analysis, improvements and new applications. PhD thesis, Darmstadt University of Technology, 1998.

What is Iterated local search?

Iterated local search (ILS)

is a metaheuristic method that generates a **sequence** of solutions generated by **local search**, leading to far better results than if one were to use repeated trials of that local search from random starting positions.

Properties

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

What is Iterated local search?

Given

- ▶ some local search algorithm: LocalSearch
- ▶ more general: any problem specific optimization algorithm

Question

- ▶ can such an algorithm be improved by iteration?

YES

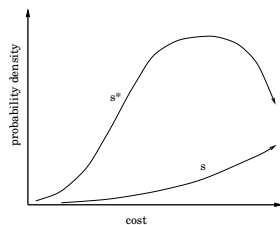
ILS — Notation

- ▶ \mathcal{S} : set of (candidate) solutions
- ▶ s : solution in \mathcal{S}
- ▶ f : cost function
- ▶ $f(s)$: cost function value of solution s
- ▶ s^* : locally optimal solution
- ▶ \mathcal{S}^* : set of locally optimal solutions
- ▶ LocalSearch defines mapping from $\mathcal{S} \mapsto \mathcal{S}^*$

Motivation

Cost distributions

Take $s \in \mathcal{S}$ or $s^* \in \mathcal{S}^*$ at random



How to go beyond LocalSearch?

Possibility: random restart

- ▶ generate multiple s^* independently
- ▶ theoretical guarantees
- ▶ practically not very effective
- ▶ for large instances leads to costs with fixed percentage excess above optimum

ILS – Principle

Basic idea

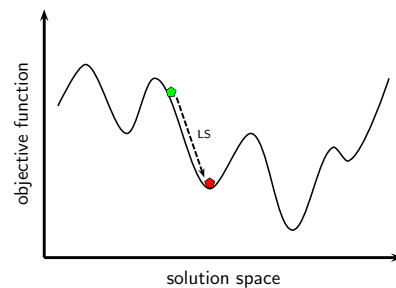
Iterate:

- ▶ given a s^* , perturb it: $s^* \rightsquigarrow s'$
- ▶ apply LocalSearch: $s' \rightsquigarrow s^{*'} /$
- ▶ apply acceptance test: $s^*, s^{*'} \rightsquigarrow s_{new}^*$

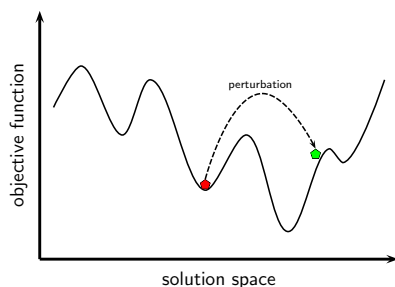
Advantage

Searching in \mathcal{S}^* leads from a large space \mathcal{S} to a smaller space \mathcal{S}^*

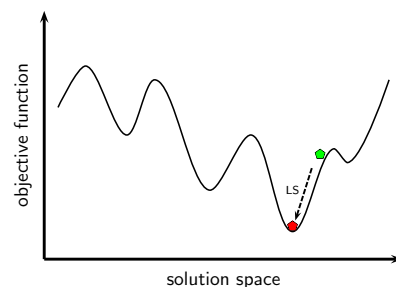
ILS – Pictorial view (step 1)



ILS – Pictorial view (step 2)



ILS – Pictorial view (step 3)



ILS – Procedural view

Pseudo code

```

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

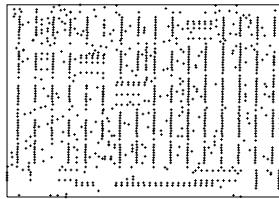
```

ILS – Important points

- ▶ performance depends on interaction among all modules
- ▶ **Basic version of ILS**
 - ▶ GenerateInitialSolution: random or construction heuristic
 - ▶ LocalSearch: often readily available
 - ▶ Perturbation: random move in higher order neighborhood
 - ▶ AcceptanceCriterion: force cost to decrease
- ▶ basic version often leads to very good performance
- ▶ basic version only requires few lines of additional code
- ▶ state-of-the-art results with further optimizations

