STOCHASTIC LOCAL SEARCH
FOUNDATIONS AND APPLICATIONS

Holger H. Hoos

Computer Science

University of BC

Canada

Thomas Stützle

FB Informatik

TU Darmstadt

Germany

Motivation: Why Stochastic Local Search?

Stochastic local search is the method of choice for solving many hard combinatorial problems.

Recent Progress & Successes:

- Ability of solving hard combinatorial problems has increased significantly

  – Solution of large propositional satisfiability problems

  – Solution of large travelling salesman problems

- Good results in new application areas

Reasons & Driving Forces:

- New algorithmic ideas

    – Nature inspired algorithms

    – New randomisation schemes

    – Hybrid and mixed search strategies

- Increased flexibility and robustness

- Improved understanding of algorithmic behaviour

- Sophisticated data structures

- Significantly improved hardware

Goals of the Tutorial

Provide answers to these questions:

- What is stochastic local search and how can it be used to solve computationally hard problems?

- Which stochastic local search techniques are available and what are their features?

- How should stochastic local search algorithms be studied and analysed empirically?

- How are specific problems solved using stochastic search?

Outline

- Introduction

- Part I: Combinatorial Problems and Search

  [Short Break]

- Part II: Stochastic Local Search Methods

  [Short Break]

- Part III: Stochastic Search Behaviour

  [Short Break]

- Part IV: Applications

- Conclusions and Issues for Future Research

Part I

Combinatorial Problems and Search

Combinatorial Problems

Examples for combinatorial problems:

- finding shortest/cheapest round trips (TSP)

- finding models of propositional formulae (SAT)

- planning, scheduling, time-tabling

- resource allocation

- protein structure prediction

- genome sequence assembly

Combinatorial problems ...

- involve finding a grouping, ordering, or assignment of a discrete set of objects which satisfies certain constraints

- arise in many domains of computer science and various application areas

- have high computational complexity ($\mathcal{NP}$-hard)

- are solved in practice by searching an exponentially large space of candidate / partial solutions

Combinatorial Decision Problems:

For a given problem instance, decide whether a solution (grouping, ordering, or assignment) exists which satisfies the given constraints.

The Propositional Satisfiability Problem (SAT)

Simple SAT instance (in CNF):

$$(a \vee b) \wedge (\neg a \vee \neg b)$$

$\rightsquigarrow$ **satisfiable, two models:**

$a = \text{true}, b = \text{false}$
$a = \text{false}, b = \text{true}$

SAT Problem – decision variant:

For a given propositional formula Φ,

decide whether Φ has at least one model.

SAT Problem – search variant:

For a given propositional formula Φ, if Φ is satisfiable,

find a model, otherwise declare Φ unsatisfiable.

SAT ...

- is a pervasive problem in computer science
  (Theory, AI, Hardware, ...)

- is computationally hard ($\mathcal{NP}$-hard)

- can encode many other combinatorial problems
  ($\mathcal{NP}$-completeness)

- is one of the conceptually simplest combinatorial
  decision problems
  $\rightsquigarrow$ facilitates development and evaluation of algorithmic ideas

Combinatorial Optimisation Problems:

Objective function assigns a numerical value to each candidate solution.

For a given problem instance, find a solution (grouping, ordering, or assignment) with maximal (or minimal) value of the objective function.

The Traveling Salesperson Problem (TSP)

TSP – optimisation variant:

For a given weighted graph $G = (V, E, w)$, find a Hamiltonian cycle in G with minimal weight,
i.e., find the shortest round-trip visiting each vertex exactly once.

TSP – decision variant:

For a given weighted graph $G = (V, E, w)$, decide whether a Hamiltonian cycle with minimal weight $\leq b$ exists in G.

TSP instance: shortest round trip through 532 US cities

TSPLIB instance att532 -- optimal solution
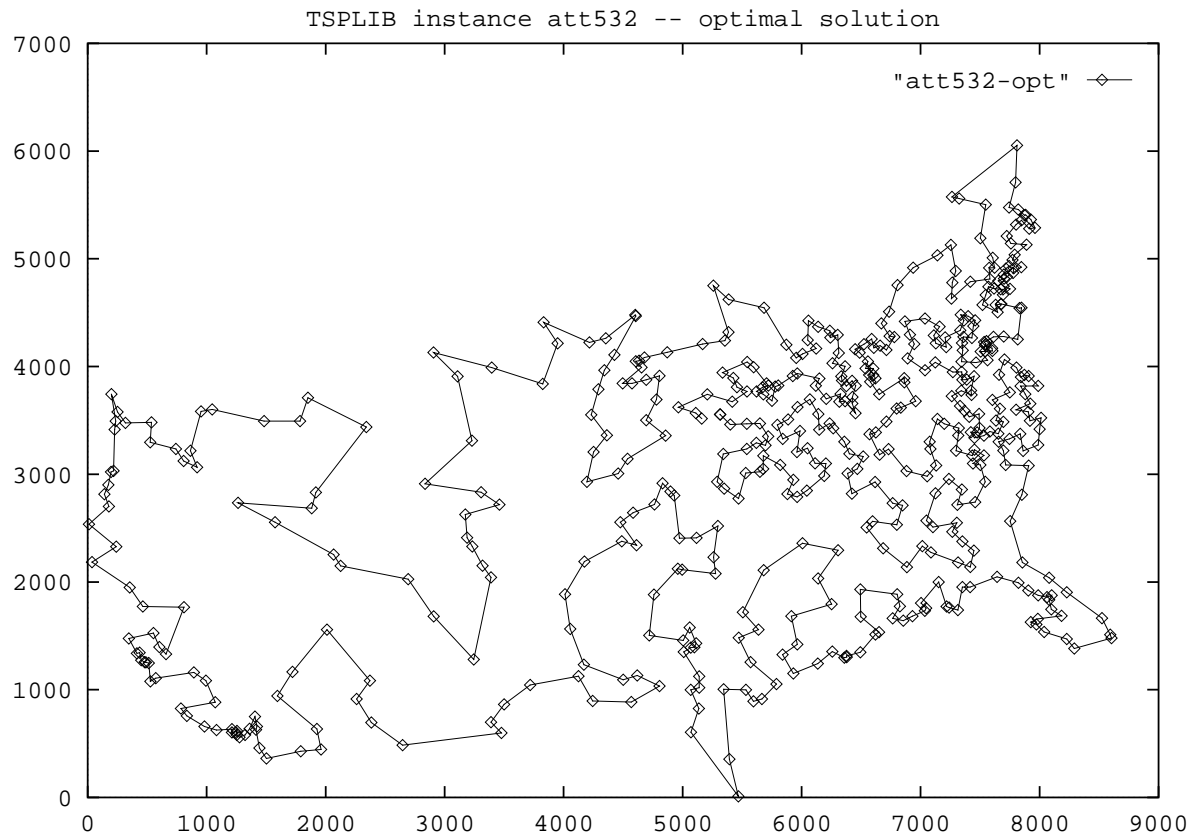
TSP ...

- is one of the most prominent and widely studied combinatorial optimisation problems in computer science and operations research

- is computationally hard ($\mathcal{NP}$-hard)

- is one of the conceptually simplest combinatorial optimisation problems
  $\rightsquigarrow$ facilitates development and evaluation of algorithmic ideas

Generalisations of combinatorial problems

- many combinatorial decision problems naturally generalise to optimisation problems, e.g. SAT to MAX-SAT

- many combinatorial problems have practically relevant dynamic variants (dynamic SAT, dynamic TSP, internet routing, dynamic scheduling)

- often, algorithms for decision problems can be generalised to optimisation and / or dynamic variants

- typically, good solutions to generalised problems require additional heuristics.

Complexity Issues

Motivation:

Analyse inherent hardness of problems and inherent complexity of algorithms.

Some basic concepts and results:

- complexity classes: $\mathcal{P}$ vs. $\mathcal{NP}$

- $\mathcal{NP}$-completeness, $\mathcal{NP}$-hardness

- $\mathcal{P} \neq \mathcal{NP}$ not known

- *but:* if there were an efficient (polynomial) algorithm for any $\mathcal{NP}$-complete problem, all $\mathcal{NP}$ problems could be solved efficiently

Complexity of combinatorial decision problems

- many combinatorial decision problems are $\mathcal{NP}$-hard

- this is reflected in search spaces of size exponential in problem size

- SAT is *the* classical example of an $\mathcal{NP}$-complete problem

- TSP (decision variant) is $\mathcal{NP}$-complete

- *but:* not all combinatorial problems with large search spaces are hard (*cf.* 2-SAT / Horn-SAT, shortest path, minimal spanning tree)

Complexity of combinatorial optimisation problems

- complexity of optimisation vs. decision variants

- approximation algorithms

- approximability / inapproximability

Some results for the TSP:

- general TSP: inapproximable for arbitrary constant bounds on solution quality

- TSP with triangle inequality: polynomially approximable to $1.5\times$ optimal solution quality

- Euclidean TSP: polynomial approximation schema exists

Are computationally hard problems really intractable?

- $\mathcal{NP}$-hardness results apply to the worst case but do not imply that all instances of a problem are hard to solve

- approximate solutions are often useful and easier to compute

- heuristics can often help to solve practically important classes of instances in reasonable time

- randomisation can help to find good solutions reasonably efficient with high probability

- parallelisation can help to increase size of practically soluble instances

Search Methods

Types of search methods:

$$\text{systematic} \quad \longleftrightarrow \quad \text{local search}$$

$$\text{deterministic} \quad \longleftrightarrow \quad \text{stochastic}$$

$$\text{sequential} \quad \longleftrightarrow \quad \text{parallel}$$

Local Search (LS) Algorithms

search space S

> (SAT: set of all complete truth assignments
> to propositional variables)

solution set $S' \subseteq S$

> (SAT: models of given formula)

neighbourhood relation $\mathcal{N} \subseteq S \times S$

> (SAT: neighbouring variable assignments differ
> in the truth value of exactly one variable)

evaluation function $g : S \mapsto \mathbb{R}_0^+$

> (SAT: number of clauses unsatisfied
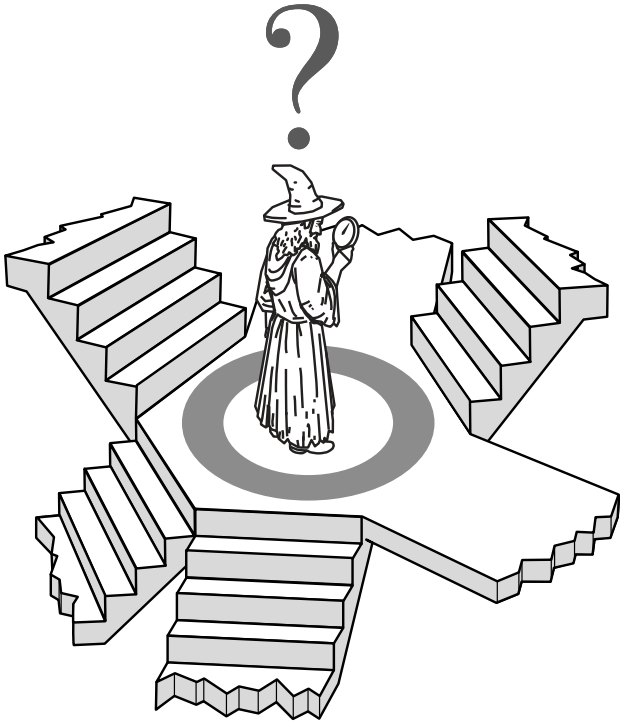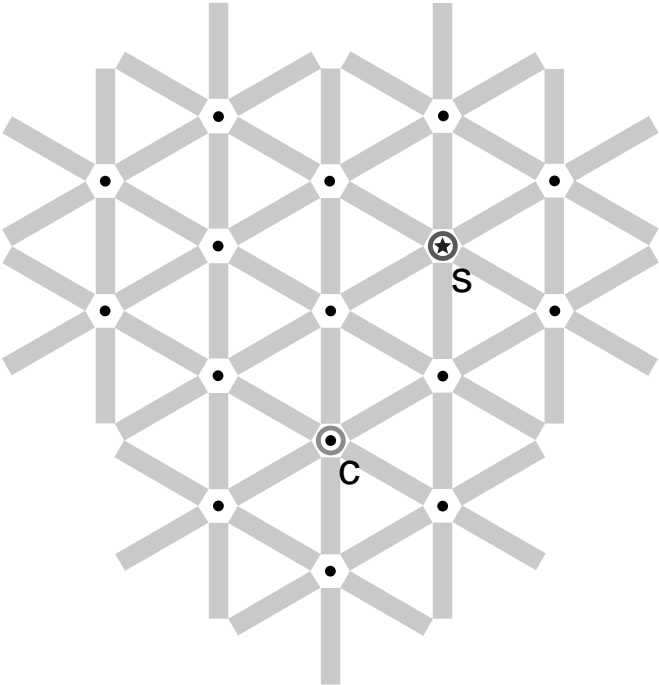> under given assignment)

Local Search:

- start from initial position

- iteratively move from current position to neighbouring position

- use evaluation function for guidance

Two main classes:

- local search on partial solutions

- local search on complete solutions

Local Search

Construction Heuristics

- specific class of LS algorithms

- search space: space of partial solutions

- search steps: extend partial solutions, but never reduce them

- neighbourhood typically given by individual solution elements

- solution elements are often ranked according to a greedy evaluation function

Nearest Neighbour heuristic for the TSP:

- at any city, choose the closest yet unvisited city

  - choose an arbitrary initial city $\pi(1)$

  - at the ith step choose city $\pi(i+1)$ to be the city j
    that minimises $d(\pi(i), j); j \neq \pi(k), 1 \leq k \leq i$

- running time: $\mathcal{O}(n^2)$

- worst case performance:
  $NN(x)/OPT(x) \leq 0.5(\lceil \log_2 n \rceil + 1)$

- other construction heuristics for TSP are available

Nearest neighbour tour through 532 US cities
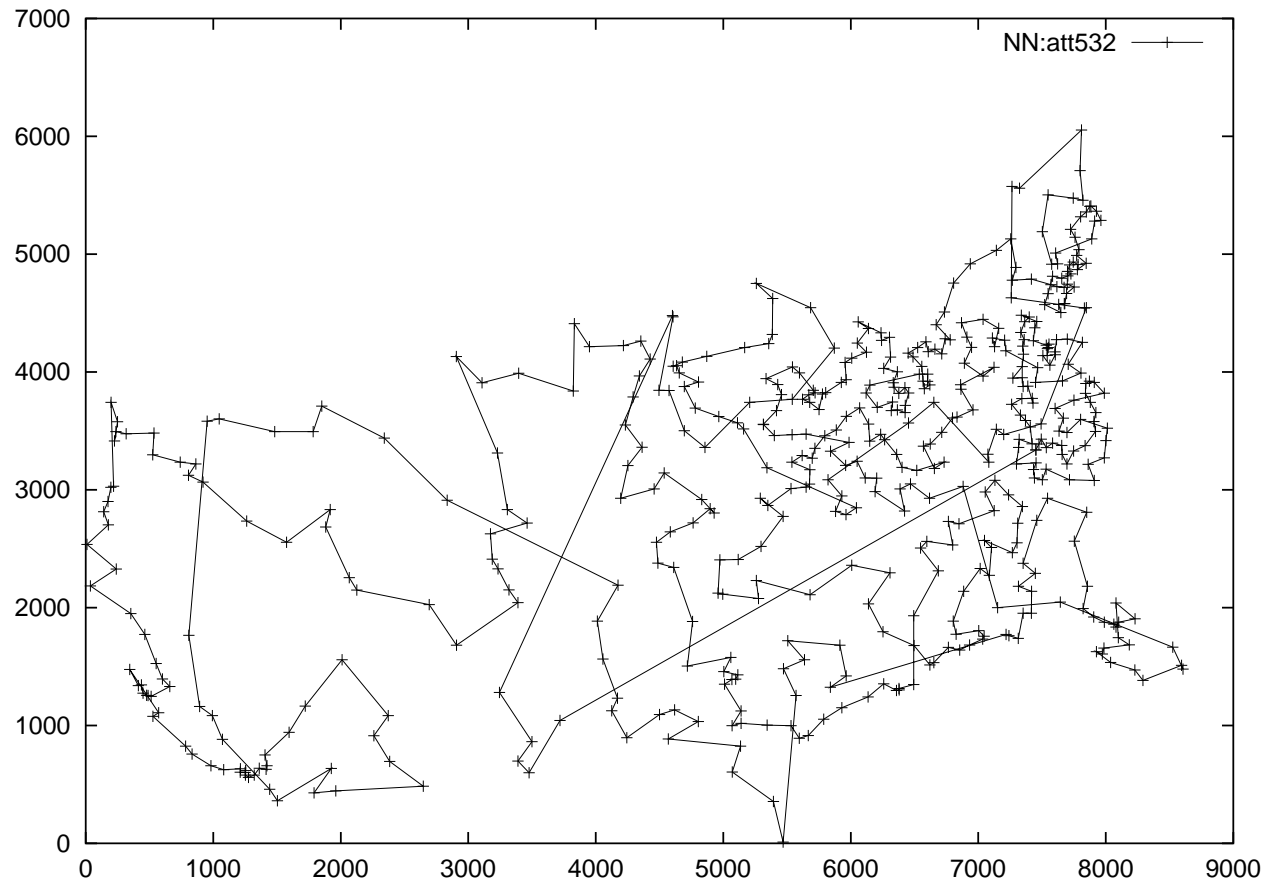
Construction Heuristics ...

