Stochastic Search Algorithms

Holger H. Hoos

Computer Science

U of British Columbia

Canada

Thomas Stützle

FB Informatik

TU Darmstadt

Germany

Motivation: Why Stochastic Search?

Stochastic 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 search and how can it be used to solve computationally hard problems?
- Which stochastic search techniques are available and what are their features?
- How should stochastic search algorithms be studied and analysed empirically?
- How are specific problems solved using stochastic search?

Outline

Introduction

Part I: Combinatorial Problems and Search

- Combinatorial problems
- Complexity issues
- Search methods
- Stochastic search

[Short Break]

Part II: Stochastic Local Search Methods

– Simulated Annealing

– Tabu Search

– Iterated Local Search

– Evolutionary Algorithms

– Ant Colony Optimization

[Short Break]

Part III: Stochastic Search Behaviour

- Empirical evaluation
- Search space characteristics
- Parameterisation and tuning

[Short Break]

Part IV: Applications

- SAT
- Traveling Salesman Problem
- Quadratic Assignment Problem
- Scheduling
- Planning
- Combinatorial Auctions

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
- sequencing genomes

Combinatorial problems ...

- typically 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):

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

$\leadsto$ **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
  $\leadsto$ facilitates development and evaluation of algorithmic ideas

Combinatorial Optimisation Problems:

Objective function (evaluation 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 evaluation function.

The Traveling Salesperson Problem (TSP)

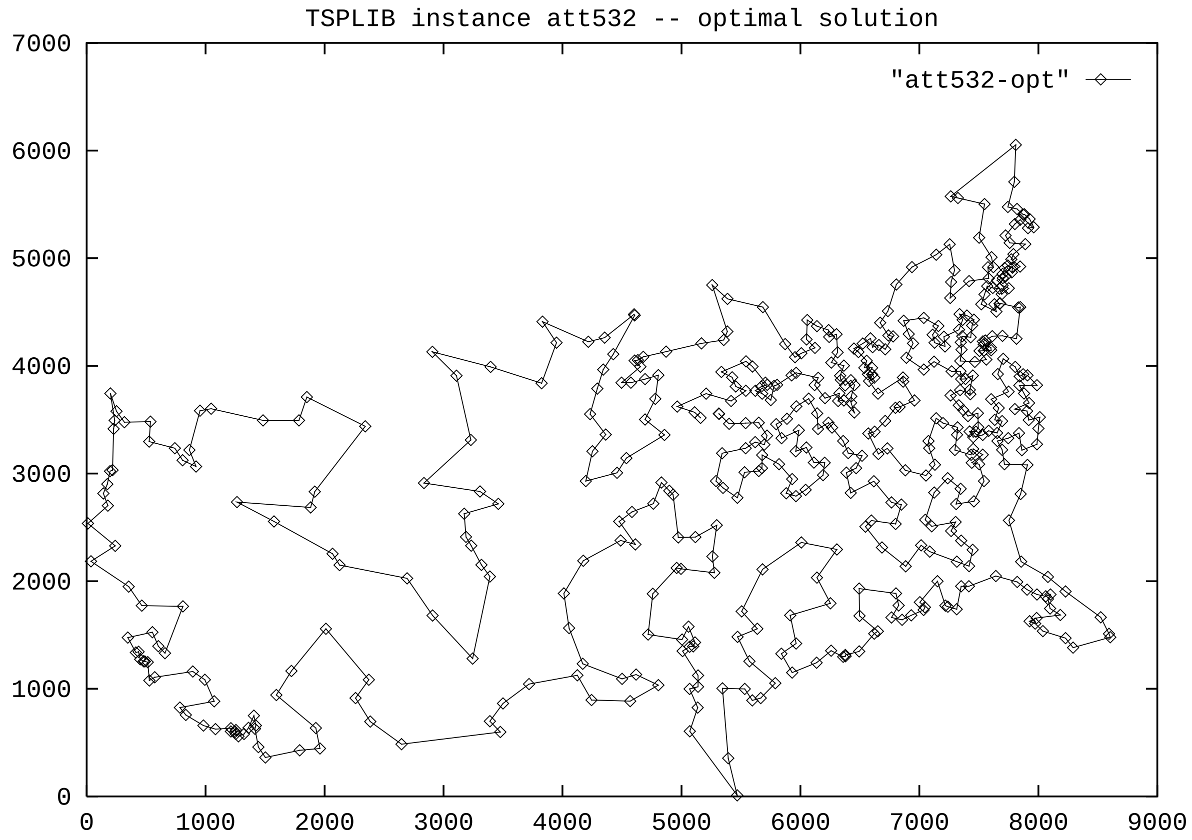
TSP – optimisation variant:

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.