ILS example: 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 4-opt move)
 - ▶ AcceptanceCriterion: accept $s^{*'} \text{ only if } f(s^{*'}) \leq f(s^*)$

ILS example: Quadratic assignment problem (QAP)

Problem specification

- ▶ **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

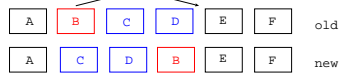
Basic ILS algorithm for the QAP

- ▶ GenerateInitialSolution: random initial solution
 - ▶ LocalSearch: 2-opt
-
- ▶ Perturbation: random k -opt move, $k > 2$
 - ▶ AcceptanceCriterion: accept $s^{*'} \text{ only if } f(s^{*'}) \leq f(s^*)$

ILS example: Permutation flow shop scheduling (PFSS)

Basic ILS algorithm for PFSP

- GenerateInitialSolution: NEH heuristic by Nawatz
- LocalSearch: insertion neighborhood



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



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

ILS — modules

ILS is a modular approach

Optimization of individual modules:

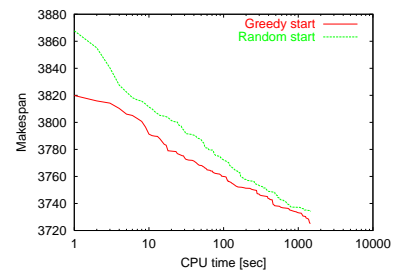
- complexity can be added step-by-step
- different implementation possibilities
- optimize single modules without considering interactions among modules
 ~> **local optimization of ILS**
- **global optimization of ILS** has to take into account interactions among components

Initial solution

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

Example: Initial solution for the PFSS

Influence of the initial solution



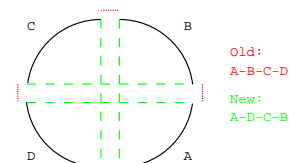
Perturbation

- important: **strength** of perturbation
 - **too strong**: close to random restart
 - **too weak**: LocalSearch may undo perturbation
- strength of perturbation may vary at run-time
- perturbation should be complementary to LocalSearch

Example: double-bridge move perturbation for TSP

Properties

- small perturbation
- complementary to Lin-Kernighan local search
- low cost increase



Perturbation strength

Attention

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.

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

Different perturbation schemes

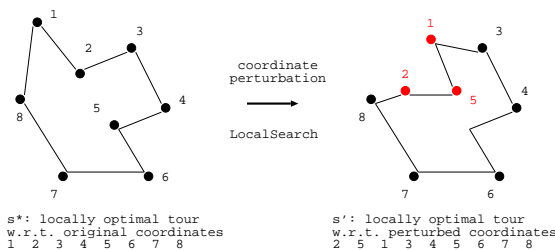
Adaptive perturbations

- ▶ single perturbation size not necessarily optimal
- ▶ perturbation size may vary at run-time
~> **basic Variable Neighborhood Search**
- ▶ perturbation size may be adapted at run-time
~> **reactive search**

Complex perturbation schemes

- ▶ optimizations of subproblems Lourenço, 1995
- ▶ input data modifications Baxter, 1981, Codenotti et al., 1996
 - ▶ modify data definition of instance
 - ▶ on modified instance run LocalSearch using input s^* , output is perturbed solution s'

Example: input data modification



Speed is an issue

- ▶ on many problems, small perturbations are sufficient
- ▶ LocalSearch in such a case will execute much faster
- ▶ sometimes access to LocalSearch in combination with Perturbation increases strongly speed (e.g. don't look bits)
- ▶ **Example:** TSP, number local searches in a given, same CPU-time