- can be used iteratively to solve combinatorial problems

- provide only a limited number of different solutions

- can be combined with back-tracking to yield systematic search algorithms (e.g., Davis-Putnam for SAT)

- are used within some state-of-the-art local search approaches (e.g., Ant Colony Optimisation)

**Iterative Improvement (Greedy Search):**

- initialise search at some point of search space

- in each step, move from the current search position
  to a neighbouring position with better evaluation function
  value

Iterative Improvement for SAT

- initialisation: randomly chosen, complete truth assignment

- neighbourhood: variable assignments are neighbours iff they differ in truth value of one variable

- neighbourhood size: $\mathcal{O}(n)$ where $n =$ number of variables

- evaluation function: number of clauses unsatisfied under given assignment

Iterative Improvement for the TSP

- initialisation: complete tour, e.g., obtained from nearest neighbour heuristic

- k-exchange neighbourhood: solutions which differ by at most k edges

2-exchange
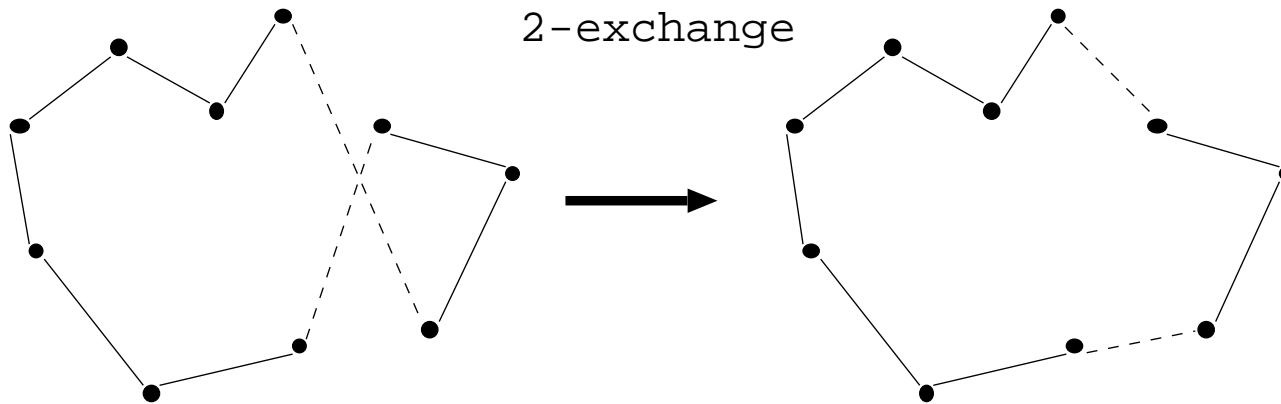
- neighbourhood size: $\mathcal{O}(n^k)$ where $n =$ number of cities

Stochastic Local Search

Typical problems with local search:

- getting stuck in local optima

- being misguided by evaluation/objective function

Stochastic Local Search:

- randomise initialisation step

    – random initial solutions

    – randomised construction heuristics

- randomise search steps
  such that suboptimal/worsening steps are allowed
  $\rightsquigarrow$ improved performance & robustness

- typically, degree of randomisation controlled by noise
  parameter

Pros:

- for many combinatorial problems more efficient than systematic search

- easy to implement

- easy to parallelise

Cons:

- often incomplete (no guarantees for finding existing solutions)

- highly stochastic behaviour

- often difficult to analyse theoretically / empirically

Simple SLS methods

Random Search (Blind Guessing):

In each step, randomly select one element of the search space.

(Uninformed) Random Walk:

In each step, randomly select one of the neighbouring positions of the search space and move there.

Randomised Iterative Improvement:

- initialise search at some point of search space

- search steps:

  - with probability p, move from current search position
    to a randomly selected neighbouring position

  - otherwise, move from current search position
    to neighbouring position with better evaluation function
    value

Part II

Stochastic Local Search Methods

Stochastic Local Search Methods – Overview

Parameterised local search extensions:

– Simulated Annealing

– Tabu Search

Hybrid SLS strategies:

- Iterated Local Search

- Evolutionary Algorithms

- Ant Colony Optimization

$\rightsquigarrow$ representation as Generalised Local Search Machines (GLSMs)

Simulated Annealing

Combinatorial search technique inspired by the

physical process of annealing [Kirkpatrick et al. 1983, Cerny 1985]

Outline

- generate a neighbour solution / state

- probabilistically accept the solution / state
  probability of acceptance depends on the objective function
  (energy function) difference and an additional parameter
  called temperature

Solution generation

- typically returns a random neighbouring solution

Acceptance criterion

- Metropolis acceptance criterion

  – better solutions are always accepted

  – worse solutions are accepted with probability

  $$\sim \exp\left(\frac{g(s) - g(s')}{T}\right)$$

Annealing

- parameter T, called temperature, is slowly decreased

Generic choices for annealing schedule

- initial temperature T_0
  (example: based on statistics of evaluation function)

- cooling schedule — how to change temperature over time
  (example: geometric cooling, $T_{n+1} = \alpha \cdot T_n, n = 0, 1, \ldots$)

- number of iterations at each temperature
  (example: multiple of the neighbourhood size)

- stopping criterion
  (example: no improved solution found for a
  number of temperature values)

Example application to the TSP [Johnson & McGeoch 1997]

- baseline implementation:

  – start with random initial solution

  – use 2-exchange neighborhood

  – simple annealing schedule

  $\leadsto$ relatively poor performance

- improvements:

  – look-up table for acceptance probabilities

  – neighbourhood pruning

  – low-temperature starts

Simulated Annealing …

- is historically important

- is easy to implement

- has interesting theoretical properties (convergence),
  but these are of very limited practical relevance

- achieves good performance often at the cost of
  substantial run-times

Tabu Search

Combinatorial search technique which heavily relies on the use of an explicit memory of the search process [Glover 1989, 1990]

- systematic use of memory to guide search process

- memory typically contains only specific attributes of previously seen solutions

- simple tabu search strategies exploit only short term memory

- more complex tabu search strategies exploit long term memory

Simple tabu search algorithm – exploiting short term memory

- in each step, move to best neighbouring solution
  although it may be worse than current one

- to avoid cycles, tabu search tries to avoid revisiting
  previously seen solutions

- avoid storing complete solutions by basing the memory on
  attributes of recently seen solutions

- tabu solution attributes are often defined via local search moves

- tabu list stores attributes of the tl most recently visited
  solutions; parameter tl is called *tabu list length* or *tabu tenure*

- solutions which contain tabu attributes are forbidden

- problem: previously unseen solutions may be tabu

  $\rightsquigarrow$ use of *aspiration criteria* to override tabu status

- stopping criteria:

  - all neighboring solutions are tabu

  - maximum number of iterations exceeded

  - number of iterations without improvement

- appropriate choice of tabu tenure critical for performance

  $\rightsquigarrow$ Robust Tabu Search [Taillard 1991], Reactive Tabu Search

  [Battiti & Tecchiolli 1994–1997]

Example: Tabu Search for SAT / MAX-SAT

[Hansen & Jaumard 1990; Selman & Kautz 1994]

Neighborhood: assignments which differ in exactly
one variable instantiation

Tabu attributes: variables

Tabu criterion: flipping a variable is forbidden
for a given number of iterations

Aspiration criterion: if flipping a tabu variable leads to
a better solution, the variable's tabu status is overridden

Tabu search — use of long term memory

Long term memory: often based on some measure of frequency, e.g., the frequency of local search moves

Intensification strategies: intensify the search in specific regions of the search space

- recover *elite* solutions and restart search around such solutions

- lock some solution attributes, e.g., in the TSP, edges contained in several elite solutions may be locked

Diversification Strategies: drive the search towards previously unexplored search space regions

- introduce solution attributes which are not very frequently used, e.g., by penalizing frequently used solution attributes

- restarting mechanisms which bias construction heuristics

Tabu Search ...

- performs often astonishingly well even when using only short term memory strategies

- can perform considerably better if additional intensification and diversification strategies are used

- can be enhanced with several additional strategies (e.g., strategic oscillation, path relinking, ejection chains, ...)

- often achieves very good performance, but may require time-intensive fine-tuning

Hybrid stochastic search techniques

Note: Many of the best-performing SLS algorithms are *combinations* of various simple search strategies.

E.g.: greedy hillclimbing + Random Walk, Ant Colony Optimisation + 3-opt, …

$\rightsquigarrow$ conceptual separation of simple search strategies and (higher-level) search control

GLSMs – Generalised Local Search Machines

- search control = non-deterministic finite state machine

- simple search strategies = states

- change of search strategy = transitions between states

State transition types:

- deterministic: DET

- conditional: COND(C)

- unconditional probabilistic: PROB(p)

- conditional probabilistic: CPROB(C,p)

The GLSM model ...

- allows adequate and uniform represention of local search algorithms

- facilitates design, implementation, and analysis of hybrid algorithms

- provides the conceptual basis for some of the best known SLS algorithms for various domains (e.g., SAT [Hoos 1999])

# GLSM representation of Randomised Best Improvement
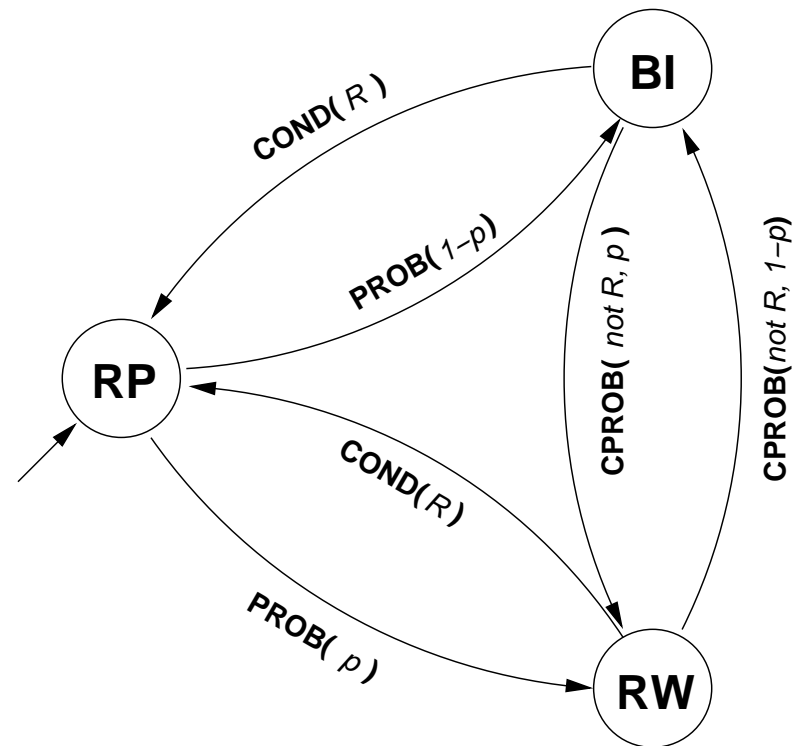
Iterated Local Search (ILS)

Iterative application of local search to modifications of previously visited local minima

- basic idea: build a chain of local minima

- the search space is reduced to the space of local minima

- simple, but powerful way to improve local search algorithms

GLSM representation of Iterated Local Search
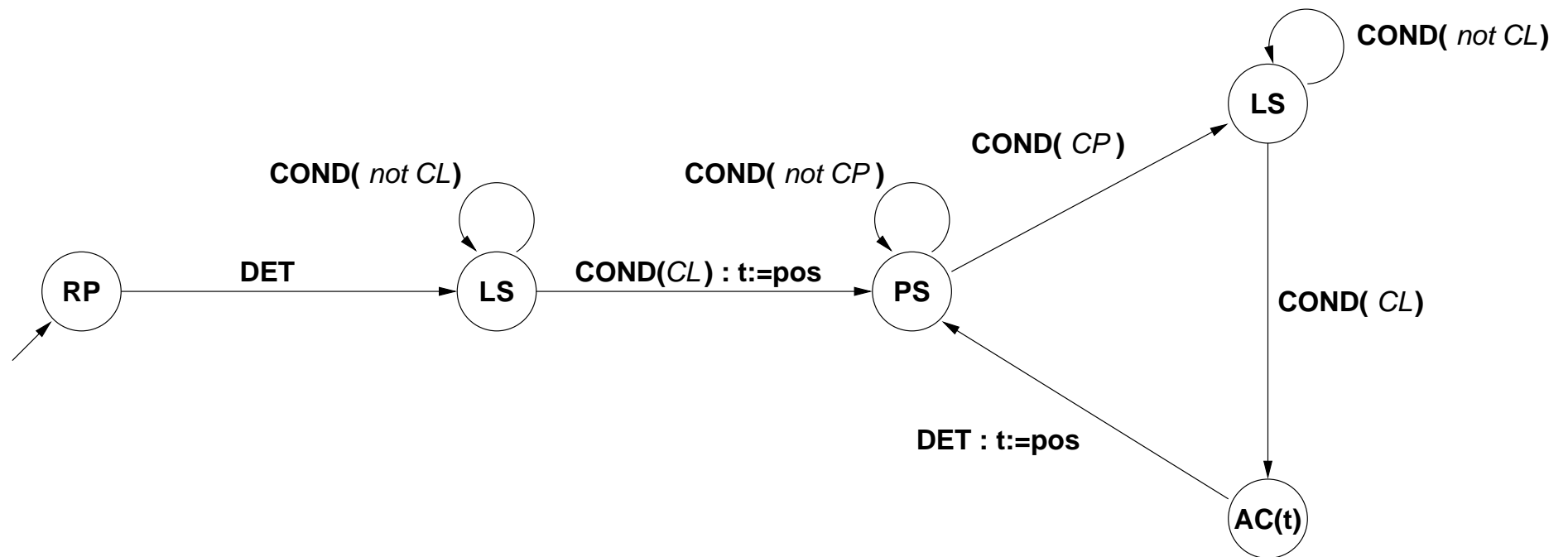
Issues for Iterated Local Search applications

- choice of initial solution

- choice of solution modification

    — Too strong: close to random restart

    — Too weak: insufficient for escaping from local minima

- choice of local search

    — effectiveness versus speed

- choice of acceptance criterion

    — strength of bias towards best found solutions

ILS for the TSP

- local search: 2-opt, 3-opt, Lin-Kernighan, Helsgaun LK

- solution modification: non-sequential 4-opt move
  (double-bridge move)

- acceptance criterion: apply solution modification to best
  solution since start of the algorithm; other acceptance criteria
  may perform better for long run times

Results

- some of the best algorithms for the TSP are based on ILS

Iterated Local Search ...