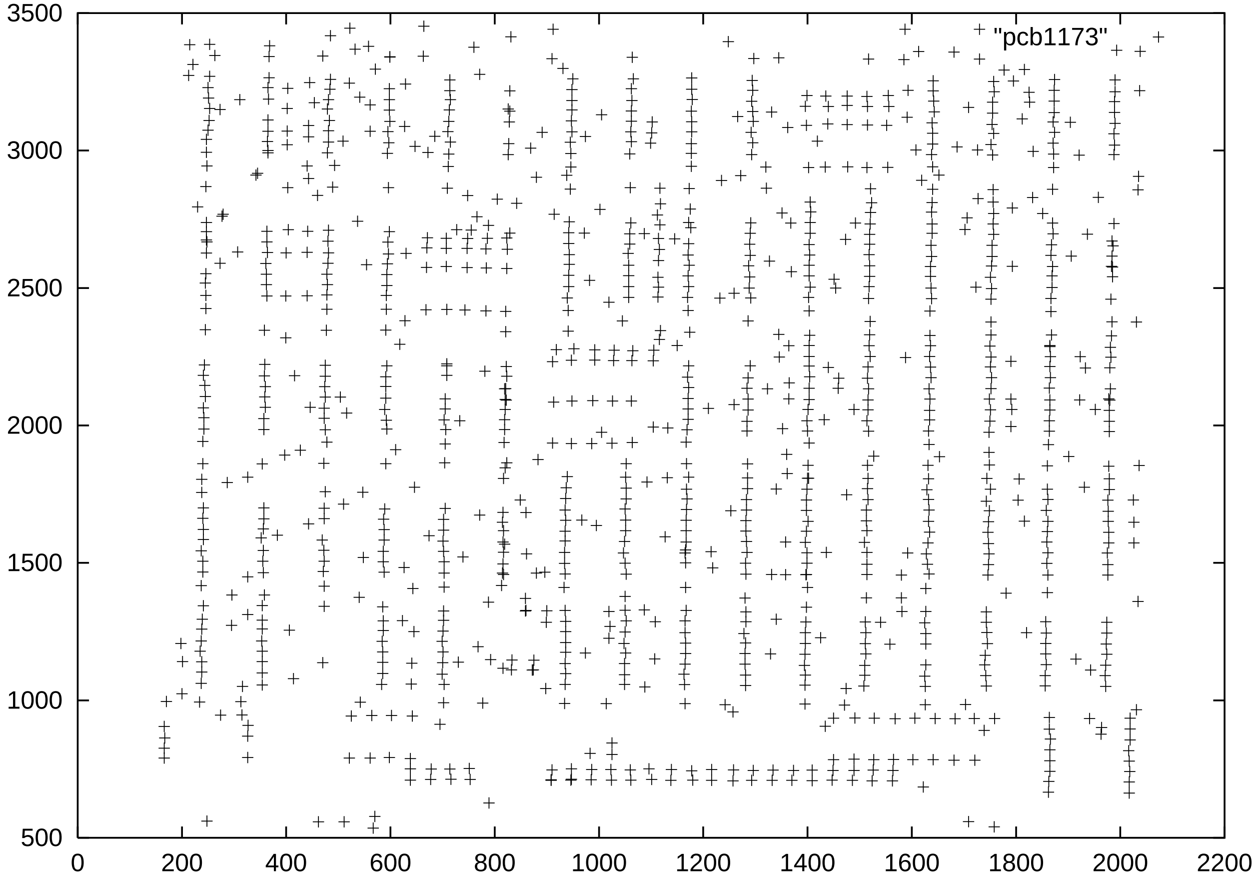
TSP – decision variant:

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

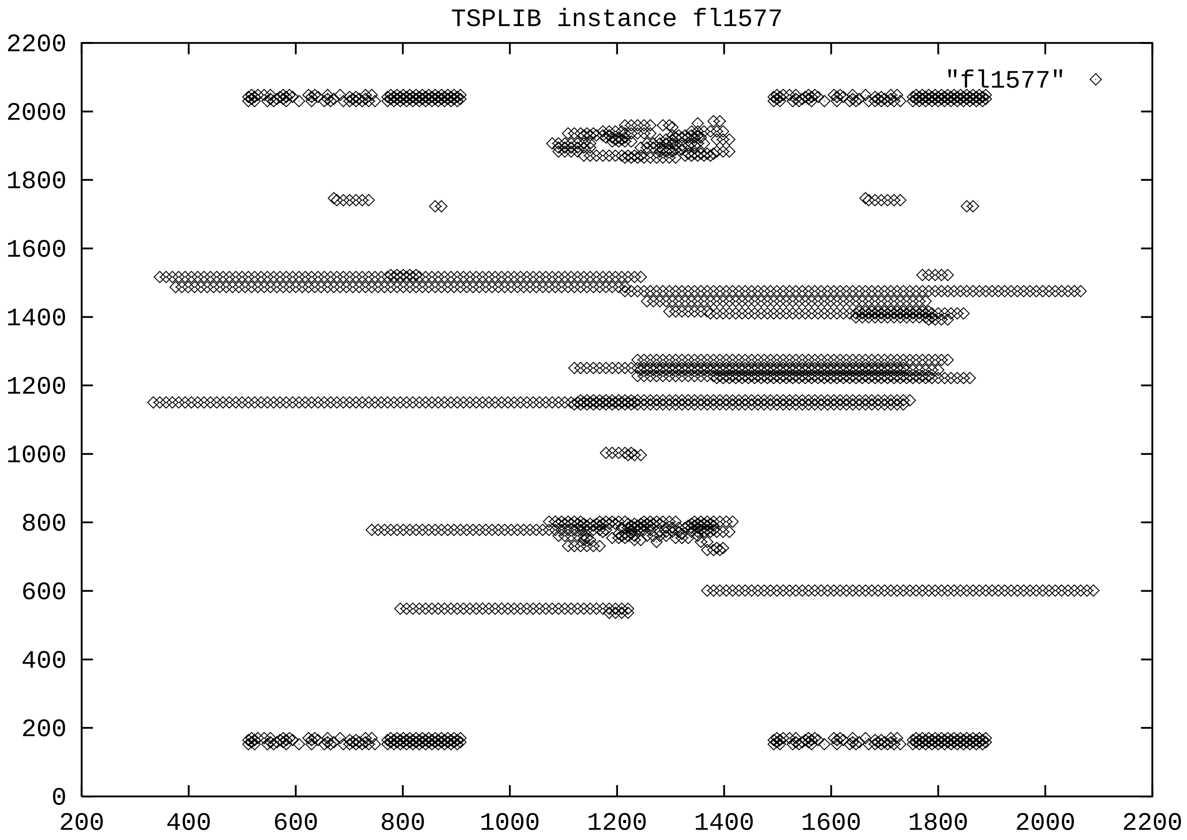
TSP instance: shortest round trip through 532 US cities



TSP instance: minimal drill path for printed circuit board



TSP instance: minimal drill path for printed circuit board



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
  $\leadsto$ 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 (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 $1.5\cdot$ optimal solution
- 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:

systematic	$\longleftrightarrow$	local search
deterministic	$\longleftrightarrow$	stochastic
sequential	$\longleftrightarrow$	parallel

Properties of search methods:

Decision problems: *complete* vs. *incomplete*

Optimisation problems: *exact* vs. *approximate*

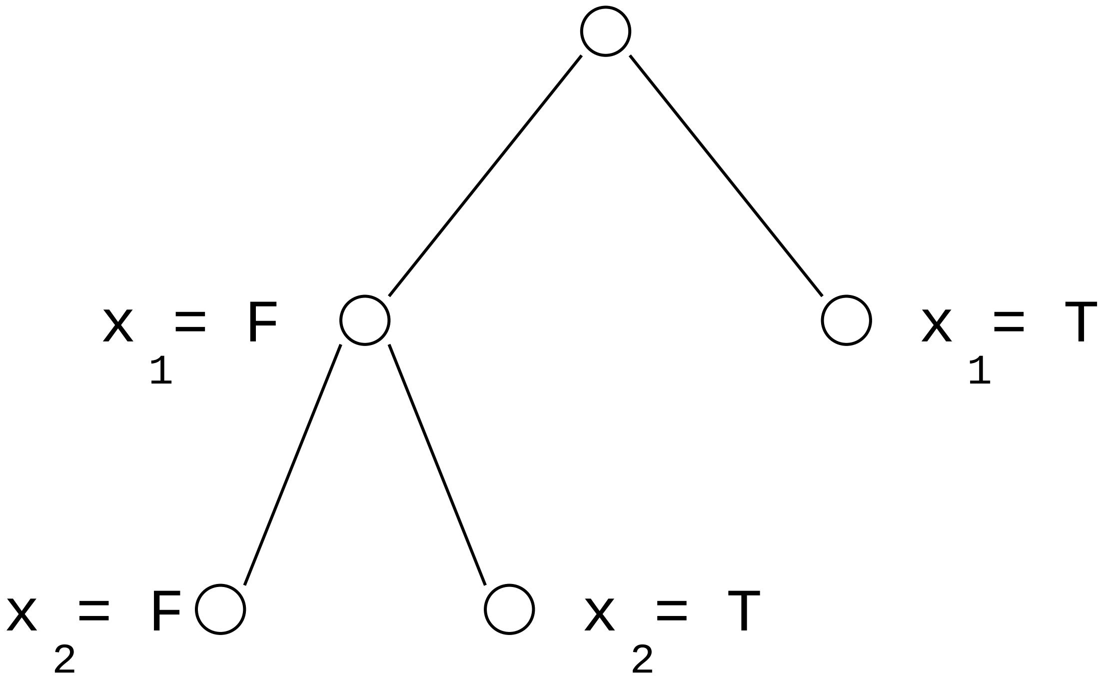
‘Classical’ search algorithms based on back-tracking:

- iteratively construct candidate solution
- use back-tracking to explore full search space
- use appropriate techniques for pruning search space

The ‘Davis-Putnam’ algorithm for SAT

- iteratively select variables and search both truth assignments for these
- backtracking guarantees completeness of search
- use unit propagation to prune search space
- ordering of variables and values is critical for performance

Binary search tree for Davis-Putnam-style algorithm



Branch-and-bound methods for the TSP

- extension of backtracking algorithms which additionally considers solution cost
- branching partitions the set of solutions represented at current node in exclusive sets
- lower bound estimates completion cost of partial solutions
- upper bound estimates the optimal solution from above
- partial solutions are discarded if lower bound is larger than upper bound
- analogous to A^* search in AI

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)

objective function $f : S \mapsto \mathbb{R}_0^+$

(SAT: number of clauses unsatisfied under given assignment)

Local Search:

- start from initial position
- iteratively move from current position to neighbouring position
- uses objective function for guidance

