Iterated Local Search
—
Variable Neighborhood Search

Thomas Stützle

stuetzle@informatik.tu-darmstadt.de

http://www.intellektik.informatik.tu-darmstadt.de/˜tom.

Darmstadt University of Technology

Department of Computer Science

Intellectics Group

Outline

- Iterated Local Search
  - Framework
  - Implementation
  - Practice
- Variable Neighborhood Search
  - Framework
  - Variants
- Concluding Remarks

What is Iterated Local Search?

Iterated local search (ILS) is an SLS method that generates a sequence of solutions generated by an embedded heuristic, leading to far better results than if one were to use repeated random trials of that heuristic.

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

Iterated Local Search — Framework

Given

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

Question

- can such an algorithm be improved by iteration?

YES

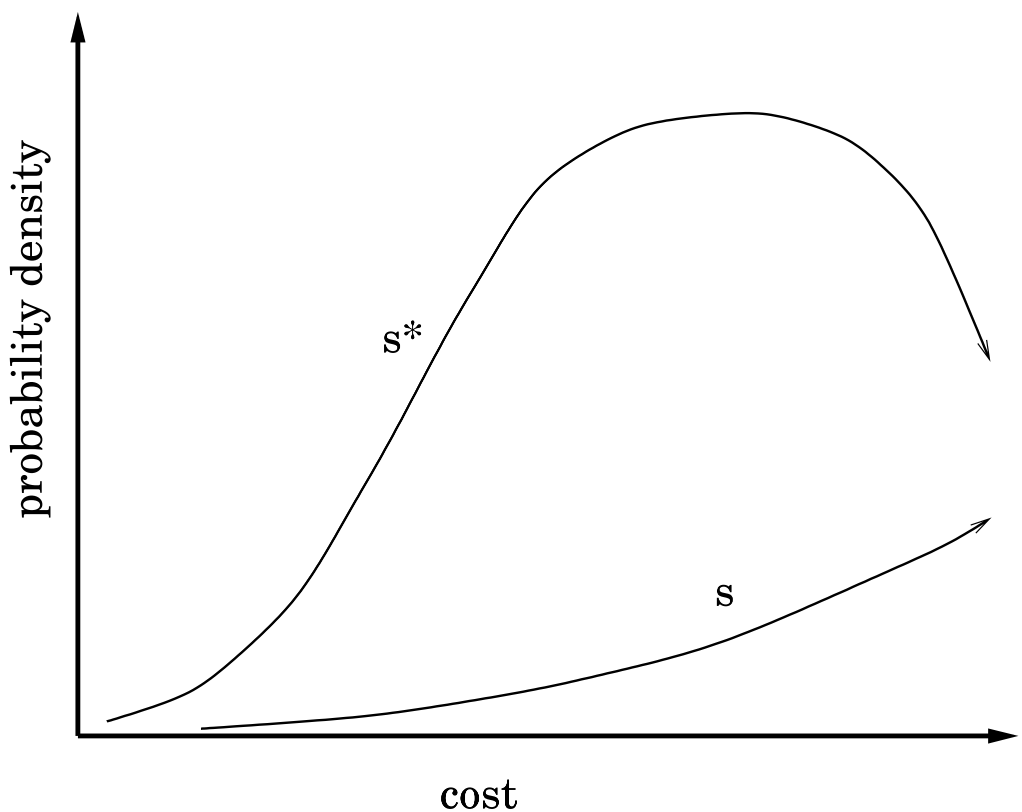
ILS — Notation

- $\mathcal{S}$: 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 $\mathcal{S} \mapsto \mathcal{S}^*$

Cost distributions

Cost distributions

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



How to go beyond **LocalSearch?**

Random Restart

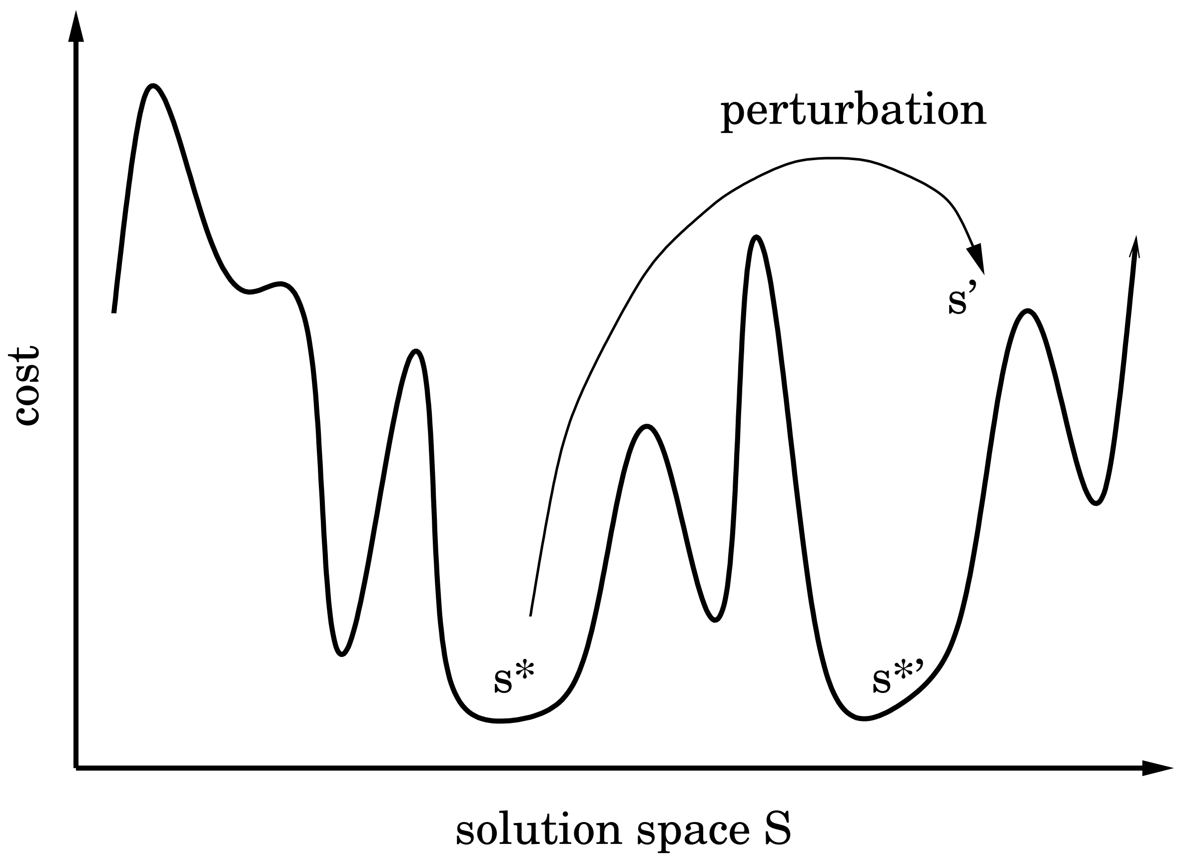
- generate multiple s^* independently
- theoretical guarantees
- practically not very effective
- for large instances leads to costs with
  - fixed percentage excess above optimum
  - distribution becomes arbitrarily peaked around the mean in the instance size limit