- is based on a simple principle

- is easy to implement (basic versions)

- has few parameters

- is highly effective

Related idea:

- Variable Neighbourhood Search

Evolutionary Algorithms

Combinatorial search technique inspired by the evolution of
biological species.

- population of individual solutions represented as strings

- individuals within population are evaluated based on their
  "fitness" (evaluation function value)

- population is manipulated via *evolutionary operators*

  - mutation

  - crossover

  - selection

Several types of evolutionary algorithms:

- Genetic algorithms [Holland 1975; Goldberg 1989]

- Evolution strategies [Rechenberg 1973; Schwefel 1981]

- Evolutionary Programming [Fogel et al. 1966]

- Genetic Programming [Koza 1992]

For combinatorial optimization, genetic algorithms are the most widely used and most effective variant type

GLSM representation of a basic Genetic Algorithm
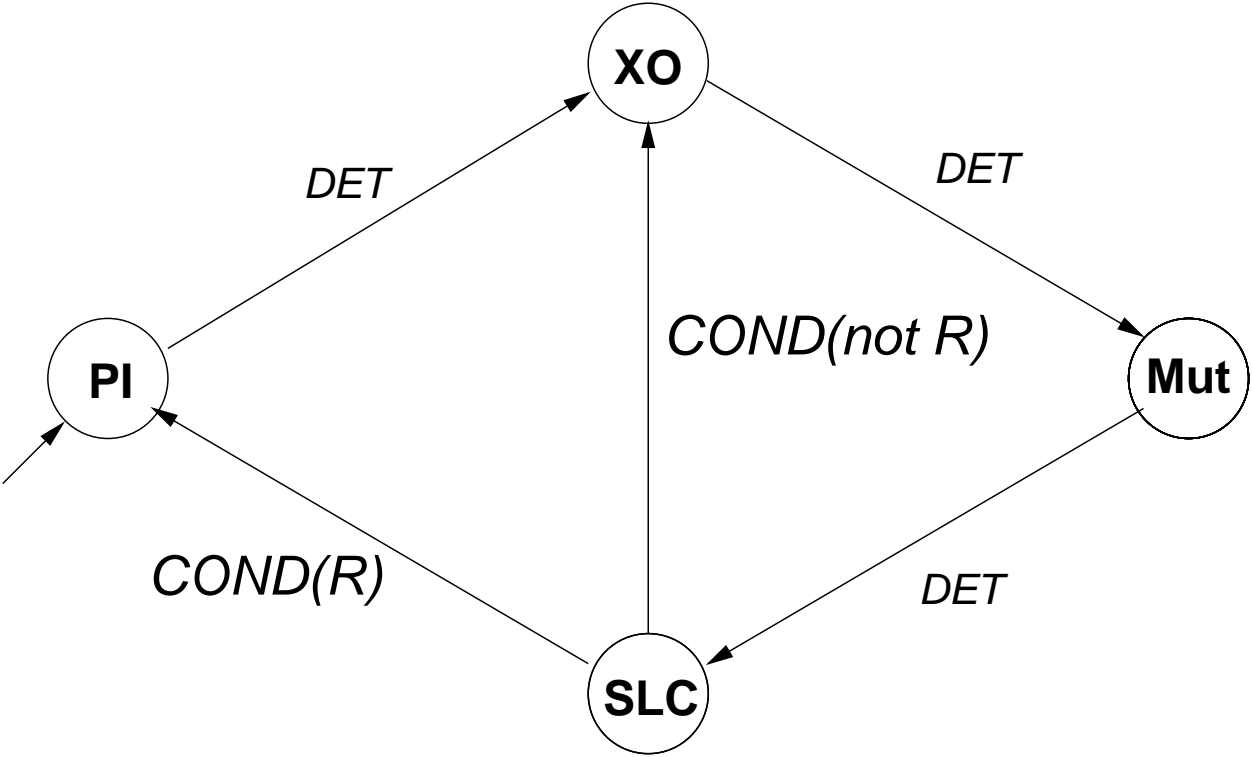
Important issues for Evolutionary Algorithms

- solution representation

  - binary vs. problem specific representation

- fitness evaluation of solutions

  - often defined by objective function of the problem

- crossover operator

  - parent selection scheme

  - problem specific vs. general purpose crossover

  - passing of meaningful information from parents to offspring

- mutation operator

  - background operator vs. driving the search

- selection scheme

    - prefer better solutions for survival

    - elitist strategies

    - maintenance of population diversity

- local search

    - often useful for improving performance

    - population based search in the space of local optima
      $\rightsquigarrow$ memetic algorithms

- stopping criteria

    - fixed number of generations

    - convergence of population

Evolutionary Algorithms . . .

- use populations, which leads to increased search space exploration

- allow for a large number of different implementation choices

- typically reach best performance when using operators that are based on problem characteristics

- achieve good performance on a wide range of problems

Ant Colony Optimisation

Combinatorial search technique inspired by the foraging
behaviour of real ants: [Dorigo et al. 1991, 1996]

- population of simple agents ("ants") communicates indirectly
  via similated "pheromone trails"

- ants follow a local stochastic policy to construct solutions

- the solution construction is probabilistically biased by
  pheromone trail information, heuristic information,
  and the partial solution of each ant

- Pheromone trails are modified during the algorithm's
 execution to reflect the search experience

GLSM representation of Ant Colony Optimization
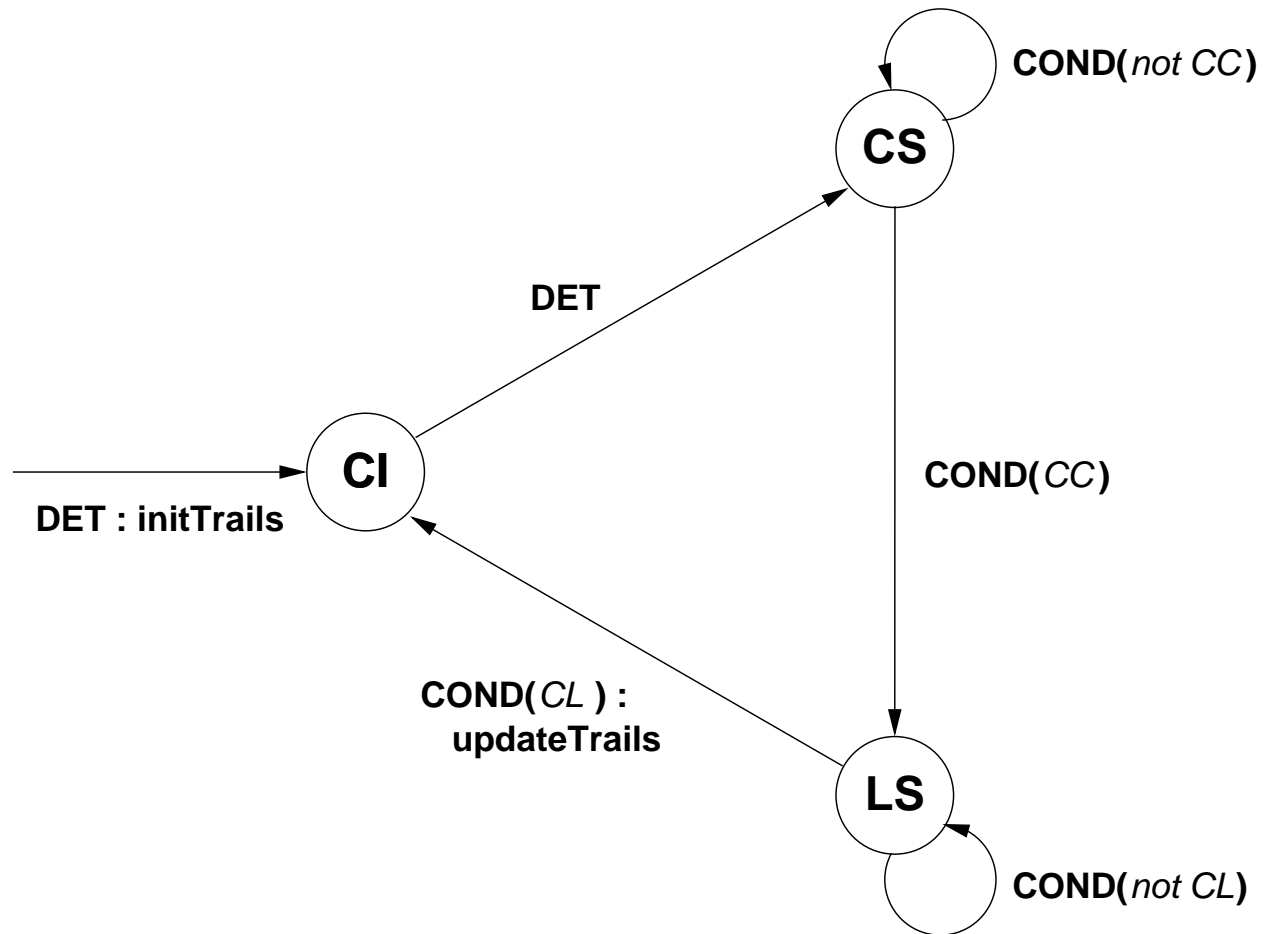
Key ideas:

- Define *solution components* for the problem to be solved

- Ants construct solutions by iteratively adding
  solution components

- Possibly improve solutions by applying local search

- Reinforce solution components of better solutions
  more strongly

Construction Process in Ant Colony Optimisation

$$\tau_{\texttt{ij}}, \eta_{\texttt{ij}}$$
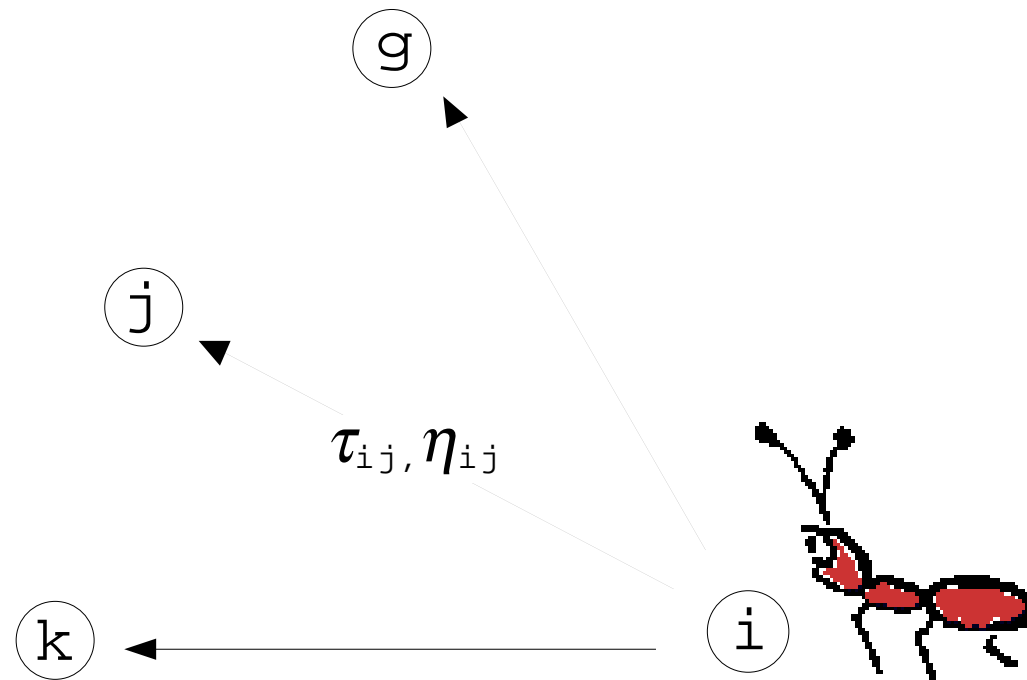
Example: ACO for the TSP – tour construction

- Solution components are edges of given graph

- $\eta_{ij} = 1/d_{ij}$: Heuristic information, indicates the utility of going from city i to city j

- $\tau_{ij}(t)$: Intensity of the pheromone trail in iteration t on edge (i, j)

- Probabilistic selection of the next city according to:

$$p_{ij}(t) \sim (\tau_{ij}(t))^\alpha \cdot (\eta_{ij})^\beta \ \ \text{if city } j \text{ not yet visited}$$

Example: ACO for the TSP – Update of pheromone trails

- Parameter $0 < \rho < 1$, $1 - \rho$ represents pheromone evaporation

- Update of the pheromone trails according to:

$$\tau_{ij}(t) = \rho \cdot \tau_{ij}(t-1) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

- $\Delta\tau_{ij}^{k} = 1/L_k$ if edge (i,j) is used by ant k on its tour
  where L_k = tour length of ant k; m = number of ants

(Several improved extensions have been proposed.)

Ant Colony Optimisation . . .

- has been successfully applied to static and dynamic
  combinatorial problems

- has shown very good performance on a range of problems
  including (abstract) protein folding problems
  [Shmygelska & Hoos 2002–2004]

New book:

M. Dorigo and T. Stützle. *Ant Colony Optimization.* MIT Press,
2004.

Characteristics of Various SLS Methods

Feature	SA	TS	ILS	GA	ACO
Single trajectory	+	+	−	−	−
Population	−	−	−	+	+
Memory	−	+	∃	∃	∃
Multiple neighborhoods	−	−	+	∃	−
Sol. construction	−	−	−	−	+
Nature-inspired	+	−	−	+	+

$+$: feature present, $\exists$: partially present, $-$: not present

Part III

Analysing and Characterising
Stochastic Search Behaviour

Analysing Stochastic Search Behaviour

Many SLS algorithms ...

- perform well in practice

- are incomplete, i.e., cannot be guaranteed to find (optimal) solutions

- are hard to analyse theoretically

$\rightsquigarrow$ empirical methods are used to analyse and characterise SLS behaviour.

Aspects of stochastic search performance:

- variability due to randomisation

- robustness w.r.t. parameter settings

- robustness across different instance types

- scaling with problem size

Insights into algorithmic performance...