Two main classes

- local search on partial solutions
- local search on complete solutions

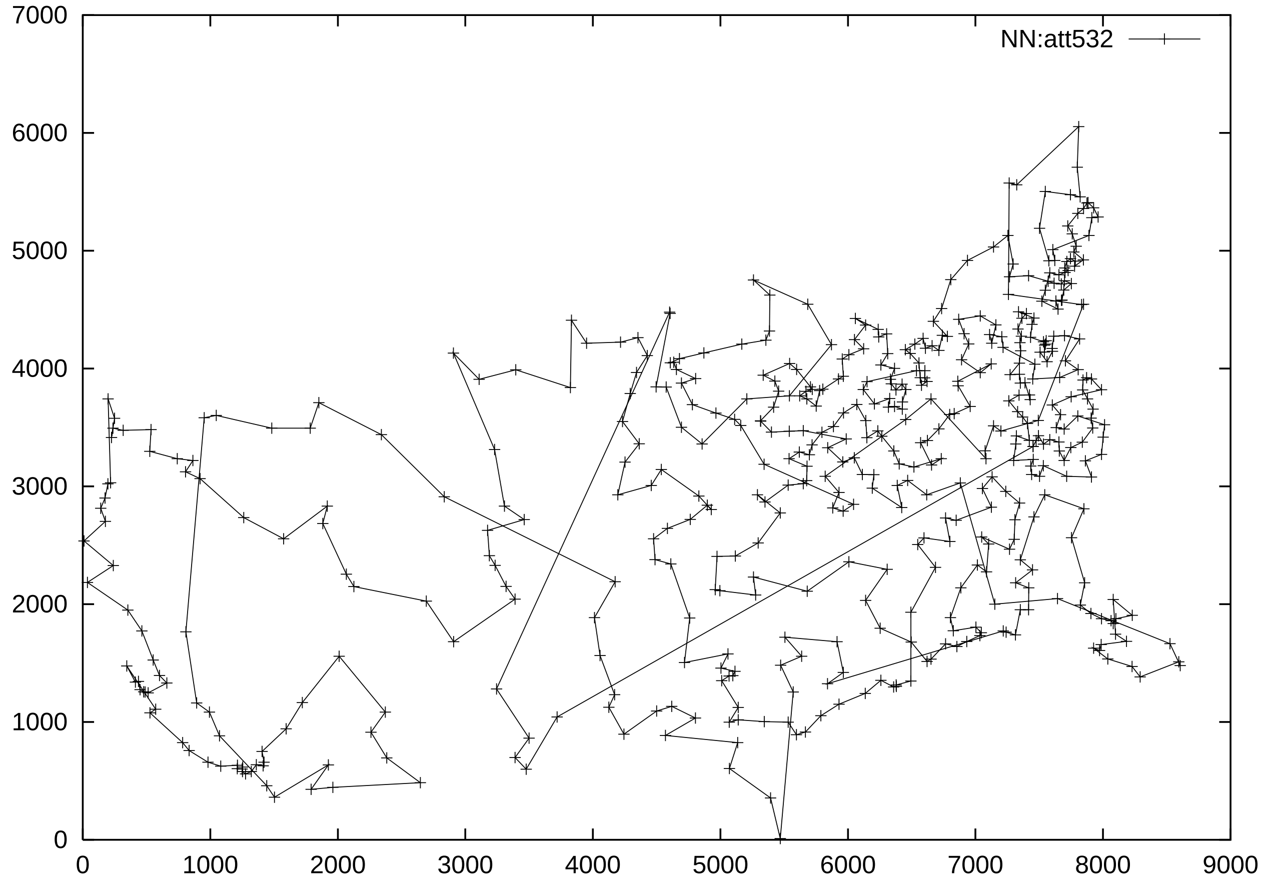
Construction Heuristics:

- 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 objective function

Nearest Neighbour heuristic for the TSP:

- always choose at the current city the closest 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



Example of a (greedy) construction heuristic for SAT

- start with an empty variable assignment
- in each step select an unassigned variable
  and set it to a truth value
  - if $\exists$ unsatisfied clause with only one unassigned variable, assign this variable to satisfy this clause
  - otherwise choose variable and truth value such that maximal number of clauses become satisfied

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)

Local Search on complete solutions

- widely used class of LS algorithms
- search space: the space of feasible solutions
  (SAT: complete truth assignments, TSP: valid round trips)
- initial position often randomly chosen
- search steps: move from current solution to neighbouring, feasible solutions

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 objective function value

procedure BestImprovementStep($s \in \mathcal{S}$)

$w = \emptyset$, $s^* = s$

while $(\mathcal{N}(s)/w \neq \emptyset)$ **do**

$s' =$ GenerateSolution(s)