Speed is an issue: TSP example

- ▶ compare No. local searches (here, 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
- ▶ $\#LS_{1-DB}/\#LS_{RR}$: factor between $\#LS_{1-DB}$ and $\#LS_{RR}$
- ▶ $\#LS_{5-DB}$: No. local searches with five double bridge moves as Perturbation
- ▶ time limit: 120 sec on a Pentium II 266 MHz PC

instance	$\#LS_{RR}$	$\#LS_{1-DB}$	$\#LS_{5-DB}$
kroa100	17507	56186	34451
d198	7715	36849	16454
l1a318	4271	25540	9430
pcb442	4394	40509	12880
rat783	1340	21937	4631
pr1002	910	17894	3345
d1291	835	23842	4312
f11577	742	22438	3915
pr2392	216	15324	1777
pcb3038	121	13323	1232
f13795	134	14478	1773
r15915	34	8820	556

Acceptance criterion

- ▶ AcceptanceCriterion has strong influence on nature and effectiveness of walk in 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

Iterated local search
Variable neighborhood search (VNS)

Examples
ILS modules

Acceptance criterion: TSP example

Properties

- ▶ small perturbations are known to be enough
- ▶ high quality solutions are known to cluster
 ↳ good strategy incorporates intensification

Observations

- ▶ **Short runs:** Better is recommendable
- ▶ **Long runs:** More diversification needed

© Christian Blum
31

Iterated local search
Variable neighborhood search (VNS)

Examples
ILS modules

Search history

How to exploit the search history?

- ▶ Many of the bells and whistles of other strategies (diversification, intensification, tabu, adaptive perturbations and acceptance criteria, etc...) are applicable
- ▶ **Simple example** $\text{Restart}(s^*, s^{*'}, \text{history})$: Restart search if for a number of iterations no improved solution is found

© Christian Blum
32

Iterated local search
Variable neighborhood search (VNS)

Examples
ILS modules

Search history: QAP example results

instance	acceptance	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

© Christian Blum
33

Iterated local search
Variable neighborhood search (VNS)

Examples
ILS modules

Search history

Observations

- ▶ **Usually:** Complex interaction of perturbation and acceptance criterion
- ▶ **Tendency:** accepting several small perturbations better than accepting few large ones

© Christian Blum
34

Iterated local search
Variable neighborhood search (VNS)

Examples
ILS modules

Local search

General recommendations

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

© Christian Blum
35

Iterated local search
Variable neighborhood search (VNS)

Examples
ILS modules

Complex local search options

Instead of best/first-improvement local search, use

- ▶ Variable neighborhood search (VNS)
- ▶ Tabu search (TS)
- ▶ Simulated annealing (SA)

© Christian Blum
36

Effectiveness of local search

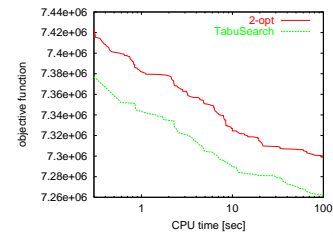
- **Often:** the more effective the local search the better performs ILS
- Example TSP:** 2-opt vs. 3-opt vs. Lin-Kernighan
- **Sometimes:** preferable to have fast but less effective local search

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

Effectiveness of local search: QAP example (1)

- short tabu search runs (6n iterations) vs. 2-opt, same CPU-time
- instance tai60a, random, unstructured instance

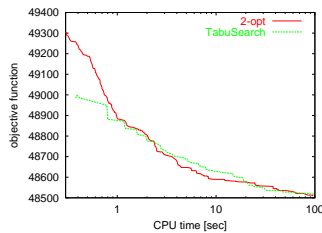
Performance comparison



Effectiveness of local search: QAP example (2)

- short tabu search runs (6n iterations) vs. 2-opt, same CPU-time
- instance sko64, grid distances, structured flows

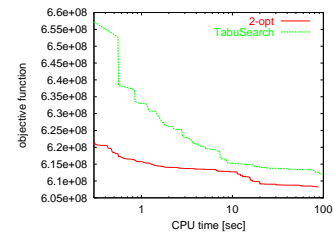
Performance comparison



Effectiveness of local search: QAP example (3)

- short tabu search runs (6n iterations) vs. 2-opt, same CPU-time
- instance tai60b, random, structured instances