- help assessing suitability for applications

- provide basis for comparing algorithms

- characterise algorithm behaviour

- facilitate improvements of algorithms

RTD-based empirical methodology:

- run algorithm multiple times on given problem instance(s)

- estimate empirical run-time distributions (RTDs)

- get simple descriptive statistics (mean, stddev, percentiles, ...) from RTD data

- approximate empirical RTDs with known distribution functions

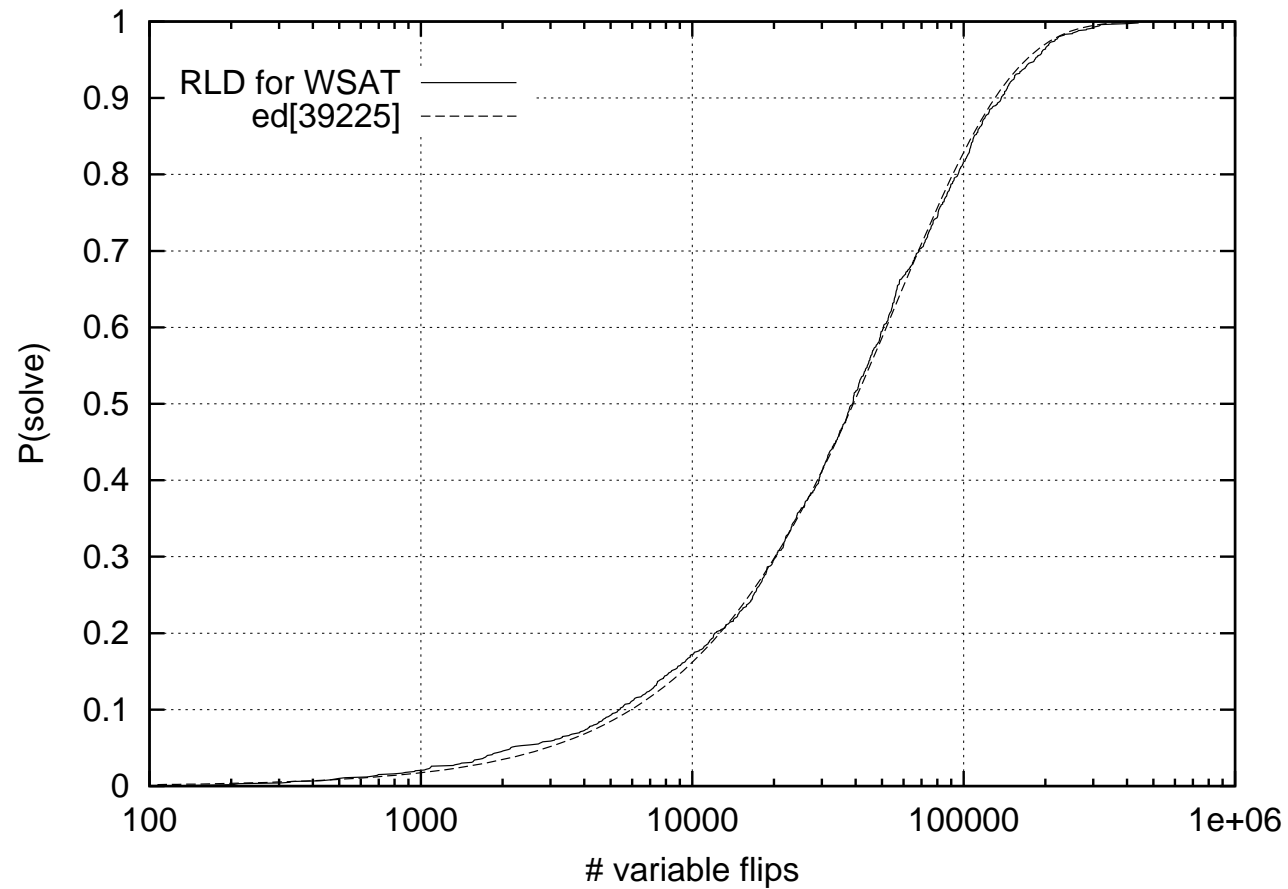- check statistical significance using goodness-of-fit test

[Hoos & Stützle 1998]

Raw run-time data (each spike one run)
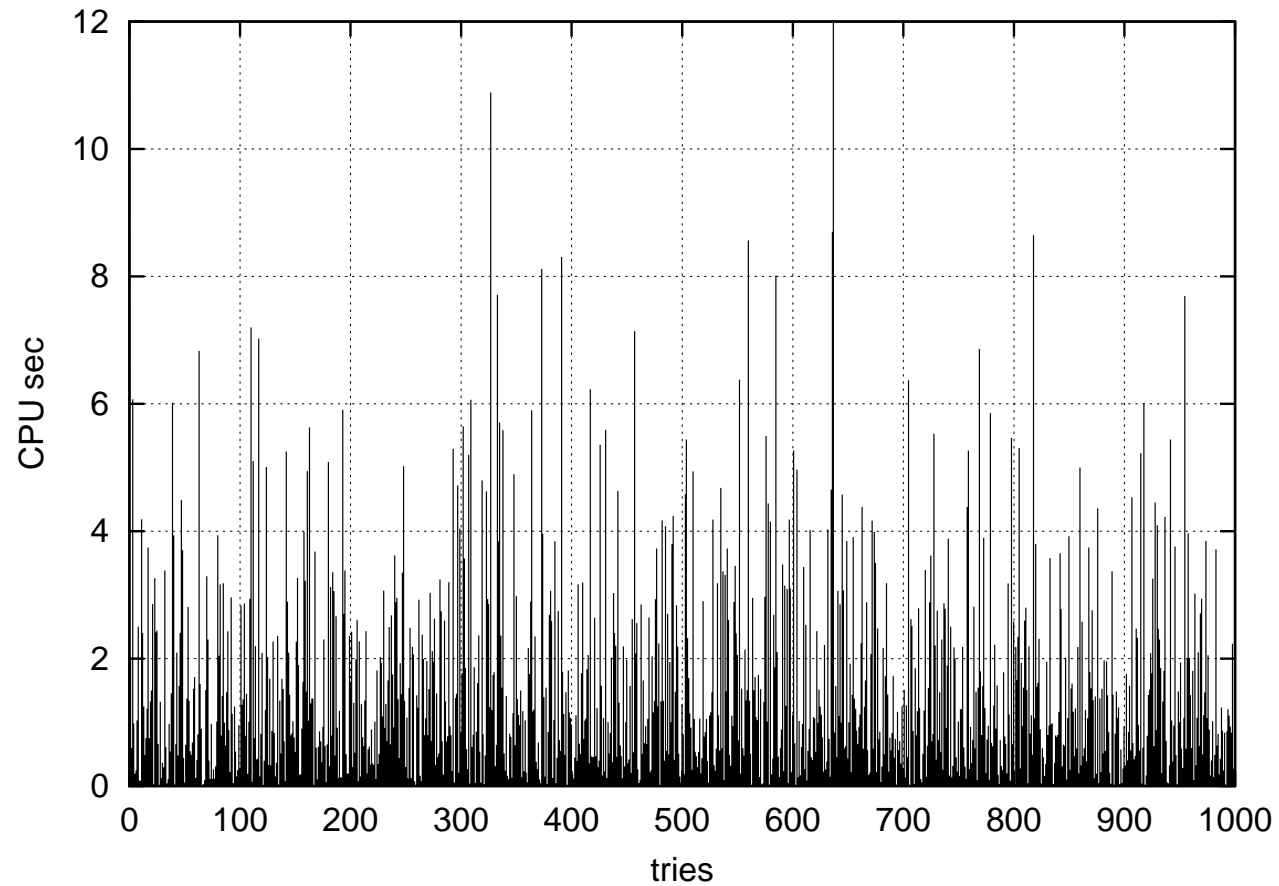
RTD graph and approximation with exponential distribution

This methodology facilitates ...

- precise characterisations of run-time behaviour

- prognoses for arbitrary cutoff times

- empirical analysis of asymptotic behaviour

- fair and accurate comparison of algorithms

- cleanly separating different sources of randomness
  (SLS algorithm / random generation of problem instances)

Asymptotic run-time behaviour of SLS algorithms

- *complete*

  — for each problem instance P there is a time bound $t_{max}(P)$
  for the time required to find a solution

- *probabilistic approximate completeness (PAC property)*

  — for each soluble problem instance a solution is found
  with probability $\rightarrow 1$ as run-time $\rightarrow \infty$.

- *essential incompleteness*

  — for some soluble problem instances, the probability
  for finding a solution is strictly smaller 1 for run-time $\rightarrow \infty$.

Some results [Hoos 1999]:

- Until recently, some of the most prominent and best-performing SLS algorithms for SAT were essentially incomplete.

- In practice, essential incompleteness often causes stagnation behaviour which drastically affects the performance of the algorithm.

- By a simple and generic modification, (Random Walk Extension) these algorithms can be made PAC in a robust manner.

- The algorithms thus obtained are among the best-performing SLS algorithms for SAT known to date.

Comparative performance analysis on single problem instance:

- measure RTDs

- check for probabilistic domination (crossing RTDs)

- use statistical tests to assess significance of
 performance differences (e.g., Mann-Whitney U-test)

Performance comparison for ACO and ILS algorithm for TSP
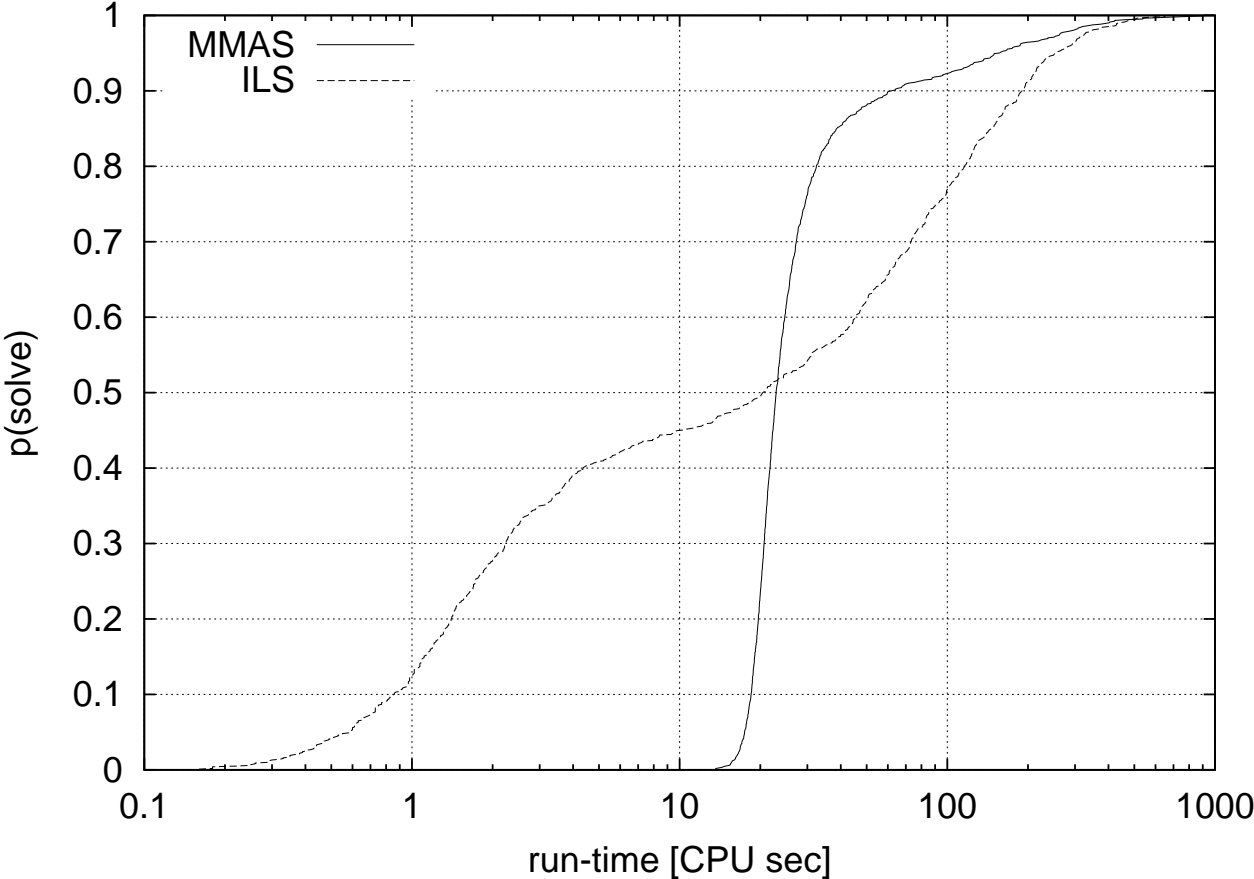
Comparative performance analysis for ensembles of instances:

- check for uniformity of RTDs

- partition ensemble according to probabilistic domination

- analyse correlation for (reasonably stable) RTD statistics

- use statistical tests to assess significance of performance differences across ensemble (e.g., Wilcoxon matched pairs signed-rank test)

Peformance correlation for ACO and ILS algorithm for TSP
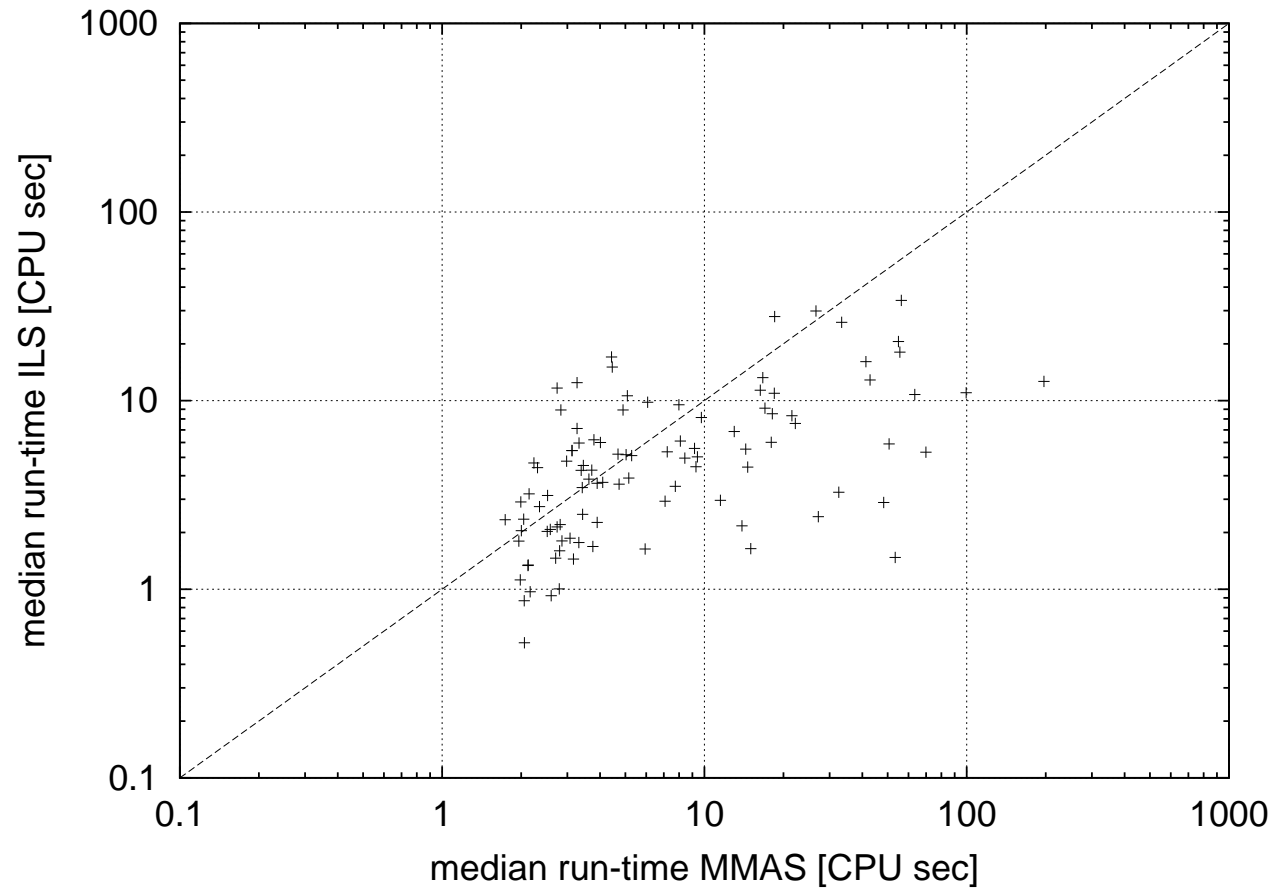
RTD-based analysis of randomised optimisation algorithms:

- additionally, solution quality has to be considered

- introduce bounds on the desired solution quality
  $\rightsquigarrow$ qualified RTDs

- bounds can be chosen w.r.t. best-known or optimal solutions, lower bounds of the optimal solution cost etc.