$w = w \cup s'$

if $(f(s') < f(s^*))$

$s^* = s'$

end

return s^*

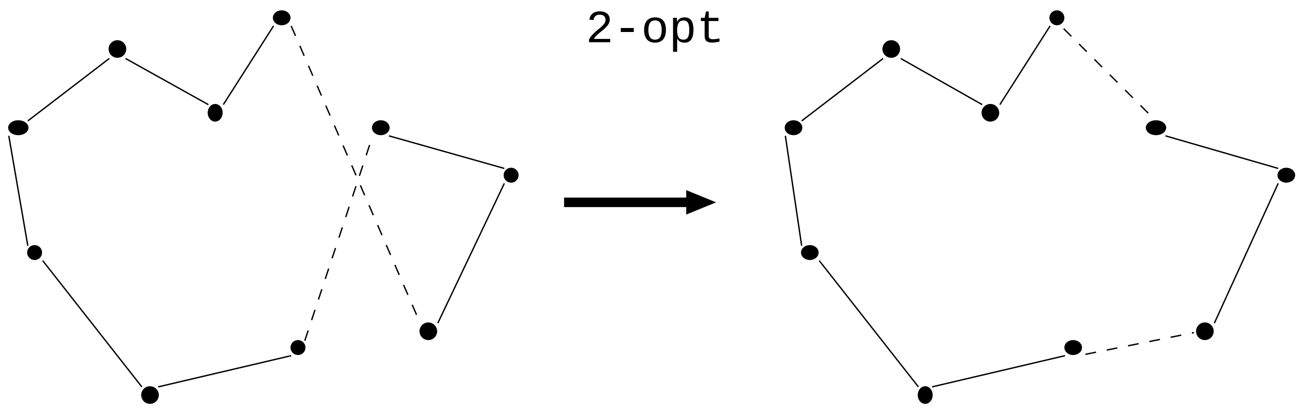
end BestImprovement

Iterative Improvement for SAT

- initialisation: randomly chosen, complete truth assignment
- neighbourhood: variable assignments are neighbours iff they differ in truth value of one variable
- objective function: number of clauses unsatisfied under given assignment

Iterative Improvement for the TSP

- initial solution is a complete tour
- k-opt neighbourhood: solutions which differ by at most k edges