ILS – Principle

Searching in $\mathcal{S}^*$

- LocalSearch leads from a large space $\mathcal{S}$ to a smaller space $\mathcal{S}^*$
- define a biased walk in $\mathcal{S}^*$
  - given a s^*, perturb it: $s^* \rightsquigarrow s'$
  - apply LocalSearch: $s' \rightsquigarrow s^{*\prime}$
  - apply acceptance test: $s^*, s^{*\prime} \rightsquigarrow s^*_{new}$

ILS – Pictorial view



ILS – Procedural view

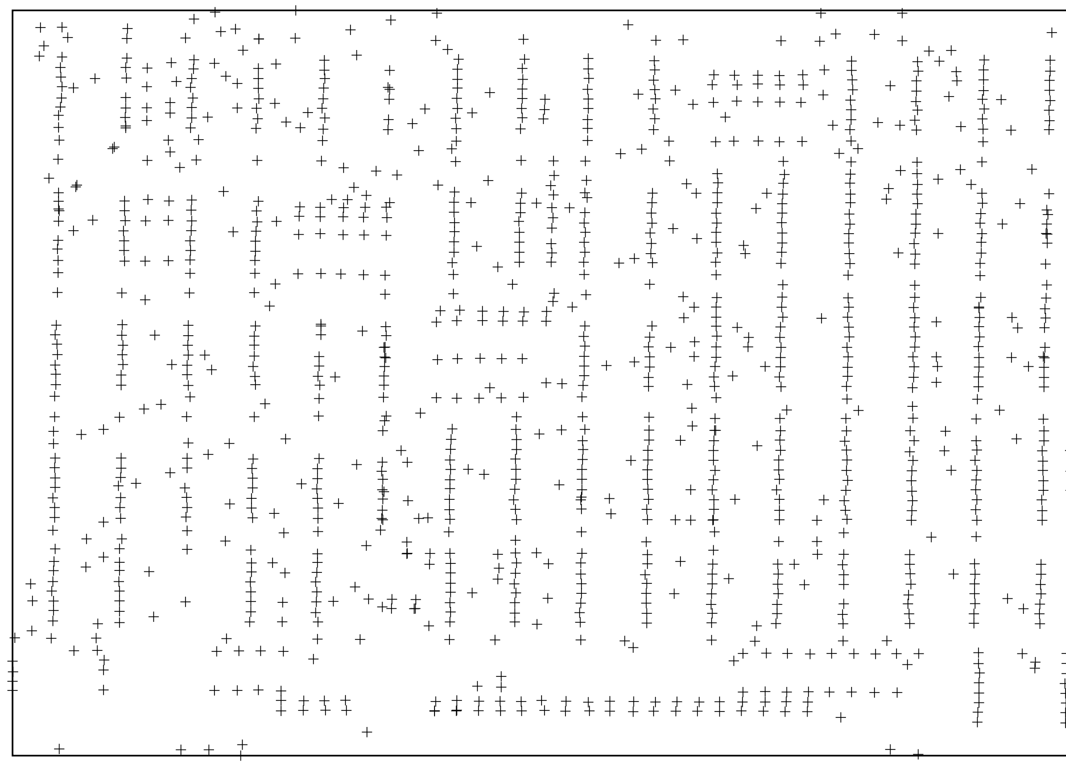
procedure *Iterated Local Search*
$s_0 \leftarrow$ GenerateInitialSolution
$s^* \leftarrow$ LocalSearch(s_0)
repeat
$s' \ \leftarrow$ Perturbation$(s^*, \textit{history})$
$s^{*\prime} \leftarrow$ LocalSearch(s')
$s^* \ \leftarrow$ AcceptanceCriterion$(s^*, s^{*\prime}, \textit{history})$
until termination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

Iterated Local Search — Algorithm

- 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 Examples — TSP

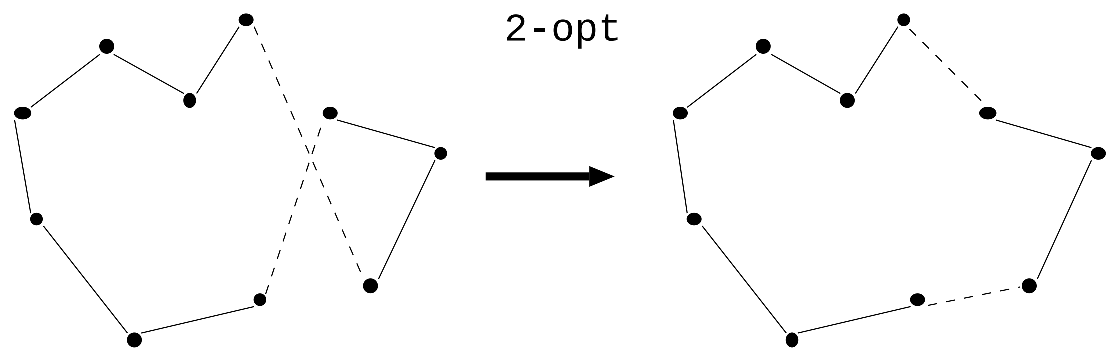
- given: fully connected, weighted Graph $G = (V, E, d)$
- goal: find shortest Hamiltonian cycle
- hardness: $\mathcal{NP}$-hard
- interest: standard benchmark problem for algorithmic ideas



ILS Examples — TSP

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 $s^{*\prime}$ only if $f(s^{*\prime}) \leq f(s^{*})$

ILS Examples — QAP

- given: n objects and n locations with
  - a_{ij} : flow from object i to object j
  - $d_r{}^s$: 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