- estimate run-time distributions for several bounds on the solution quality

Qualified RTDs for TSP instance att532 with ILS

Run-time distributions for iterated 3-opt on att532.tsp
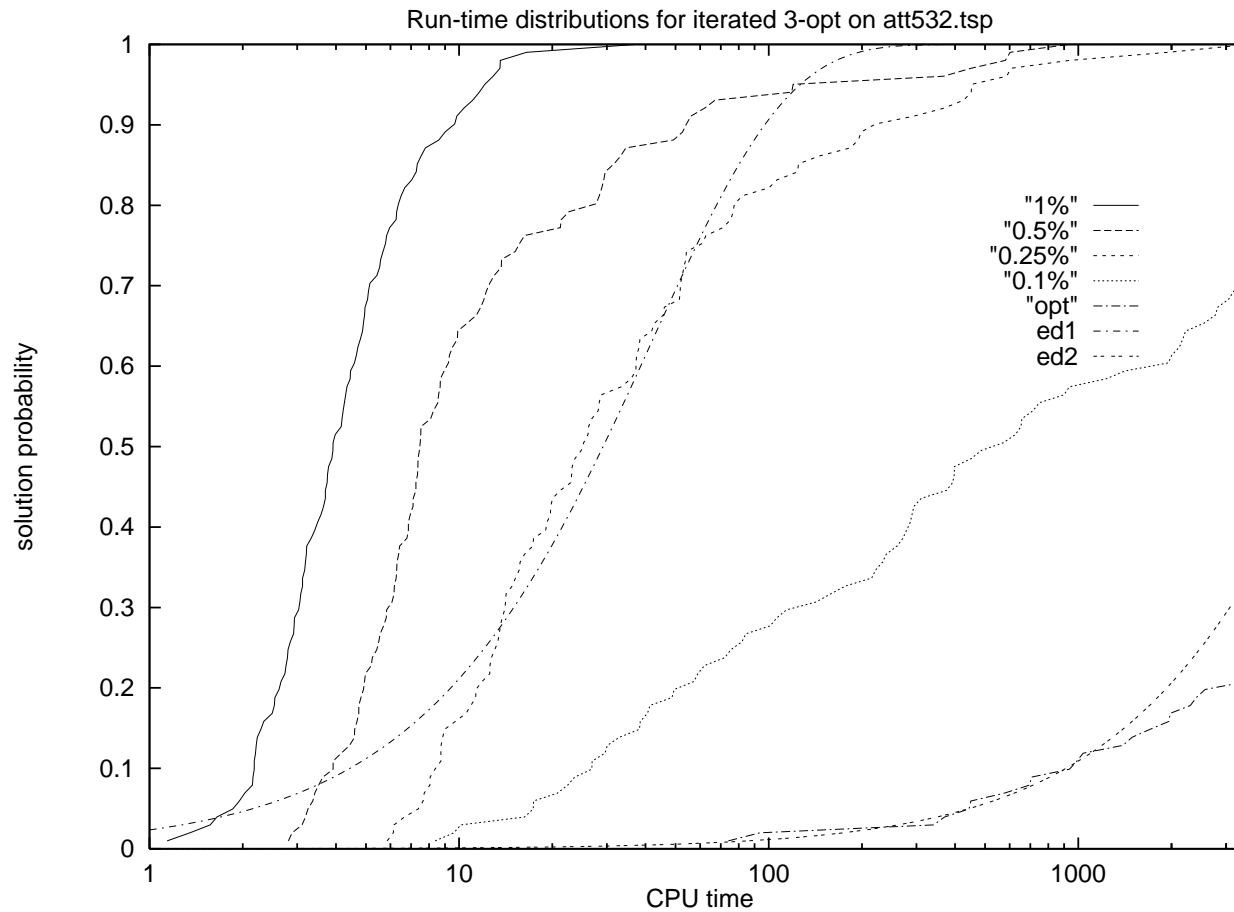
SQD-based methodology:

- run algorithm multiple times on given problem instance(s)

- estimate empirical solution quality distributions (SQDs) for different run-times

- get simple descriptive statistics (mean, stddev, percentiles, ...) from SQD data

- approximate empirical SQD with known distribution functions

- check statistical significance using goodness-of-fit test

SQD analysis for Graph Partitioning (GP)
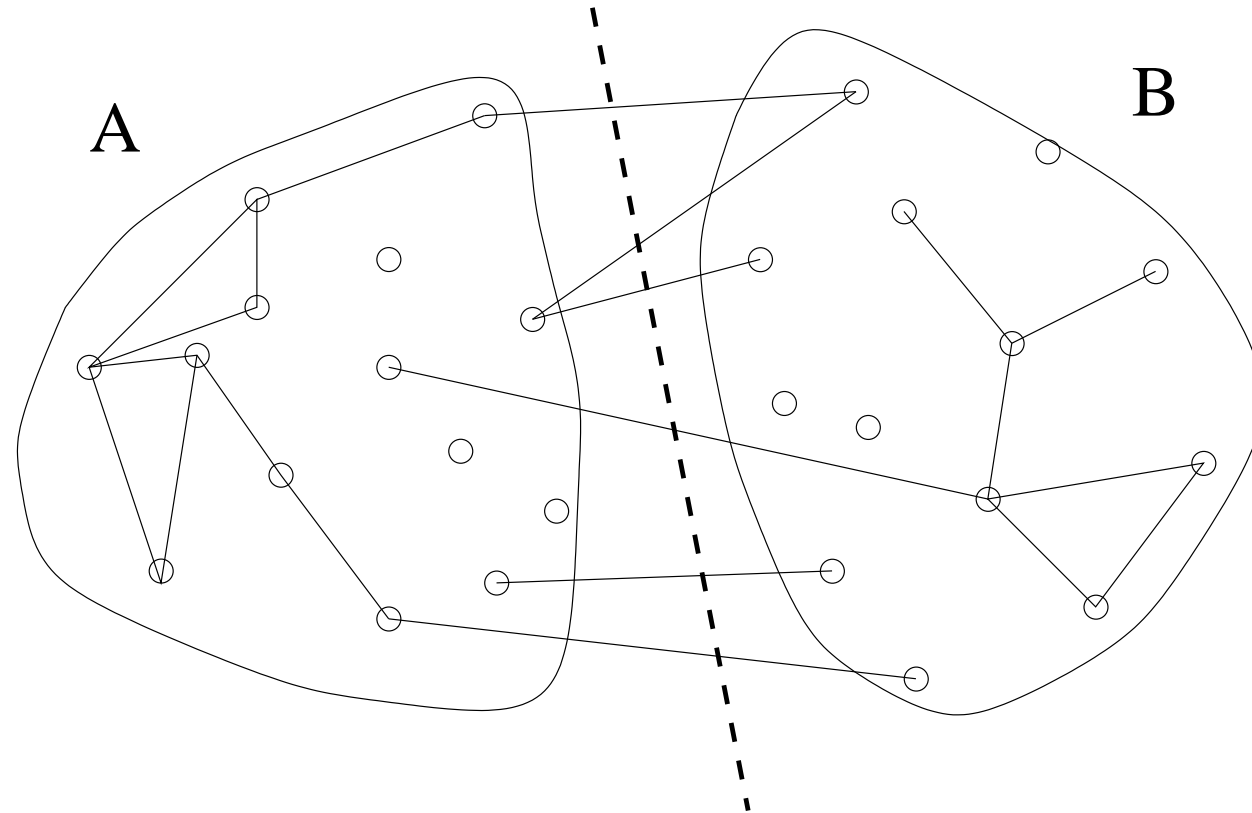
[Martin et al. 1999] studied best performing SLS algorithms for GP on ensemble of randomly generated graphs.

Results:

- SQD distributions approach a limiting Gaussian shape, both for individual graphs and across ensembles.

- For increasing instance size SQD distributions become increasingly peaked.
  $\rightsquigarrow$ solution quality becomes dominating factor when comparing SLS algorithms on large instances.

- Similar results also for the TSP.

Using SQDs for estimating optimum solution qualities

Consider sample of k feasible solutions and let x be the extreme value from the sample.

$\rightsquigarrow$ for large k, the distribution of x approaches a Weibull distribution with position parameter μ, where μ is optimal solution quality [Dannenbring 1977]

Estimation procedure:

- generate m independent samples of x

- estimate parameters of Weibull distribution

- obtain confidence interval for optimum value

Search Space Analysis

Intuition:

- SLS algorithm moves in a *search landscape* induced by the given problem instance and aspects of algorithm

- search landscape can be imagined as a mountainous region with peaks, basins, valleys, saddles, . . .

- goal of search process is to find lowest point in this landscape (for minimization problems)

⤳ connection between SLS behaviour and landscape structure

Search landscape is defined by:

- the set of all possible solutions (search space)

- an evaluation function that assigns to every solution a solution quality value

- a neighbourhood relation, which induces a distance measure between solutions

Distance between solutions:

- defined as the minimum number of search steps needed
  to reach one solution from the other

- often surrogate distance metrics are used
  (e.g., for the TSP, distance between tours measured by
  number of edges contained in one tour, but not the other)

Important aspects of search landscapes:

- number of local optima

- ruggedness

- distribution of local minima and their relative location to the global minima

- size, topology, and location of plateaus and basins

- connections between plateaus and basins

Two widely used types of search space analysis:

- analysis of search space ruggedness

- analysis of (linear) correlation between solution fitness and distance to global optima (fitness-distance correlation)
  e.g., [Boese 1994, 1996; Jones & Forrest 1995]

Measures for landscape ruggedness:

- autocorrelation function [Weinberger 1990; Stadler 1995]

- correlation length [Stadler 1995]

- autocorrelation coefficient [Angel & Zissimopoulos 1998, 1999]
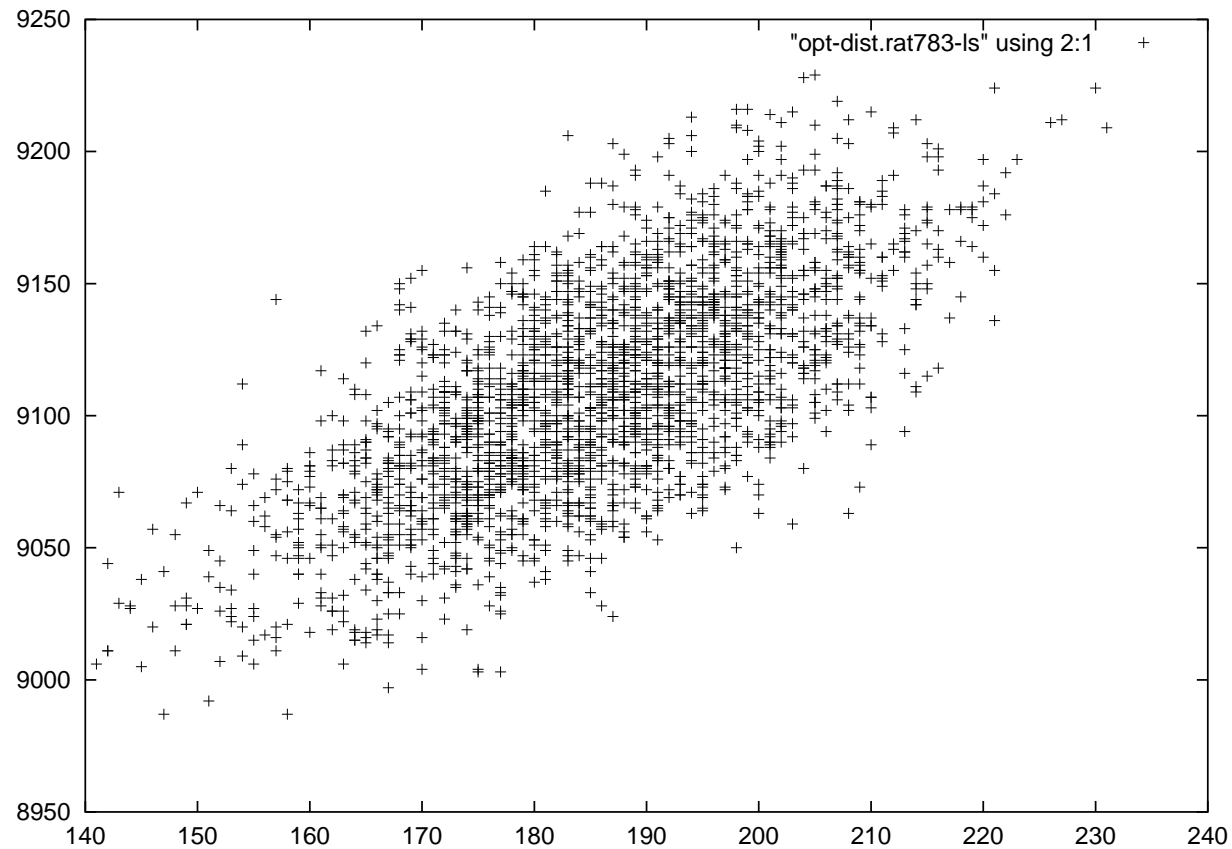
Measure for fitness-distance correlation:

- correlation coefficient

$$\rho(F, D) = \frac{\mathbf{Cov}(F, D)}{\sqrt{\mathbf{Var}(F)} \cdot \sqrt{\mathbf{Var}(D)}}$$

- graphical visualization through plots of fitness–distance relationship

Fitness–distance relationship in TSP instance rat783

Some results for the TSP:

instance	Δ_{avg}	avg_{d-opt}	avg_{d-opt}/n	ρ_{ls}
lin318.tsp	3.56	67.25	0.211	0.469
rat783.tsp	4.85	204.24	0.261	0.624
pcb1173.tsp	5.91	274.34	0.234	0.585
pr2392.tsp	5.71	552.49	0.231	0.538

Δ_{avg}: Percentage deviation from optimum

avg_{d-opt}: Average distance from optimum

ρ_{ls}: Correlation coefficient

Advanced Search Space Analysis

- **Idea:** collapse states on the same plateau or in the same basin
  into "macro states" and illustrate connections between
  these regions
  $\rightsquigarrow$ Plateau Connection Graphs (PCG),
      Basin Partition Trees (BPT)

- Analysing PCG or BPT properties and structure can give
  deeper insights into SLS behaviour and problem hardness
  than global measures, such as FDC or ACC.

- But: PCG and BPT analysis is computationally expensive and
  requires enumeration of large parts of the search space.