Performance comparison



Optimization of ILS

- optimization of the interaction of ILS components
- optimization goal has to be given (optimize average solution quality, etc.)
- complex interactions among components exist
- global optimization of ILS is complex, therefore often heuristic approach
- global optimization is important to reach peak performance
- robustness is an important issue

Optimization of ILS

Guidelines

- GenerateInitialSolution should be to a large extent irrelevant for longer runs
- LocalSearch should be as effective and as fast as possible
- best choice of Perturbation may depend strongly on LocalSearch
- best choice of AcceptanceCriterion depends strongly on Perturbation and LocalSearch
- particularly important can be interactions among perturbation strength and AcceptanceCriterion

Optimization of ILS

Recommendations

- ▶ perturbation should not be easily undone by the 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 is a challenging problem

Recent References

Papers

- ▶ A. Subramanian, M. Battarra and C. N. Potts. **An Iterated Local Search heuristic for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times**, *International Journal of Production Research*, 52(9), pages 2729–2742, 2014
- ▶ D. Cattaruzza, N. Absi, D. Feillet and D. Vigo. **An iterated local search for the multi-commodity multi-trip vehicle routing problem with time windows.**, *Computers & Operations Research*, 51, pages 257–267, 2014
- ▶ M. Dell’Amico, J. C. D. Díaz, G. Hasle and M. Iori. **An adaptive iterated local search for the mixed capacitated general routing problem**, *Transportation Science*, 50(4), pages 1223–1238, 2016

bla

Variable neighborhood search

Historical note

- ▶ Main idea is very similar to ILS
- ▶ Algorithmic framework introduced by [Mladenović and Hansen, 1997]
- ▶ One of the first applications was to the p -median problem [Hansen and Mladenović, 1997]

Literature:

1. [Mladenović and Hansen, 1997] **Variable neighborhood search**, *Computers & Operations Research*, 24, pages 1097–1100, 1997
2. [Hansen and Mladenović, 1997] **Variable neighborhood search for the p -median**, 5(4), pages 207–226, 1997

What is variable neighborhood search?

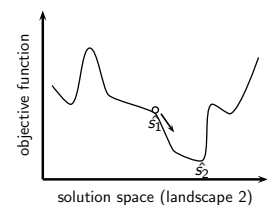
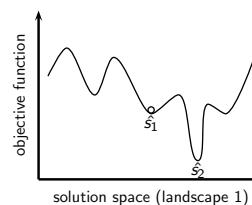
Variable neighborhood search (VNS)

is a metaheuristic method that is based on the systematic change of the neighborhood during the search.

Central observations

- ▶ A local minimum w.r.t. one neighborhood structure is not necessarily locally minimal w.r.t. another neighborhood structure
- ▶ A global optimum is locally optimal w.r.t. **all** neighborhood structures

Pictorial view



VNS Idea

- ▶ **principle**: change the neighborhood during the search
- ▶ **several variations** of this central principle:
 - ▶ variable neighborhood descent
 - ▶ basic variable neighborhood search
 - ▶ reduced variable neighborhood search
 - ▶ variable neighborhood decomposition search
- ▶ **Notation**
 - ▶ $\mathcal{N}_k, k = 1, \dots, k_{\max}$ is a set of neighborhood structures
 - ▶ $\mathcal{N}_k(s)$ is the set of solutions in the k th neighborhood of s

Examples for neighborhood structures

k -exchange neighborhoods

- ▶ **Sub-set problems**: k objects in, k other ones out
- ▶ **Permutation problems**: k -opt moves

Increasing distance measures

- ▶ **Bit-strings**: Neighborhood of all bitstrings with distance k
- ▶ ...