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

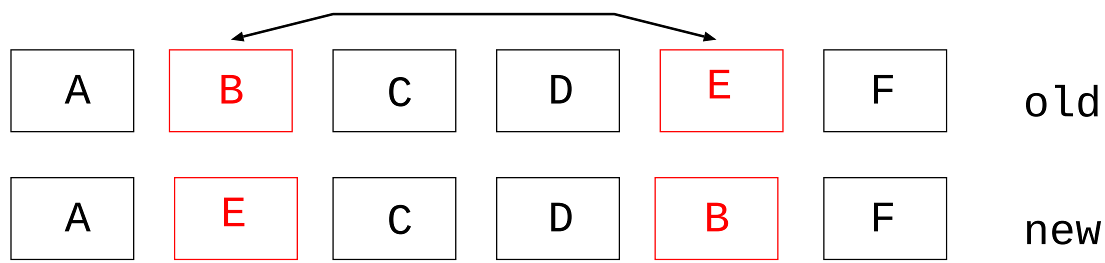
  $\pi(i)$ gives location of object i

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

ILS Examples — QAP

basic ILS algorithm for QAP

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



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

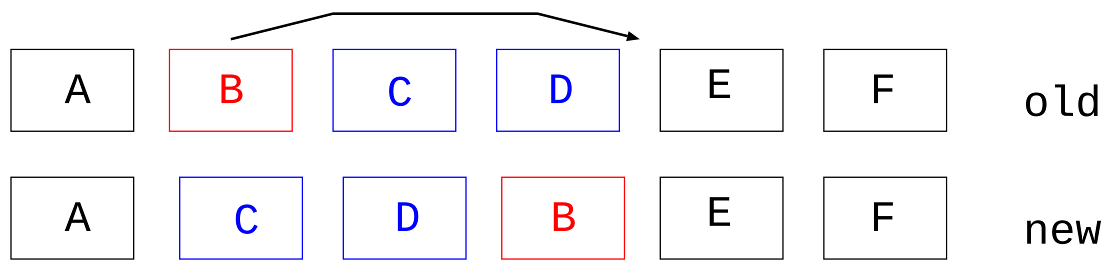
ILS Examples — Permutation FSP

- 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
- goal: minimize the completion time $C_{\max}$ of last job (makespan).
- interest: prototypical scheduling problem, $\mathcal{NP}$-hard