(Partial) Plateau Connection Graph for hard random 3-SAT instance
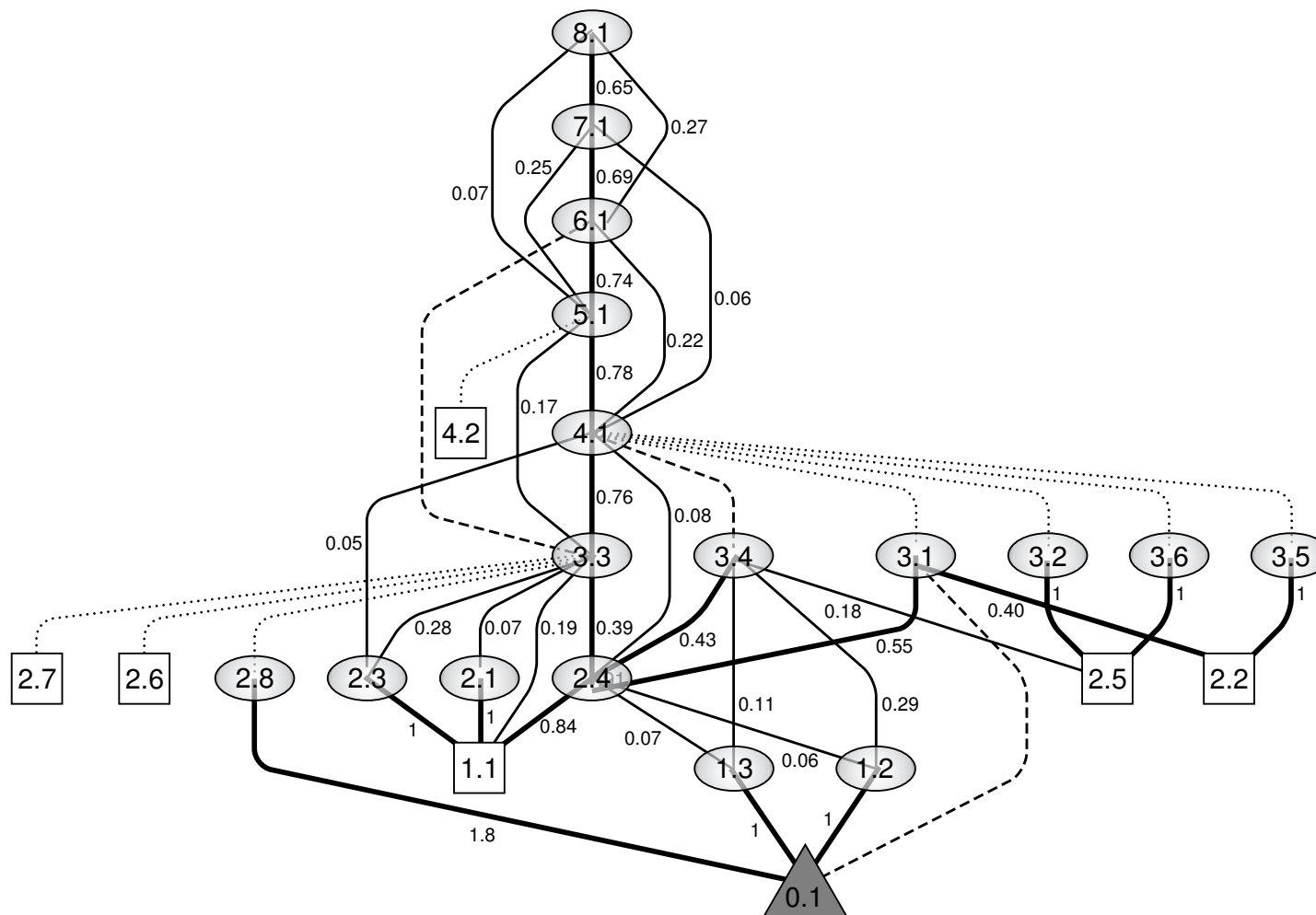
(Partial) Plateau Connection Graph for easy random 3-SAT instance
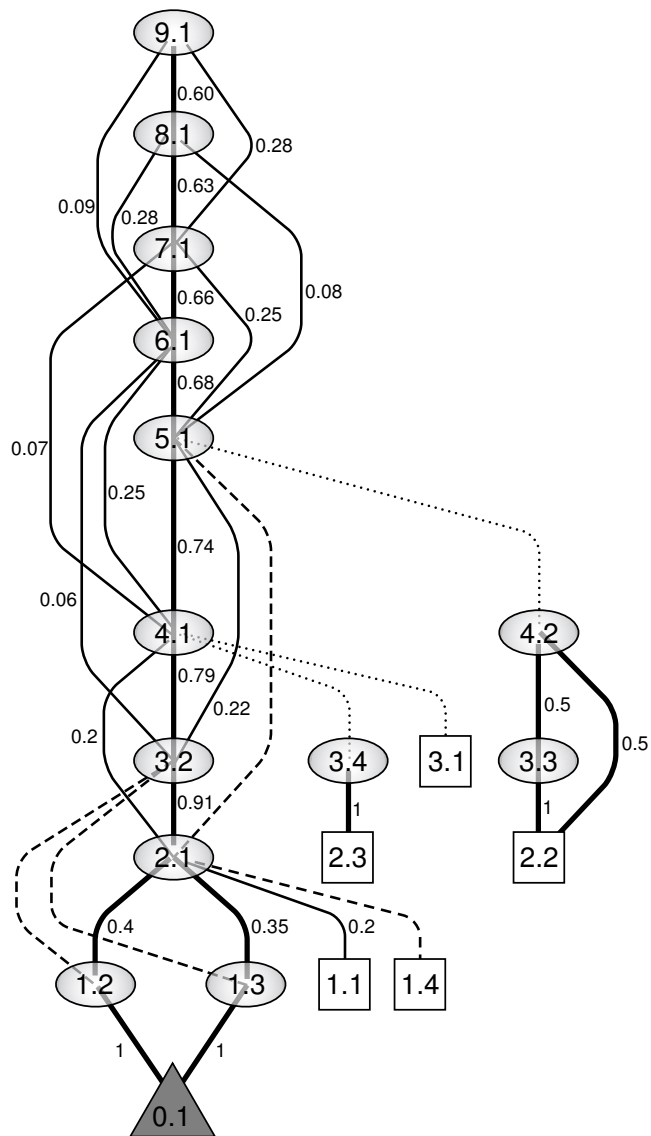
Paramaterisation and Tuning

- performance of stochastic search algorithms depends
  very strongly on appropriate parameterisation and tuning

- tuning can be very time-intensive

- limited understanding of how performance depends on
  parameter settings

- many stochastic search algorithms leave
  important implementation choices to the user

- experience with stochastic search algorithms is required
  to obtain best performance

Difficulties with parameterisation

- SLS algorithms are often highly stochastic
  $\rightsquigarrow$ empirical analysis more difficult

- algorithm parameters are not independent

- seemingly small implementation choices can have significant effects on algorithm behaviour and performance

Possible remedies

- use of adequate empirical methodology (statistical techniques) for analysing behaviour

- automatic parameter adjustment during search
  $\rightsquigarrow$ reactive search

Parallelising Stochastic Search

- randomised algorithms allow extremely easy and scalable parallelisation: multiple independent tries

- effectiveness depends on run-time behaviour of underlying algorithms

- under certain conditions, optimal speedup can be obtained

Analysis of the distribution type of RTDs for various well-performing SLS algorithms for a number of problem classes

$\rightsquigarrow$ **Result [Hoos & Stützle 1998]:**

For optimal parameterisations and applied to hard problem instances, many state-of-the-art SLS algorithms show exponential run-time distributions (EPAC property).

Goodness of fit for hard Random-3-SAT instances
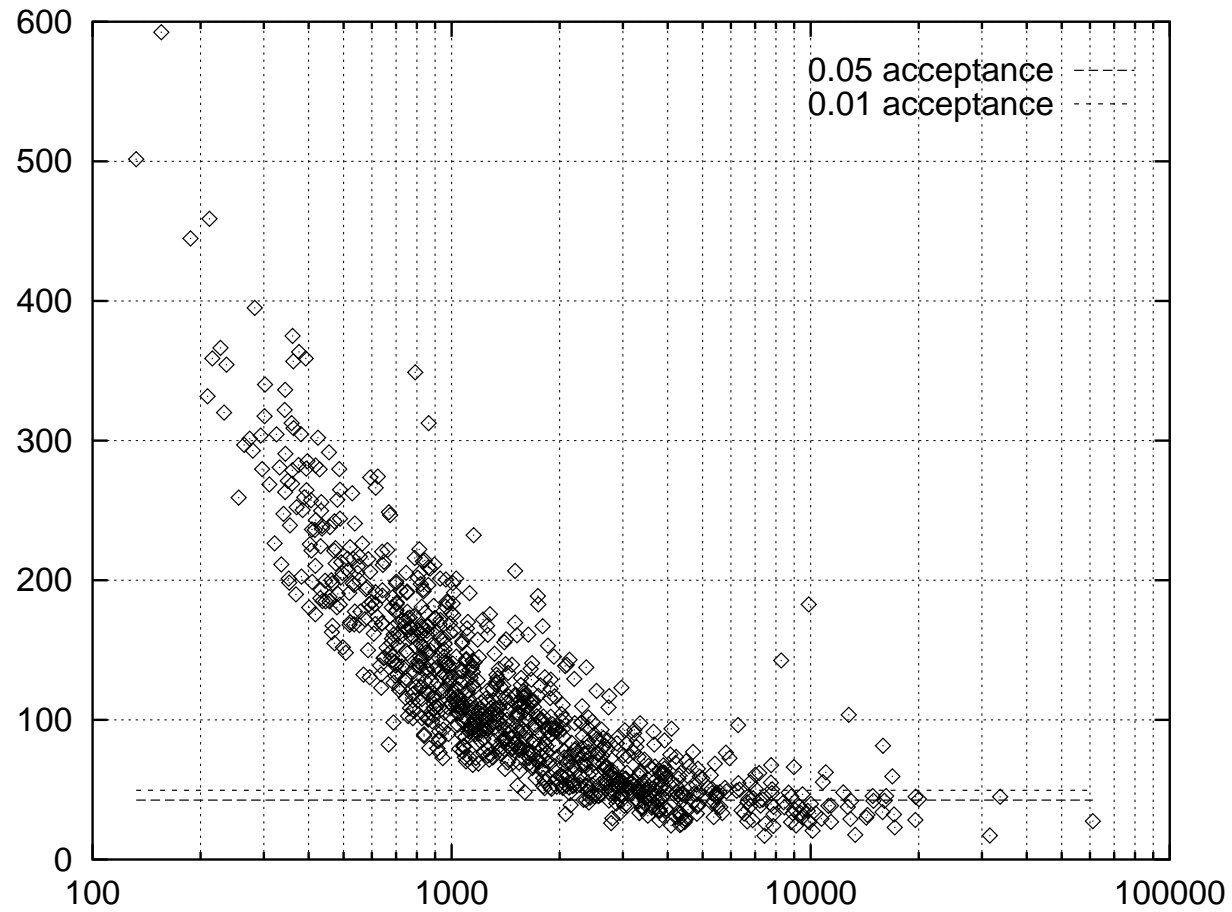
EPAC property implies:

- search is "memory-less"

  $\leadsto$ "random restart" is ineffective

- optimal speedup can be achieved through
  "multiple independent tries" parallelisation
  (very easy to implement, almost no communication overhead,
  arbitrarily scalable)

- interpretation of SLS behaviour as "random picking"
  from "virtual search space" (whose size can be computed
  from RTD data)

Simulated Annealing

- course-grained parallelisation: independent parallel runs

- fine-grained parallelisation: parallel evaluation of moves

Tabu Search

- exponential run-time distributions (QAP)

- cooperative approaches [Crainic et al.]

ACO algorithms

- course-grained parallelisation more successful
  than fine-grained

Evolutionary Algorithms

- fine-grained and coarse-grained parallel implementations

Part IV

Applications

Applications of Stochastic Search

Some recent successful applications:

- SAT [e.g., Selman et al. 1992–1997; Hoos et. al. 1994–2004]

- MAX-SAT [e.g., Hansen and Jaumard 1990; Hoos et al. 2003-2004]

- TSP [e.g., Johnson & McGeoch 1997–2002; Stützle & Hoos 1997–2004]

- Scheduling [e.g., Congram et al. 2002, den Besten, Stützle, Dorigo 2001]

- Time-Tabling [e.g., Rossi-Doria et al. 2003; Chiarandini et al. 2003]

- Protein Folding (HP model) [e.g., Shmygelska, Hoos, Aguirre-Hernandez 2002–2004]

Propositional Satisfiability (SAT)

Progress in SAT solving:

classic SAT solvers: based on 'Davis-Putnam' algorithm
(systematic search based on back-tracking)

1990–1992: successful application of SLS algorithms for solving
hard SAT instances [Selman et al.; Gu]

1993–1994: new, hybrid SLS strategies with improved
performance and robustness [Selman et al.; Gent & Walsh]

1996–1997: improvements in SLS algorithms [McAllester et al.];
randomised systematic search methods [Gomes et al.]

1998–2000: further improvements in SLS algorithms, based on GLSM model [Hoos & Stützle]

2001–2004: high-performance dynamic local search algorithms [Schuurmans et al., Hoos et al.], self-tuning /adaptive WalkSAT [Kautz et al., Hoos]

SLS algorithms for SAT

- **search space:** set of all complete variable assignments

- **solutions:** models of the given formula

- two assignments are **neighbours** if they differ in the value of exactly one variable

- **evaluation function** is the number of clauses unsatisfied under a given assignment

- **search initialisation:** randomly chosen assignment

- **search steps:** algorithm dependent, use random choices and/or tie-breaking rules

- most algorithms use random restart if no solution has been found after a fixed number of search steps

The GSAT Algorithm [Selman, Mitchell, Levesque 1992]

- in each search step, flip the variable which gives
  the maximal increase (or minimal decrease)
  in the number of unsatisfied claused

- ties are broken randomly

- if no model found after *maxSteps* steps, restart from
  randomly chosen assignment

The GWSAT Algorithm [Selman, Kautz, Cohen 1994]

- **search initialisation:** randomly chosen assignment

- **Random Walk step:** randomly choose and flip a variable occurring in a currently unsatisfied clause

- **GWSAT steps:** choose probabilistically between a GSAT step and a Random Walk step with 'walk probability' (noise) *wp*

The WalkSAT algorithm family [McAllester et al. 1997]

- **search initialisation:** randomly chosen assignment

- **search steps:**

  1. randomly select a currently unsatisfied clause

  2. select a literal from this clause according to a heuristic h

- if no model found after *maxSteps* steps, restart from randomly chosen assignment

Some WalkSAT algorithms

SKC: Select variable such that minimal number of currently
satisfied clauses become unsatisfied by flipping;
if 'zero-damage' possible, always go for it; otherwise,
with probability wp variable is randomly selected

Tabu: Select variable that maximises increase in total number
of satisfied clauses when flipped; use constant length tabu-list
for flipped variables and random tie-breaking

Novelty: Order variables according to increase in total number of satisfied clauses when flipped; if best variable in this ordering is not most recently flipped, always go for it; otherwise, select second-best with probability wp

R-Rovelty: Similar to Novelty, but more complex decision between the variable with best and second-best score

Some properties of these algorithms
**[Hoos & Stützle 1998-1999; Hoos 1999]:**

- SKC, Tabu, Novelty, and R-Novelty: exponential run-time
  distributions when applied to hard SAT instances and using
  optimal (or greater-than-optimal) noise settings
  $\rightsquigarrow$ optimal parallelisation

- Tabu, Novelty, and R-Novelty: essentially incomplete,
  i.e., can get stuck in non-solution areas of search space

**Novelty$^+$ and R-Novelty$^+$ [Hoos 1998]:**

- extend Novelty, R-Novelty with (unconditional) Random Walk
  — simple generic modification of the corresponding GLSM

- resulting algorithms are probabilistically approximately complete

- significantly improved performance & robustness

These algorithms are amongst the best-performing SLS algorithms for SAT known today. (A self-tuning variant of Novelty$^+$ won the random instance category of the SAT Solver Competition 2004!)