A first simple use of this idea

Variable Neighborhood Descent (VND)

```

Select a set of neighborhood structures  $\mathcal{N}_k, k = 1, \dots, k_{\max}$ 
 $s \leftarrow \text{GenerateInitialSolution}()$ 
 $k \leftarrow 1$ 
while  $k \leq k_{\max}$  do
     $s' \leftarrow \text{PickBestNeighbor}(\mathcal{N}_k(s))$ 
    if  $f(s') < f(s)$  then
         $s \leftarrow s'$ 
         $k \leftarrow 1$ 
    else
         $k \leftarrow k + 1$ 
    end if
end while
    
```

Comments on VND

- ▶ final solution is locally optimal with respect to all neighborhoods
- ▶ typically, order neighborhoods from smallest to largest
- ▶ if local search algorithms $\mathcal{L}_k, k = 1, \dots, k_{\max}$ are available as black-box procedures
 - ▶ order black-boxes
 - ▶ apply them in the given order
 - ▶ possibly iterate starting from the first one
 - ▶ advantage: **solution quality** and **speed**

Basic variable neighborhood search

```

Select a set of neighborhood structures  $\mathcal{N}_k, k = 1, \dots, k_{\max}$ 
 $s \leftarrow \text{GenerateInitialSolution}()$ 
while termination conditions not met do
     $k \leftarrow 1$ 
    while  $k \leq k_{\max}$  do
         $s' \leftarrow \text{PickAtRandom}(\mathcal{N}_k(s))$       {Shaking phase}
         $s'' \leftarrow \text{LocalSearch}(s')$ 
        if  $f(s'') < f(s)$  then
             $s \leftarrow s''$ 
             $k \leftarrow 1$ 
        else
             $k \leftarrow k + 1$ 
        end if
    end while
end while
    
```

Properties of basic variable neighborhood search

- ▶ (standard) local search is applied in \mathcal{N}_1
- ▶ other neighborhoods are explored only randomly
- ▶ explorations of other neighborhoods are perturbations in the ILS-sense
- ▶ perturbation is systematically varied
- ▶ AcceptanceCriterion: $\text{Better}(s^*, s'^*)$

Variants of VNS

- **Order of the neighborhoods**
 - **forward VNS**: start with $k = 1$ and increase k by one if no better solution is found; otherwise set $k \leftarrow 1$
 - **backward VNS**: start with $k = k_{\max}$ and decrease k by one if no better solution is found
 - **extended version**: parameters k_{\min} and k_{\max} ; set $k \leftarrow k_{\min}$ and increase k by $k_{\max} - k_{\min} + 1$ if no better solution is found

- **Acceptance of worse solutions**

- accept worse solutions with some probability
- **Skewed VNS**: accept if

$$f(s'') - \alpha d(s^*, s'') < f(s^*)$$

$d(s^*, s'')$ measures the distance between solutions

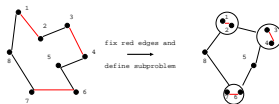
Reduced VNS

- ... is the same as basic VNS except that no LocalSearch procedure is applied
- ... only explores randomly different neighborhoods
- ... can be faster than standard local search algorithms for reaching good quality solutions

Variable neighborhood decomposition search (VNDS)

Central idea

- generate subproblems by keeping all but k solution components fixed
- apply a local search only to the k “free” components



VNDS — pseudo code

```

Select a set of neighborhood structures  $\mathcal{N}_k, k = 1, \dots, k_{\max}$ 
 $s \leftarrow \text{GenerateInitialSolution}()$ 
while termination conditions not met do
   $k \leftarrow 1$ 
  while  $k \leq k_{\max}$  do
     $s' \leftarrow \text{PickAtRandom}(\mathcal{N}_k(s))$     { $s$  and  $s'$  differ in  $k$ 
    attributes}
     $s'' \leftarrow \text{LocalSearch}(s')$     {only moves involving the  $k$ 
    attributes}
    if  $f(s'') < f(s)$  then
       $s \leftarrow s''$ 
       $k \leftarrow 1$ 
    else
       $k \leftarrow k + 1$ 
  end if
end while

```

Relationship between ILS and VNS

- the two metaheuristic methods are based on different underlying “philosophies”
- they are similar in many respects
- ILS appears to be more flexible with respect to the optimization of the interaction of modules
- generally they are both robust metaheuristics
- in general highly efficient

Questions?

Questions?