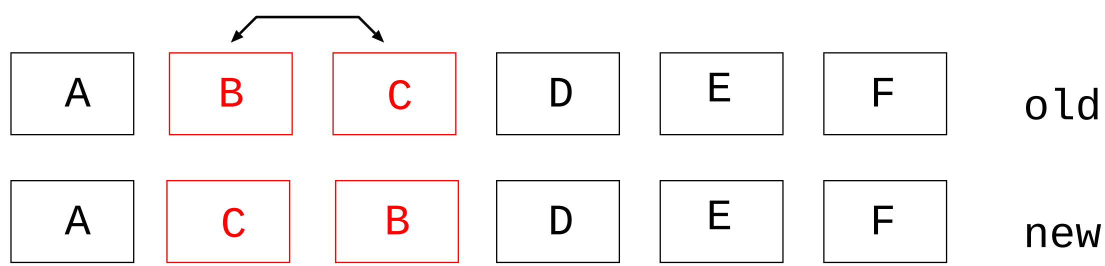
ILS Examples — FSP

basic ILS algorithm for FSP

- GenerateInitialSolution: NEH heuristic
- LocalSearch: insertion neighborhood



- Perturbation: a number of swap- or interchange moves

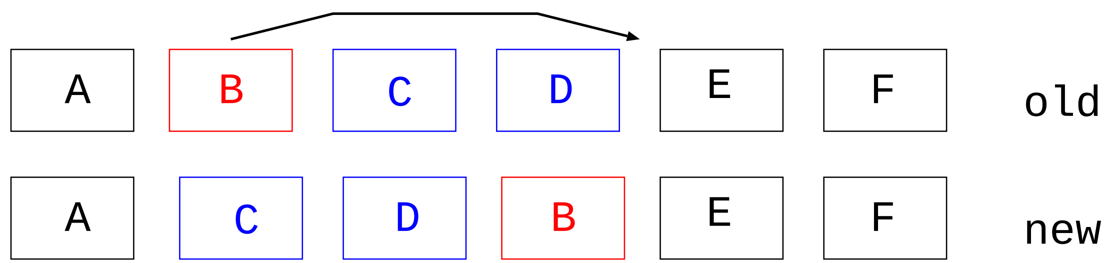


- AcceptanceCriterion: accept $s^{*\prime}$ only if $f(s^{*\prime}) \leq f(s^*)$

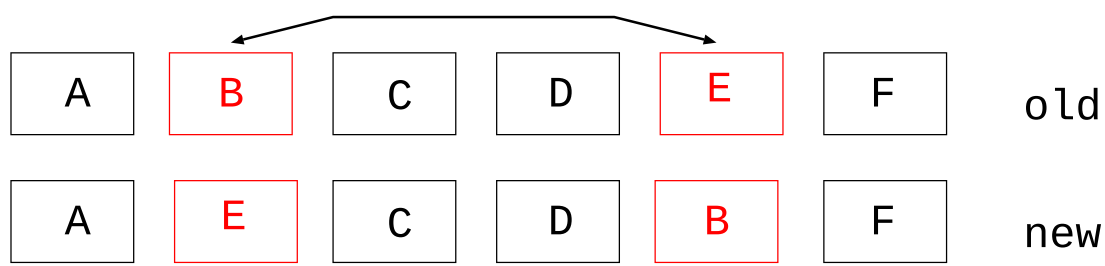
ILS Examples — FSP

basic ILS algorithm for FSP

- GenerateInitialSolution: NEH heuristic
- LocalSearch: insertion neighborhood



- Perturbation: a number of swap- or interchange moves



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

ILS — modules

ILS is a modular approach

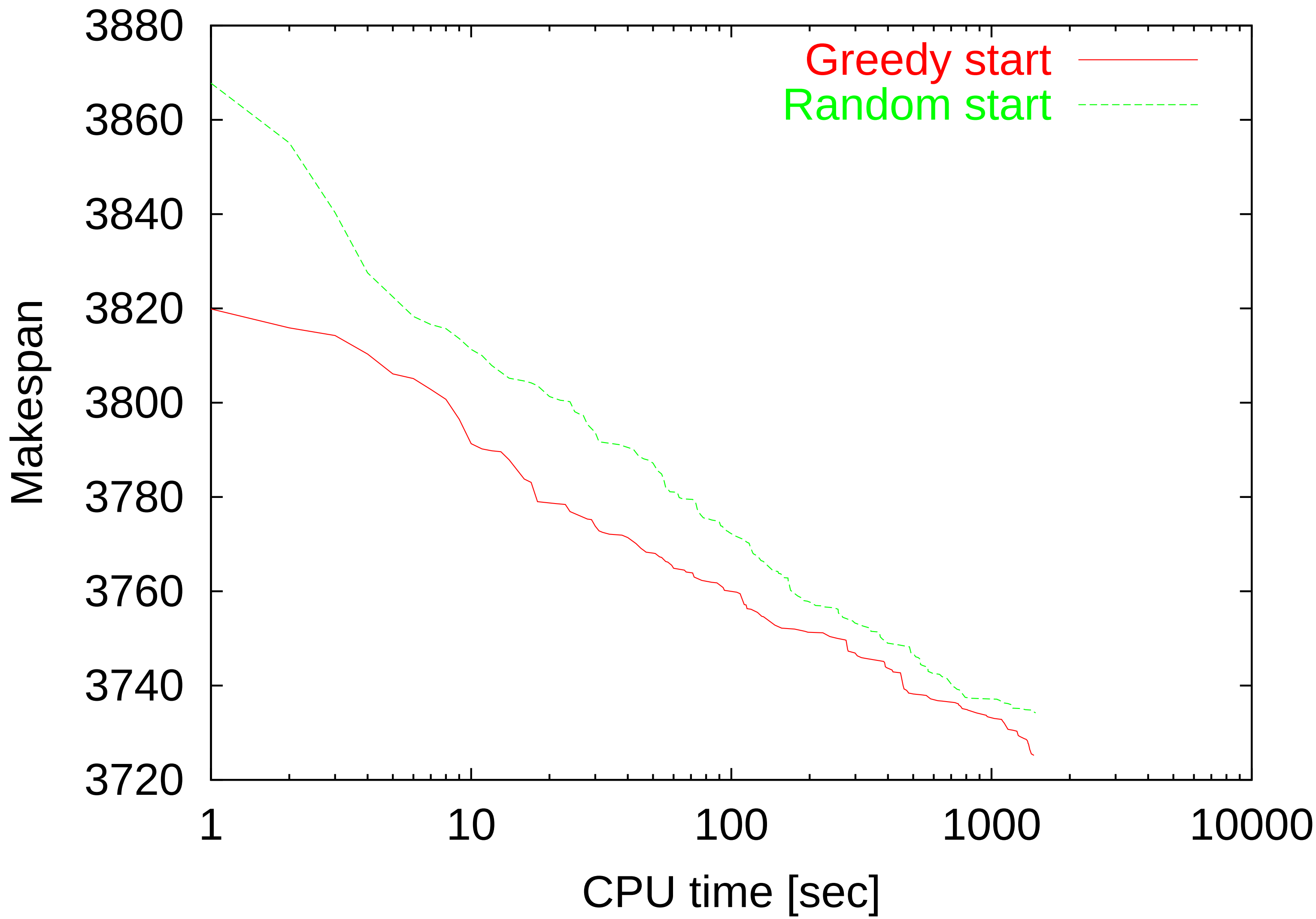
Optimization of individual modules

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

ILS — Initial solution

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

ILS for FSP, initial solution

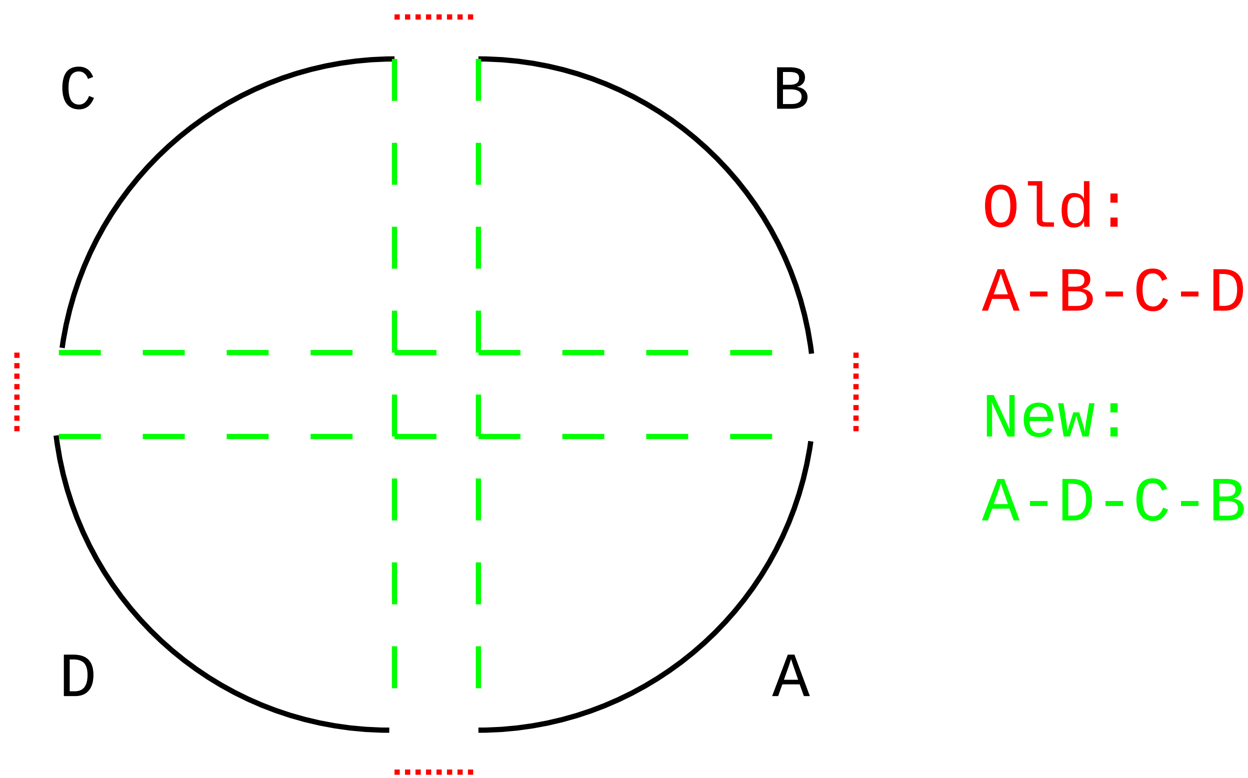


ILS — Perturbation

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

Example: double-bridge move for TSP

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



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.