GLSM representations of Novelty vs. Novelty$^+$
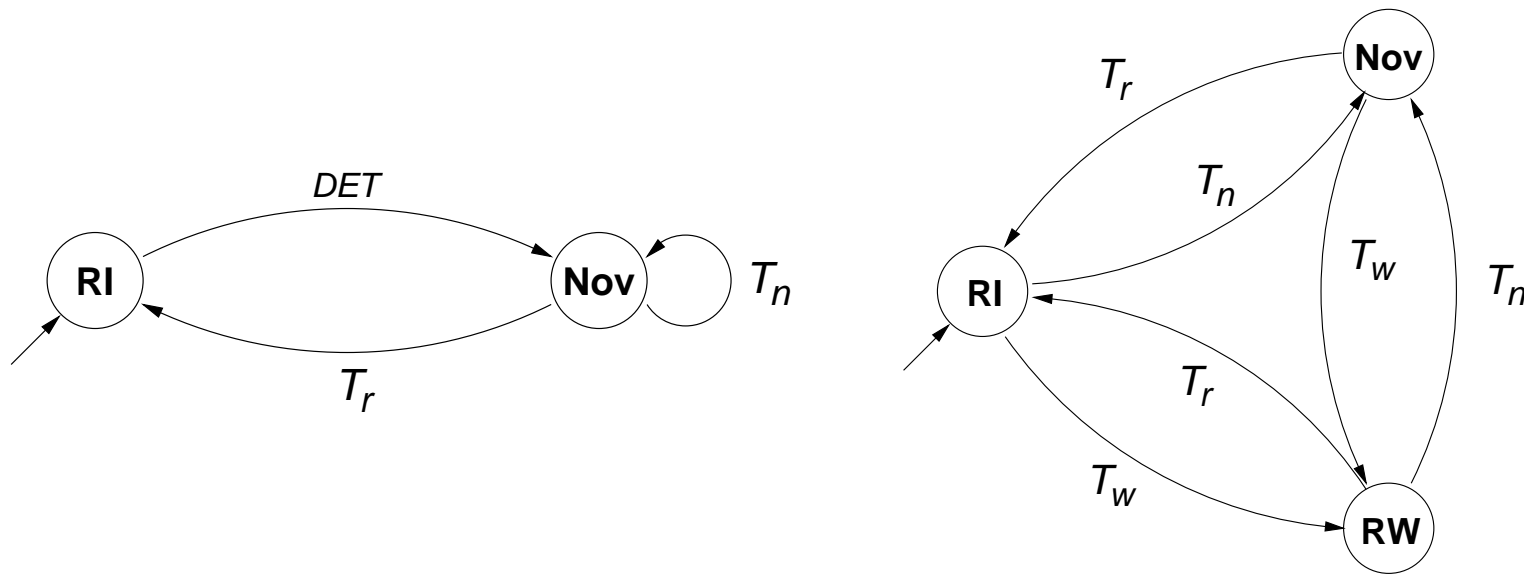
Essential Incompleteness vs. PAC behaviour
of Novelty vs. Novelty^{+}
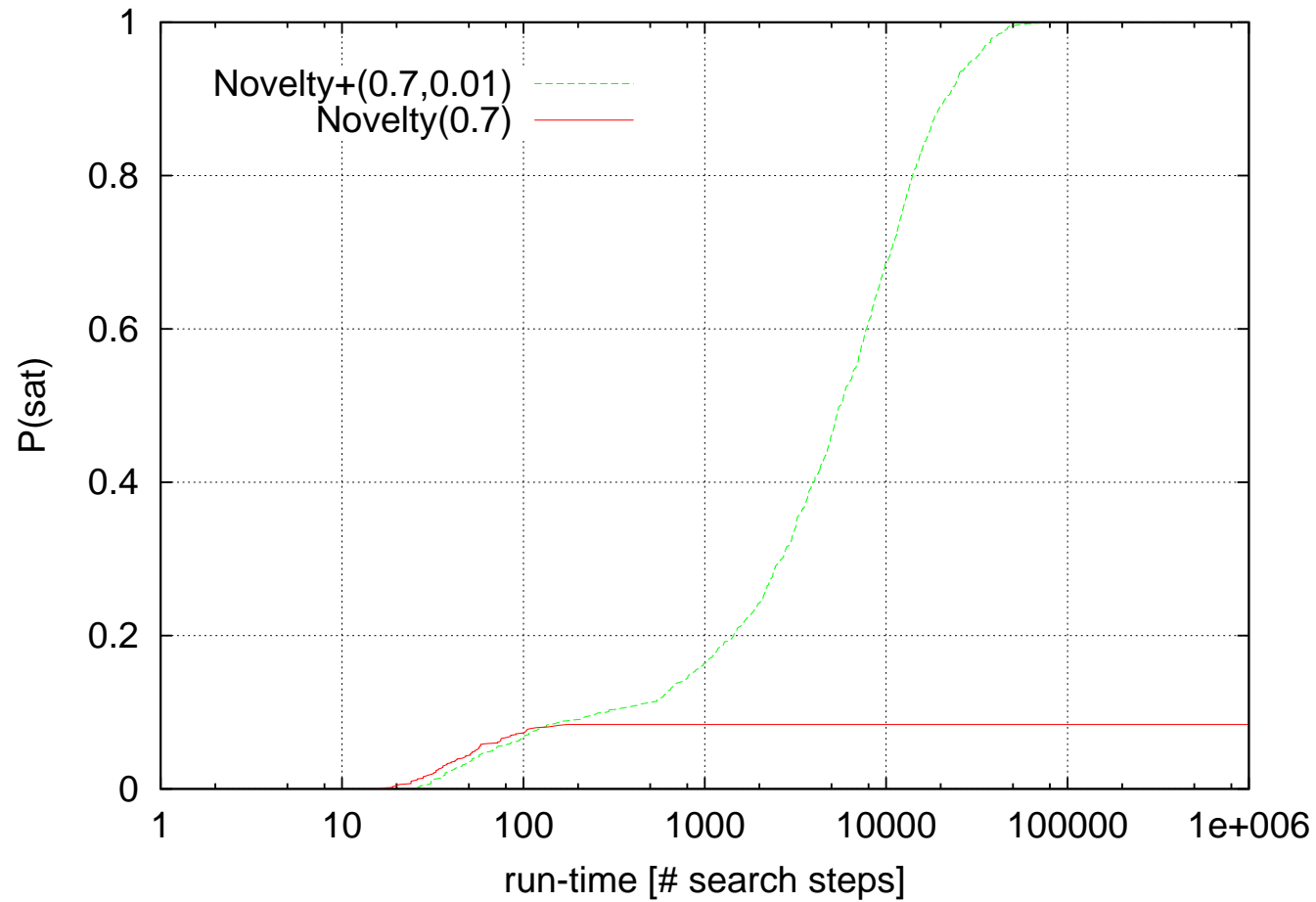
Some other stochastic search approaches for SAT:

- simulated annealing [Spears 1993; Beringer et al. 1994]

- genetic algorithms [Frank 1994]

- GRASP [Feo & Resende 1996]

- simple learning strategies for SLS [Selman & Kautz 1993; Frank 1997; Boyan & Moore 1998]

- Discrete Lagrangian Methods [Wah et al. 1997–2000]

- SDF, ESG [Schuurmans et al. 2000–2001]

- SAPS, RSAPS [Hoos et al. 2002–2003]

- Satz-Rand, REL_SAT-Rand [Gomes et al. 1998]

Maximum Satisfiability (MAX-SAT)

- Generalisation of SAT for CNF formulae

- Prototypical $\mathcal{NP}$-hard combinatorial optimisation problem

- Widely studied in algorithmics, artificial intelligence, computational theory, operations research

- conceptually simple

    $\rightsquigarrow$ facilitates development and analysis of general algorithmic techniques for combinatorial problem solving

- close relationship with SAT facilitates study of differences in the structure of typical decision and optimisation problems

Example: A Simple MAX-SAT Instance

$$F := (\neg x_1)$$
$$\land (\neg x_2 \lor x_1)$$
$$\land (\neg x_1 \lor \neg x_2 \lor \neg x_3)$$
$$\land (x_1 \lor x_2)$$
$$\land (\neg x_4 \lor x_3)$$
$$\land (\neg x_5 \lor x_3)$$

- minimum number of unsatisfied clauses? 1

  $(e.g., x_1 := x_2 := x_3 := x_4 := x_5 := \bot)$

Example: A Simple Weighted MAX-SAT Instance

$$\begin{aligned}
F := \quad & (\neg x_1) & w &= 2 \\
\wedge\ & (\neg x_2 \vee x_1) & w &= 1 \\
\wedge\ & (\neg x_1 \vee \neg x_2 \vee \neg x_3) & w &= 7 \\
\wedge\ & (x_1 \vee x_2) & w &= 3 \\
\wedge\ & (\neg x_4 \vee x_3) & w &= 7 \\
\wedge\ & (\neg x_5 \vee x_3) & w &= 7
\end{aligned}$$

- minimum total weight of unsatisfied clauses? 1

  (*e.g.*, $x_1 := \bot, x_2 := x_3 := x_4 := x_5 := \top$)

Unweighted MAX-SAT:

Given: Propositional formula F in conjunctive normal form (CNF)

Goal: Find truth assignment a^* that minimises the number of
unsatisfied clauses in F, *i.e.*,

$$a^* \in \text{argmin}_{a \in \text{Assign}(F)} \ \#CU(F, a),$$

where $CU(F, a) :=$ set of unsatisfied clauses in F under a.

Weighted MAX-SAT:

Given: Propositional formula F in conjunctive normal form (CNF),
weights $w(c)$ for each clause c in F

Goal: Find truth assignment a^* that minimises the total weight
of unsatisfied clauses in F, *i.e.*,

$$a^* \in \text{argmin}_{a \in \text{Assign}(F)} \sum_{c \in CU(F,a)} w(c),$$

where $CU(F, a) :=$ set of unsatisfied clauses in F under a.

Note: instances of other combinatorial problems can be
encoded into and solved as MAX-SAT, *e.g.*:

- (Weighted) Set Covering

- Most Probable Explanation Finding in Bayes Nets

- Spin Glass Ground State Determination

State-of-the-art SLS algorithms for MAX-SAT fall into
three categories:

- WalkSAT algorithms (Novelty$^+$/*wcs+we*)

- Iterated local search algorithms (IRoTS)

- Dynamic local search algorithms (GLSSAT, SAPS)

SLS algorithms for MAX-SAT also play a major role
as components in high-performance branch & bound algorithms
for MAX-SAT [Alsinet *et al.*, 2003]

Weighted WalkSAT / Novelty$^+$ [Hoos, Smyth, Stützle, 2003]:

weighted evaluation (we): Evaluate variable flips using
$$\sum_{c \in CU} w(c) \quad \text{instead of} \quad \sum_{c \in CU} 1 = \#CU$$

weighted clause selection (wcs): Select unsatisfied clauses
with probability proportional to their weight
(roulette wheel selection)

$\rightsquigarrow$ Three weighted variants of each WalkSAT algorithms:
wcs, we, wcs+we

Performance of Novelty$^+$/*wcs+we* on Structured Instances

Test-set	GLSSAT2			Novelty$^+$/*wcs+we*		
	$t = 0.1$s	$t = 1$s	$t = 10$s	$t = 0.1$s	$t = 1$s	$t = 10$s
scp4	1.33	0.02	0.01	**0.02**	**0.01**	**0.00**
gcp-yi	11.24	0.22	0.15	**0.12**	**0.05**	**0.01**
lgcmp75	$8.1 \cdot 10^6$	49.3	0.20	**3.27**	**0.09**	**0.00**
lgcmp100	$1.1 \cdot 10^6$	248.82	**0.25**	**10.02**	**1.39**	0.51

Performance measure = median relative solution quality for fixed cut-off times

Novelty$^+$/*wcs+we vs.* GLSSAT: Solution Quality Over Time
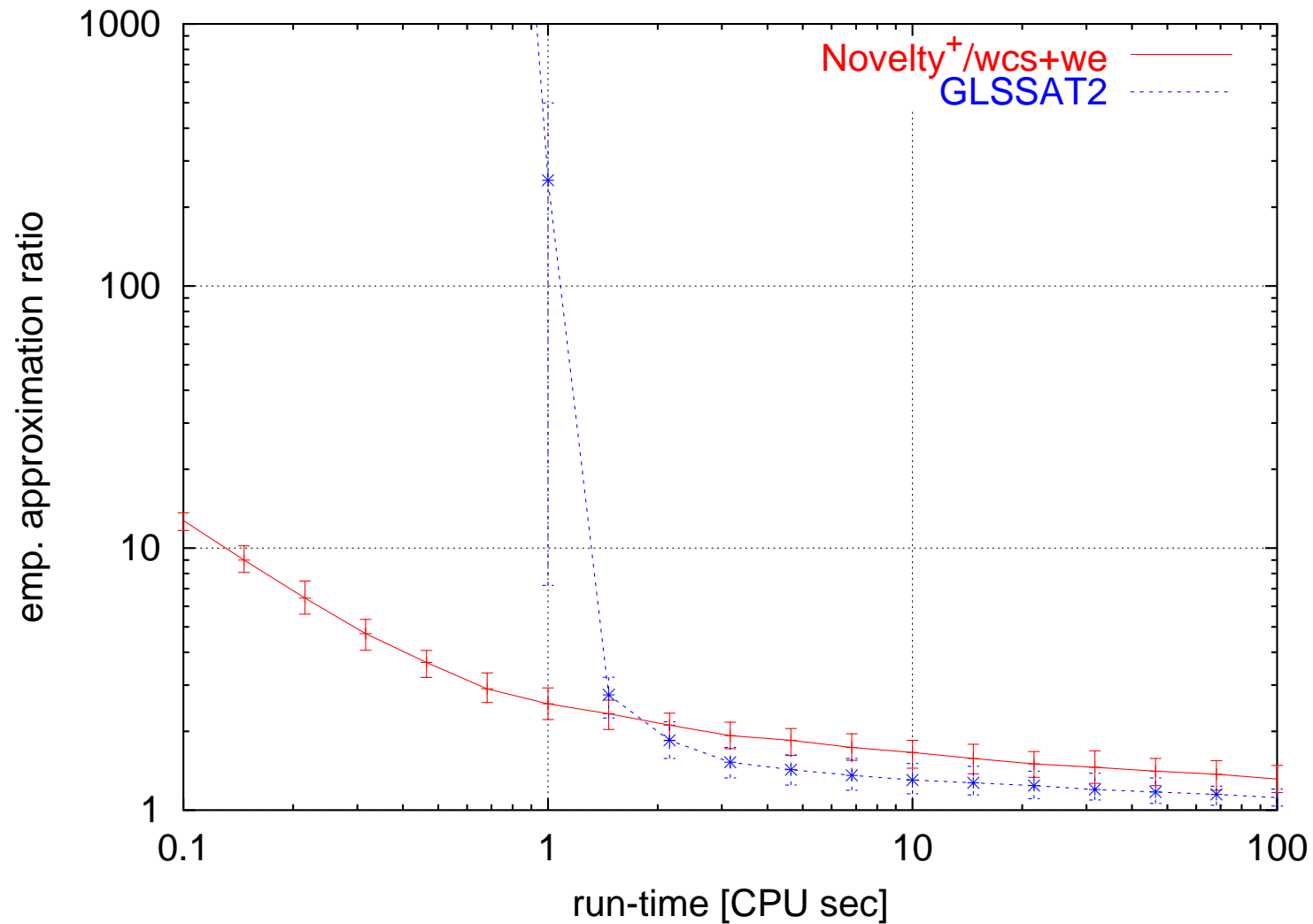
Note:

- Different algorithms define state-of-the-art for various types of MAX-SAT instances

- Some types of MAX-SAT instances are solved very well by straightforward generalisations of high-performance, others require different SLS methods.

Scheduling

Scheduling: allocation of limited resources to tasks over time

Resources: examples are machines in a workshop, runways at airport, . . .

Tasks: examples are operations in a production process, take-offs and landings at an airport, . . .

Objectives: usually depending on the tasks completion time, shipping dates to be met, . . .