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

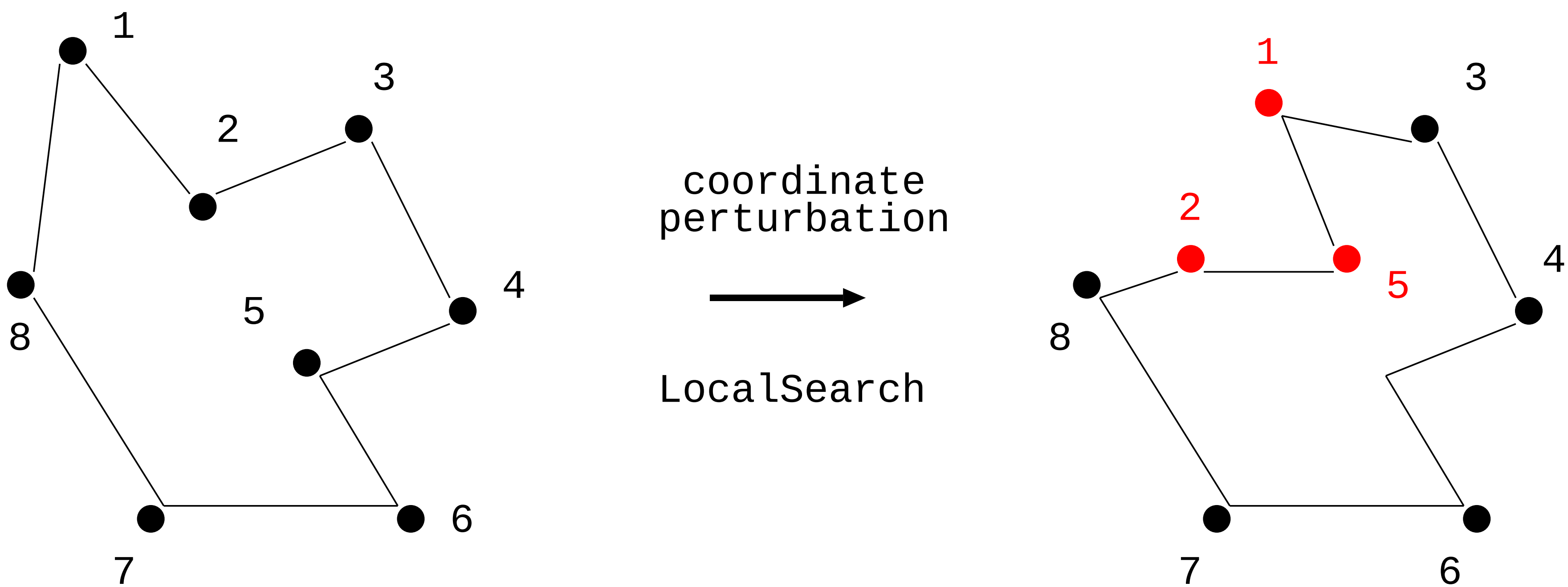
Adaptive perturbations

- single perturbation size not necessarily optimal
- perturbation size may vary at run-time
  $\rightsquigarrow$ basic Variable Neighborhood Search
- perturbation size may be adapted at run-time
  $\rightsquigarrow$ 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'

Input data modifications



s*: locally optimal tour
w.r.t. original coordinates
1 2 3 4 5 6 7 8

s': locally optimal tour
w.r.t. perturbed coordinate
2 5 1 3 4 5 6 7 8

Perturbation — Speed

- 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

Perturbation — Speed, ILS for TSP

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

instance	$\#LS_{RR}$	$\#LS_{1\text{-DB}}$	$\#LS_{1\text{-DB}}/\#LS_{RR}$
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
fl1577	742	22438	30.24
pr2392	216	15324	70.94
pcb3038	121	13323	110.1
fl3795	134	14478	108.0
rl5915	34	8820	259.4

Perturbation — Speed, ILS for TSP

- compare No. local searches (here, 3-opt) in fixed computation time
- $\#LS_{RR}$: No. local searches with random restart
- $\#LS_{1\text{-}DB}$: No. local searches with one double bridge move as Perturbation
- $\#LS_{5\text{-}DB}$: No. local searches with five double bridge moves as Perturbation
- time limit: 120 sec on a Pentium II 266 MHz PC

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

ILS — Acceptance Criterion

- 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`$(s^*, s^{*\prime}, \mathit{history})$: accept $s^{*\prime}$ only if $f(s^{*\prime}) < f(s^*)$
- extreme diversification:
  `RW`$(s^*, s^{*\prime}, \mathit{history})$: accept $s^{*\prime}$ always
- many intermediate choices possible

ILS — Acceptance Criterion

Example: TSP

- small perturbations are known to be enough
- high quality solutions are known to cluster
  $\rightsquigarrow$ good strategy incorporates intensification

ILS — Example results TSP

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

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
`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

ILS — Acceptance Criterion

Example: TSP

- small perturbations are known to be enough
- high quality solutions are known to cluster
$\rightsquigarrow$ good strategy incorporates intensification

Observations

- best results for short runs with `Better`
- for long runs, effective diversification strategies result in much improved performance over

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

ILS — 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

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

Observations

- complex interaction of perturbation and acceptance criterion
- tendency: accepting several small perturbations better than accepting few large ones

ILS — Local search

- 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)

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

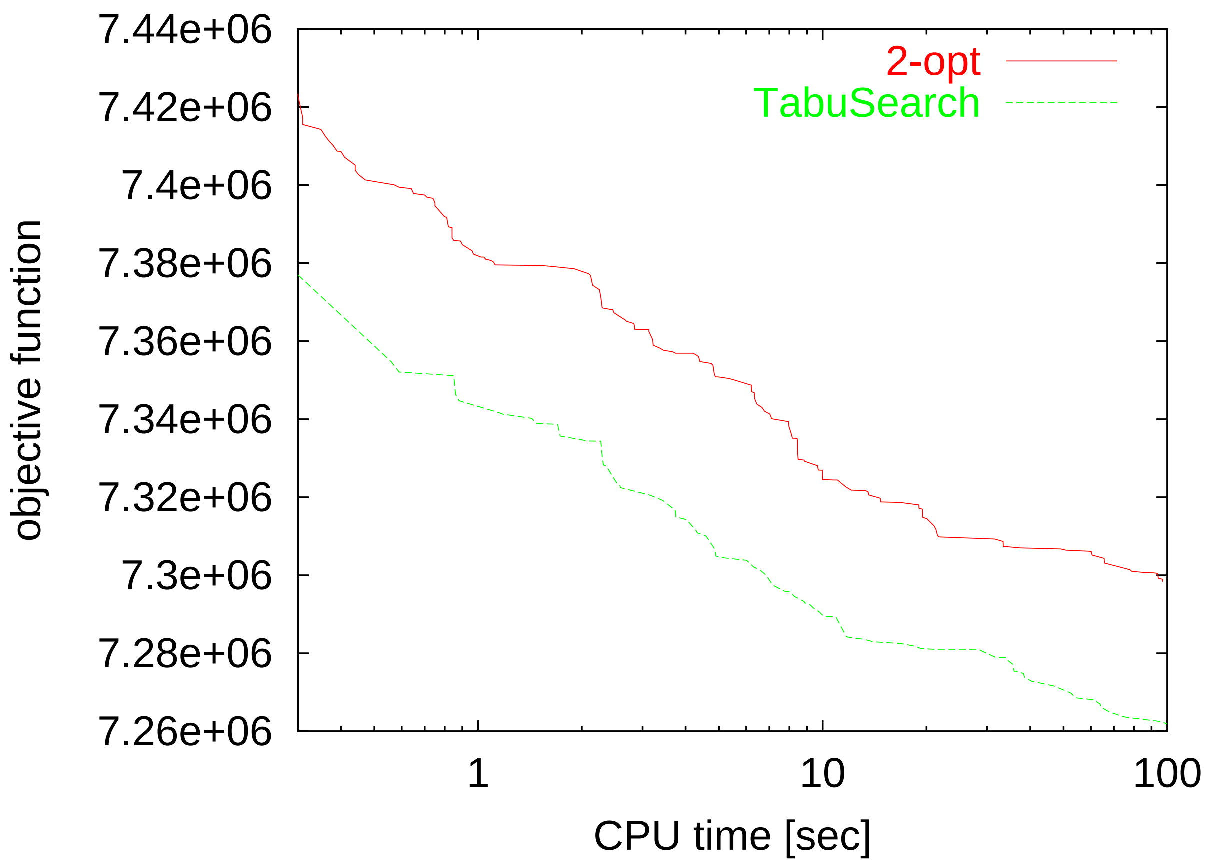
Effectiveness of local search?

- often: the more efective 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 adressed when optimizing an ILS algorithm