Airport example

Resources: runways available for starts and landings

Tasks: a number of *flights* with

- processing times

- time windows for allowable take-off and landing times

- minimium time distance between successive take-offs and landings

- weight indicating importance

Objectives: minimize the total weighted delay over all flights

The Single Machine Total Weighted Tardiness Problem

Often, a single machine is the bottleneck in production environment.

Given (for each job):

- due date d_j

- weight (importance) w_j

- processing time p_j

Objective: minimise sum of weighted tardiness

(tardiness = surtime a job is completed after its due date)

The SMTWTP is $\mathcal{NP}$-hard.

Local search algorithms for SMTWTP

- **initialisation:** several construction heuristics available

- **search space:** all possible task sequences

- **neighbourhoods:** transpose, exchange, insert

transpose neighbourhood

exchange neighbourhood

insert neighbourhood
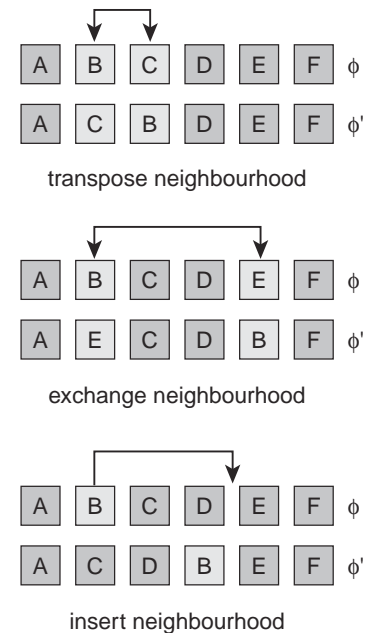
Variable Neighbourhood Search

Observation: local optimum w.r.t. one neighbourhood need not be
a local optimum for another neighbourhood relation

Idea: systematically change neighbourhood during local search
process [Mladenovic & Hansen 1995–2003]

Tests: effectiveness of concatenating interchange and insert
neighbourhood with different construction heuristics

Instances: 125 randomly generated benchmark instances with 100
jobs; results are averaged over the benchmark instances

constr.	no local search			interchange			insert		
heuristic	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}
EDD	135	24	0.001	0.62	33	0.140	1.19	38	0.64
MDD	61	24	0.002	0.65	38	0.078	1.31	36	0.77
AU	21	21	0.008	0.92	28	0.040	0.56	42	0.26

constr.	interchange+insert			insert+interchange		
heuristic	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}
EDD	0.24	46	0.20	0.47	48	0.67
MDD	0.40	46	0.14	0.44	42	0.79
AU	0.59	46	0.10	0.21	49	0.27

Δ_{avg}: avg. deviation from best known, n_{opt}: # of best known candidate solutions found t_{avg}: avg. computation in seconds on Pentium II 266MHz

Results:

- concatenated local search significantly more effective

- final solution quality depends strongly on
  initial candidate solution

- other local search extension:
  Dynasearch [Congram, Potts, van de Velde 2002]

- further improvements can be achieved by using
  hybrid SLS methods, e.g. ILS algorithms

ILS for SMTWTP

- *local search:* several possibilities examined, best results with interchange+insert

- *perturbation:* variable number of random left-insert moves

- *acceptance criterion:* Apply candidate solution modification to best candidate solution since start of the algorithm

Summary results for ILS

- finds best-known candidate solutions on all known benchmark
  instances in reasonable time (few CPU seconds up to some
  minutes)

- reasons for good performance were identified by
  search space analysis (FDC)

- most other high-performance algorithms for SMTWTP are
  ILS-based (Iterated Dynasearch)

Timetabling

Timetabling concerns the placement of a set of events to time slots and resources.

- highly application-relevant research area

- many research efforts (PATAT conference series, . . .)

- wide class of resource allocation problems

  - employee timetabling, academic timetabling, transportation timetables (railways, buses), . . .

Academic timetabling

- classes of problems

  - school timetabling

  - course timetabling

  - examination timetabling

- wide variety of models and constraints

- SLS based approaches are becoming
  increasingly important

Example: A course timetabling problem

- problem

    - given is a set of events visited by a set of students

    - *goal:* assign events to time-slots and rooms subject to *hard* constraints and optimisation criteria

- *hard* constraints

    - no student attends more than one event at the same time

    - room is big enough and satisfies all features required by the event

   - at any time-slot, there is at most one event in a room

- optimisation criteria defined by *soft* constraints

  – student should not have an event in last slot of a day

  – student should not have more than two events in a row

  – student should have only a single class per day

- soft constraint violations are penalised

- *optimisation objective:*

  – find a feasible solution with minimum number of soft constraint violations

- models real-world problem at Napier University, Edinburgh

- was tackled in the Metaheuristics Network

- was part of the *International Timetabling Competition*

- various SLS methods tested on this problem

  – Ant Colony Optimisation

  – Iterated Local Search

  – Simulated Annealing

  – Tabu Search

  – Evolutionary Algorithms

- all SLS methods were implemented by the expert labs and extensively tested on a set of benchmark instances

- implementations were done in two phases

- first phase

  - all labs were given the same local search procedure

  - all labs were given one month of development time

  - then all algorithms had to be submitted and were evaluated

- experimental tests run across various instance sizes

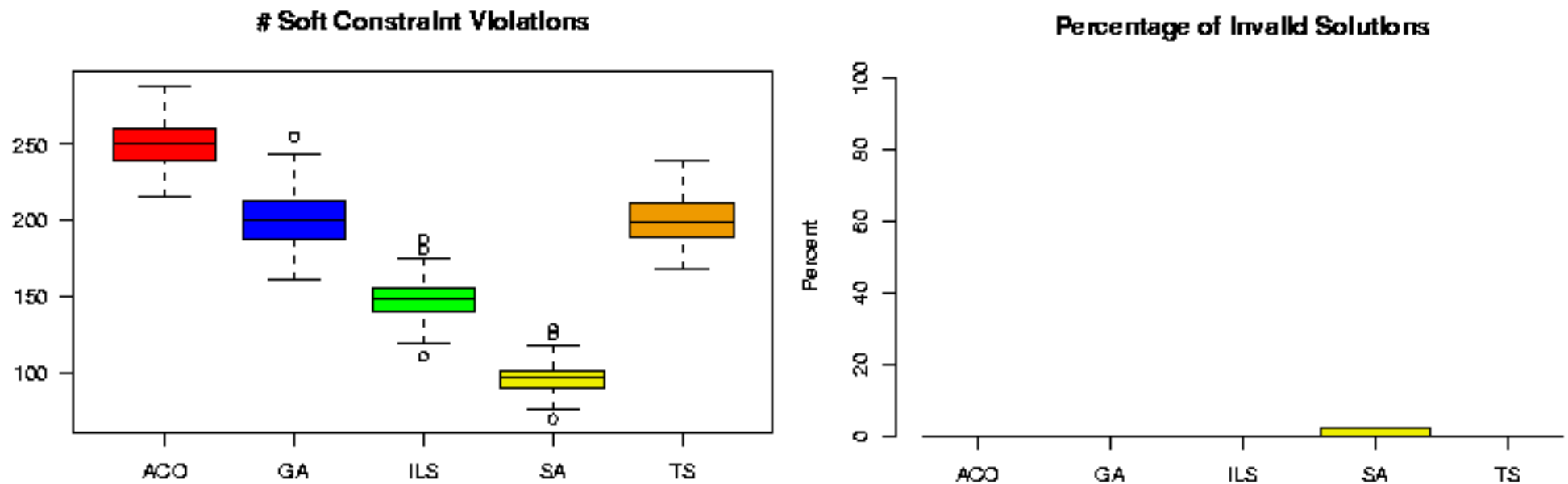- the computational results were extensively analysed with non-parametric statistical tests

small size instance

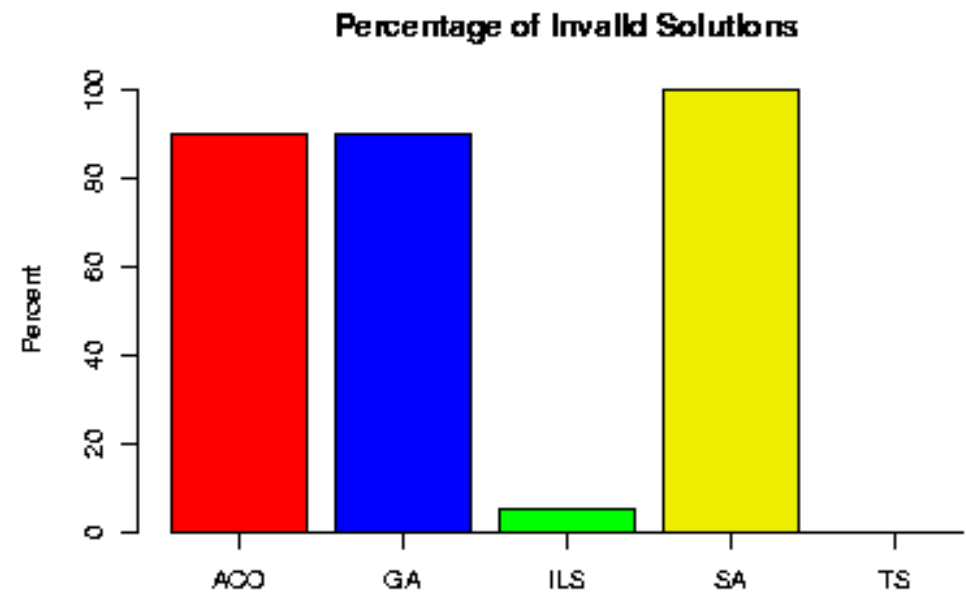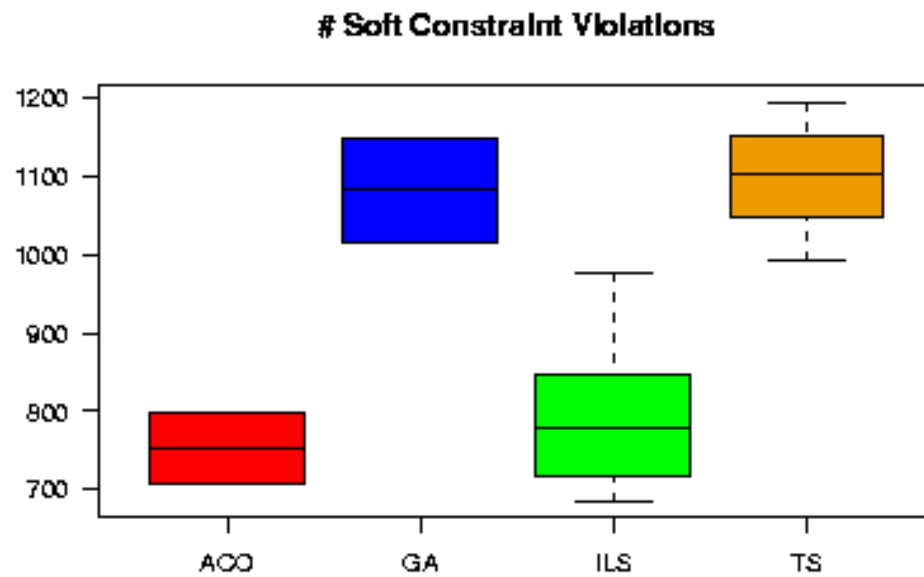*100 events, 80 students, 5 rooms, 5 features; 500 trials run per instance*
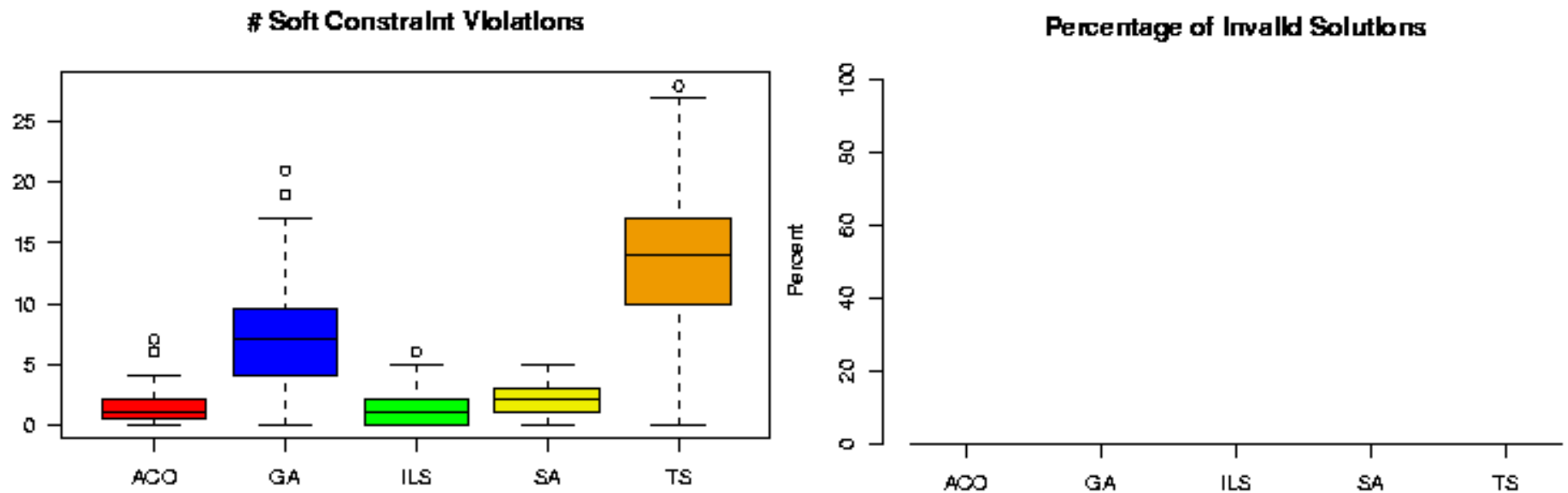
medium size instance

400 events, 200 students, 10 rooms, 5 features; 50 trials run per instance

large size instance

400 events, 400 students, 10 rooms, 10 features; 20 trials run per instance

- main observations

  - SA very good for optimising soft constraint violations but poor for hard constraints

  - tabu search based approaches good for solving hard constraints but poor for soft constraints

- new SLS algorithm developed based on results of first phase

  - various SLS techniques (TS, SA, VNS) in dependence of problem solving stage (hard, soft constraints, etc.)

  - configuration of hybrid technique using semi-automatic statistical procedures

- this new SLS algorithm was the (unofficial) winner of the International Timetabling Competition!

Conclusions and Future Research

Conclusions

- Stochastic search is one of the most efficient approaches for solving combinatorial problems in practice

- Studying stochastic search algorithms for conceptually simple domains, such as SAT or TSP, facilitates development, analysis, and understanding of algorithms

- Advanced empirical methodology helps to characterise and exploit stochastic search behaviour based on computational experiments

- Lots of potential application areas, lots of interesting research questions

Future Work

- further advance understanding of stochastic search behaviour
  ($\rightsquigarrow$ search space analysis, new theoretical & empirical results)

- further improvements in stochastic search algorithms
  ($\rightsquigarrow$ hybrid algorithms, adaptive algorithms)

- application to real-world problems
  (e.g., intelligent systems, e-commerce, bioinformatics;
  dynamic, stochastic, multi-objective problems)

Take-Home Message:

- Stochastic search is one of the most successful
  and most widely used methods for solving
  combinatorial problems (but certainly no panacea)

- Many open problems, significant research
  and practical potential

- SLS research is fun!

Want to learn more about Stochastic Local Search?

- **New book:**
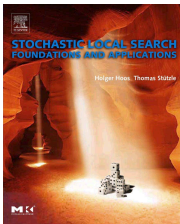
  Holger H. Hoos & Thomas Stützle:

  Stochastic Local Search: Foundations and Applications

  Morgan Kaufmann Publishers, Summer 2004.

- **New website:**

  www.stochastic-local-search.net