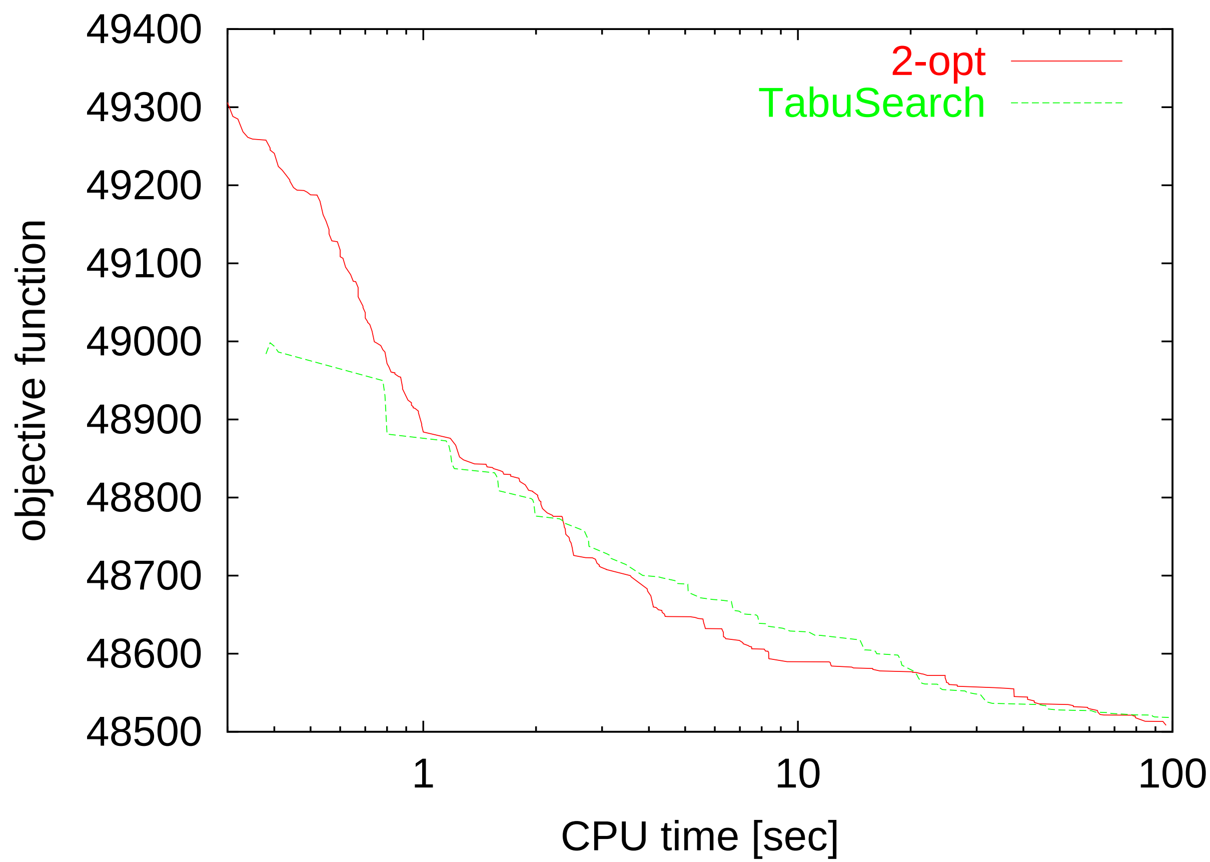
ILS, QAP — tabu search vs. 2-opt

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



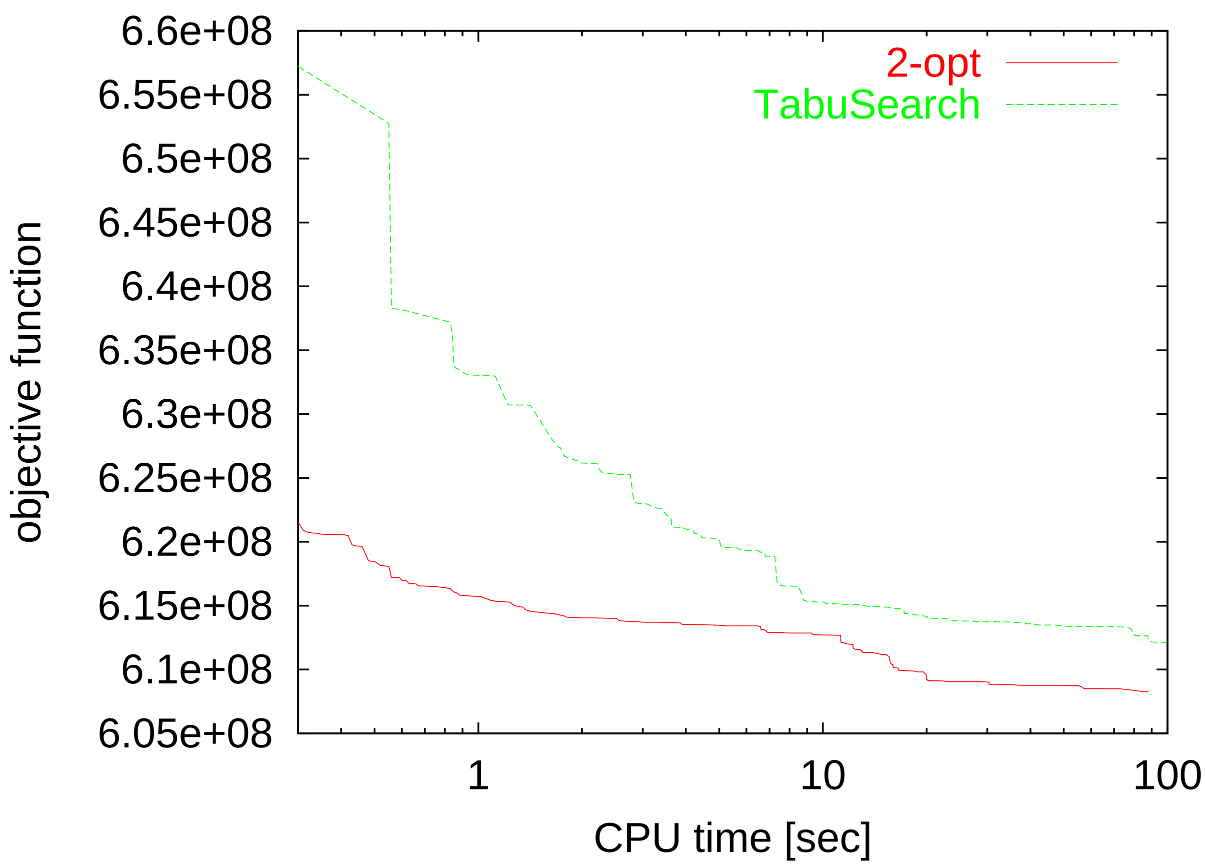
ILS, QAP — tabu search vs. 2-opt

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



ILS, QAP — tabu search vs. 2-opt

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



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

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

- Ad-hoc optimization
  - optimize single components, e.g. in the order GenerateInitialSolution, LocalSearch, Perturbation, AcceptanceCriterion
  - iterate through this process
- Experimental design techniques
  - factorial experimental design
  - racing algorithms
  - response surface methodolgy
  - run-time distributions methodology

Optimization of ILS

Main dependencies

- 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

Iterated Local Search — Applications

- first approaches by Baxter, 1981 to a location problem and Baum, 1986 to TSP
- most developed applications are those to TSP
- several applications to scheduling problems
- for several problems state-of-the-art results

ILS for TSPs

- iterated descent *Baum, 1986*
  - first approach, relatively poor results
- large step Markov chains *Martin, Otto, Felten, 1991, 1992, 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

ILS for TSPs

- improved LSMC *Hong, Kahng, Moon, 1997*
  - study of different perturbation sizes, acceptance criteria
- 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–today
  - diversification mechansim in ILS for long run times
- ILS with genetic transformation *Katayama, Narisha, 1999*
  - perturbation guided by a second solution

ILS for scheduling problems

- single machine total weighted tardiness problem
  - iterated dynasearch *Congram, Potts, Van de Velde, 1998*
  - ILS with VND local search *den Besten, Stützle, Dorigo, 2000*
- single and parallel machine scheduling
  - several problems attacked by *Brucker, Hurink, Werner 1996, 1997*
- flow shop scheduling
  - permutation flow shop problem *Stützle, 1998*
  - flow shop problem with stages in series *Yang, Kreipl, Pinedo, 2000*

ILS for scheduling problems

- job shop scheduling (JSP)
  - ILS approach to JSP with makespan criterion

    Lourenço 1995, Lourenço, Zwijnenburg, 1996
  - guided local search extensions to JSP with makespan criterion *Balas, Vazacopoulos 1998*
  - total weighted tardiness job shop problem *Kreipl, 2000*

Other applications

- graph partitioning *Martin, Otto, 1995*
  - problem specific perturbation
- unweighted MAX-SAT *Battiti, Protasi, 1997*
  - a reactive search algorithm which fits into ILS framework; tabu search was used in perturbation phase; good performance also due to good tie-breaking criterion
- weighted MAX-SAT *Smith, Hoos, Stützle, 2002*
  - tabu-type search in perturbation; currently a state-of-the-art algorithm for MAX-SAT; a preliminary version won the "competition" in the Metaheuristics Network
- graph colouring *Chiarandini, Paquete, Stützle, 2001, 2002*
  - very good performance over a wide range of instances
- . . .

Reference

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 via

`http://www.intellektik.informatik.tu-darmstadt.de/~tom/pub.html`

Variable Neighborhood Search (VNS)

Variable Neighborhood Search is an SLS 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

Variable Neighborhood Search (VNS)

- principle: change the neighborhood during the search
- several adaptations of this central principle
  - variable neighborhood descent
  - basic variable neighborhood search
  - reduced variable neighborhood search
  - variable neighborhood decomposition search
- notation
  - $\mathcal{N}_k, k = 1, \ldots, k_{max}$ is a set of neighborhood structures
  - $\mathcal{N}_k(s)$ is the set of solutions in the kth neighborhood of s

Variable Neighborhood Search (VNS)

How to generate the various neighborhood structures?

- for many problems different neighborhood structures (local searches) exist / are in use
- change parameters of existing local search algorithms
- use k-exchange neighborhoods; these can be naturally extended
- many neighborhood structures are associated with distance measures; in this case increase the distance

Variable Neighborhood Descent (VND)

- change the neighborhood in a deterministic way

procedure *VND*
    $s_0 \leftarrow$ GenerateInitialSolution, choose $\{\mathcal{N}_k\}, k = 1, \ldots, k_{max}$
    $k \leftarrow 1$
    repeat
        $s' \leftarrow$ FindBestNeighbor(s)
        if $f(s') < f(s)$ **then**
            $s \leftarrow s'$
            $k \leftarrow 1$
        else
            $k \leftarrow k + 1$
    until $k > k_{max}$
end

VND

- final solution is locally optimal w.r.t. all neighborhoods
- first improvement may be applied instead of best improvement
- typically, order neighborhoods from smallest to largest
- if local search algorithms $\mathcal{L}_k, k = 1, \ldots, k_{max}$ 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

VND — example results

- VND for single-machine total weighted tardiness problem
  - candidate solutions are permutations of job indices
- examined were insert and interchange neighborhoods
- influence of different starting heuristics examined

initial	interchange		insert		inter+insert		insert+inter	
solution	Δ_{avg}	t_{avg}	Δ_{avg}	t_{avg}	Δ_{avg}	t_{avg}	Δ_{avg}	t_{avg}
EDD	0.62	0.140	1.19	0.64	0.24	0.20	0.47	0.67
MDD	0.65	0.078	1.31	0.77	0.40	0.14	0.44	0.79
AU	0.92	0.040	0.56	0.26	0.59	0.10	0.21	0.27

Δ_{avg}: deviation from best-known solutions, averaged over 100 instances

t_{avg}: average computation time on a Pentium II 266MHz

Basic VNS

- uses neighborhood structures $\mathcal{N}_k, k = 1, \ldots, k_{max}$
- (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: `Better`$(s^*, s^{*\prime})$

Basic VNS — Procedural view

procedure *basic VNS*

$s_0 \leftarrow$ GenerateInitialSolution, choose $\{\mathcal{N}_k\}, k = 1, \ldots, k_{max}$

repeat

$s' \leftarrow$ RandomSolution$(\mathcal{N}_k(s^*))$

$s^{*\prime} \leftarrow$ LocalSearch(s') % local search w.r.t. $\mathcal{N}_1$

if $f(s^{*\prime}) < f(s^*)$ **then**

$s^* \leftarrow s^{*\prime}$

$k \leftarrow 1$

else

$k \leftarrow k + 1$

until termination condition

end

Basic VNS — variants

- 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_{step}; set $k \leftarrow k_{min}$ and increase k by k_{step} if no better solution is found
- acceptance of worse solutions
  - accept worse solutions with some probability
  - Skewed VNS: accept if

$$f(s^{*\prime}) - \alpha d(s^*, s^{*\prime}) < f(s^*)$$

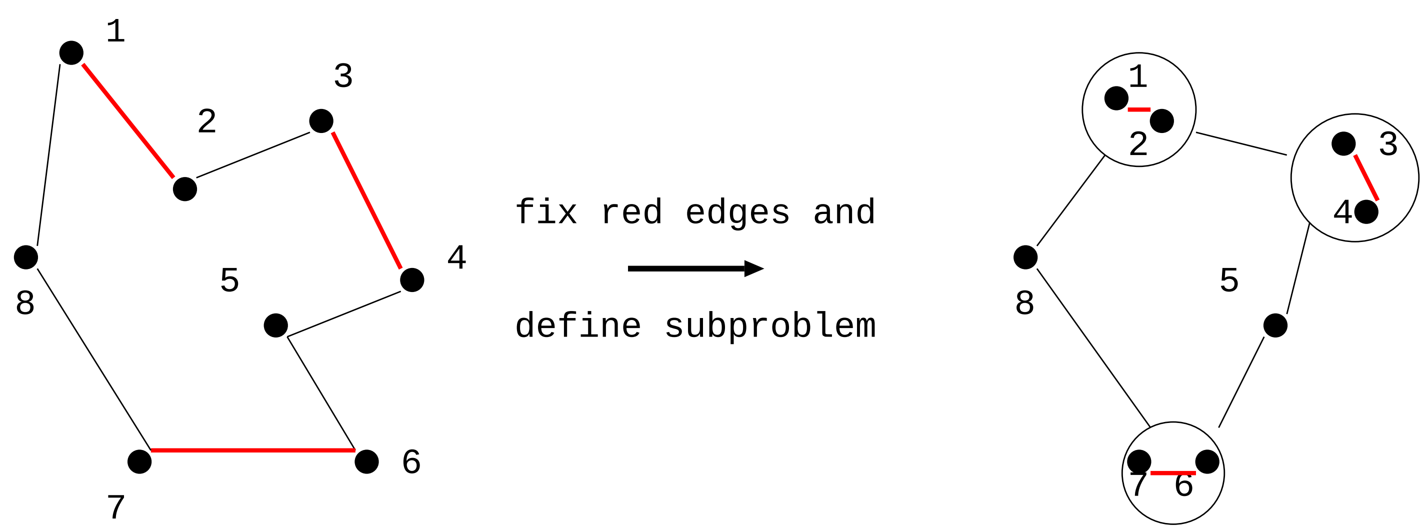
$d(s^*, s^{*\prime})$ measures the distance between solutions

Reduced VNS

- 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

Var. Neighborhood Decomposition Search

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



- related approaches: POPMUSIC, MIMAUSA, etc.

VNDS — Procedural view

procedure *VNDS*
  $s_0 \leftarrow$ GenerateInitialSolution, choose $\{\mathcal{N}_k\}, k = 1, \ldots, k_{max}$
  repeat
    $s' \leftarrow$ RandomSolution$(\mathcal{N}_k(s))$
    $t \leftarrow$ FreeComponents(s', s)
    $t^* \leftarrow$ LocalSearch(t) % local search w.r.t. $\mathcal{N}_1$
    $s'' \leftarrow$ InjectComponents(t^*, s')
    if $f(s'') < f(s)$ **then**
      $s \leftarrow s''$
      $k \leftarrow 1$
    else
      $k \leftarrow k + 1$
  until termination condition
end

Relationship between ILS and VNS

- the two SLS methods are based on different underlying "philosophies"
- they are similar in many respects
- ILS apears to be more flexible w.r.t. optimization of the interaction of modules
- VNS gives place to approaches like VND for obtaining more powerful local search approaches

ILS and VNS — Conclusions

ILS and VNS are ..

- based on simple principles
- easy to understand
- basic versions are easy to implement
- robust
- highly effective

Future work

- applications to new types of problems

  ⇝ multi-objective, dynamic, stochastic, logic, etc.

- reasons for the success of ILS/VNS

  ⇝ search space analysis

- understanding where they fail

  ⇝ search space analysis

- understanding of the interaction between the modules GenerateInitialSolution, LocalSearch, Perturbation, and AcceptanceCriterion

  ⇝ experimental design techniques

- systematic configuration of ILS/VNS algorithms

  ⇝ experimental design techniques, machine learning approaches