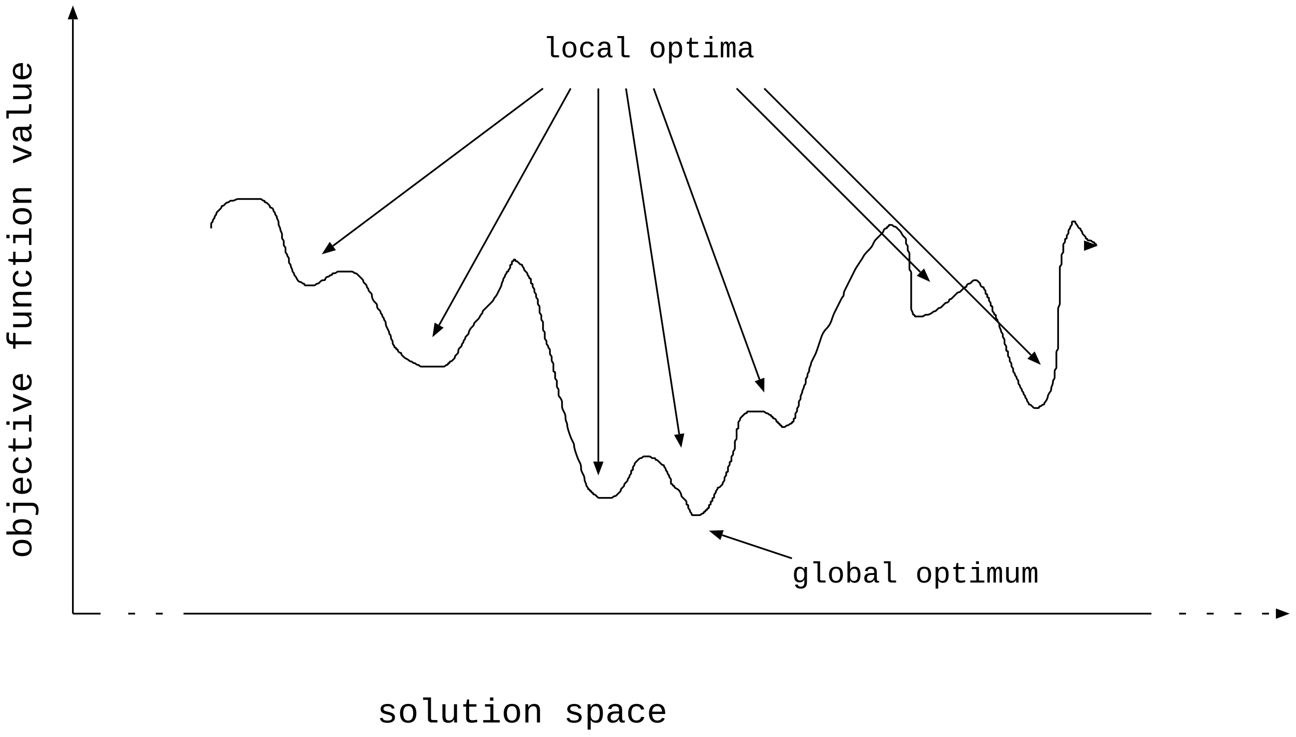


- neighbourhood size $\mathcal{O}(n^k)$
- More complex neighbourhoods: variable depth search

Stochastic Local Search

Typical problems with local search:

- getting stuck in local optima
- being misguided by objective function



local optima
objective function value
global optimum
solution space

Stochastic Local Search:

- randomise initialisation step
  - random initial solutions
  - randomised construction heuristics
- randomise search steps
  such that suboptimal/worsening steps are allowed
  $\leadsto$ improved performance & robustness
- typically, degree of randomisation controlled by noise parameter
- allows to invest arbitrary computation times

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 objective function value

Randomised Systematic Search

Typical problems with systematic search:

- abysmal performance for certain instances
- being misguided by search heuristic

Randomised Systematic Search:

- randomise heuristic choices using the concept of *heuristic equivalence* $\leadsto$ improved robustness
- degree of randomisation controlled by noise parameter
- additionally, use random restart

Randomised Davis-Putnam algorithm for SAT

- randomise selection of variable to be instantiated next and/or order of instantiations (with truth values)

- algorithm remains complete

$\rightsquigarrow$ increased robustness and performance when combined with random restart strategy and using tuned parameter settings (e.g., satz-rand [Gomes et al.])

Pros:

- increased robustness, in particular when using suitably tuned noise and restart strategies
- simple, generic extension of systematic search
- resulting algorithms typically still complete
- potential for easy parallelisation

Cons:

- highly stochastic behaviour
- difficult to analyse theoretically / empirically
- parameter tuning often difficult, but critical for obtaining good performance

Part II

Stochastic Local Search Methods

Overview:

- Parameterised local search extensions
  - Simulated Annealing
  - Tabu Search

- Hybrid strategies
  - Iterated Local Search
  - Evolutionary Algorithms
  - Ant Colony Optimization

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

These SLS algorithms are general frameworks for the design of heuristic algorithms and are often called *metaheuristics*.

Metaheuristic: general-purpose heuristic method designed to guide underlying problem specific heuristic (e.g., local search algorithm or construction heuristic) towards promising regions of the search space.

Simulated Annealing

Combinatorial search technique inspired by the
physical process of annealing [Kirkpatrick et al. 1983, Cerny 1985]

- physical annealing process:
  1. heat up a solid until it melts
  2. slowly decrease the temperature to reach a ground state
- algorithmic analogy:

$$\begin{array}{rcl} \text{states} & \longleftrightarrow & \text{solutions} \\ \text{energy} & \longleftrightarrow & \text{objective function} \end{array}$$

Simulated Annealing – general 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, neighboured solution

Acceptance criterion

- Metropolis acceptance criterion
  - better solutions are always accepted
  - worse solutions are accepted with probability

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

Annealing

- parameter T, called temperature, is slowly decreased

Generic choices for annealing schedule

- initial temperature T_0
  (example: based on statistics of objective function)
- temperature function — how to change temperature
  (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 temperatures)

Example application to the TSP [Johnson & McGeoch, 1997]

- baseline implementation
  - start with random initial solution
  - uses 2-opt neighborhood
  - straightforward annealing schedule

  $\leadsto$ relatively poor performance

- improvements
  - look-up table for acceptance probabilities
  - neighbourhood pruning
  - low-temperature starts

Discussion

- SA is historically important
- easy to implement
- convergence proofs: theoretically interesting, but practical relevance very limited
- 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 solution attributes of recently seen solutions
- tabu solution attributes are often defined via local search moves
- a 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 overwrite tabu status

- stopping criteria:
  - all neighbored solutions are tabu
  - maximum number of iterations
  - 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 overwritten

Tabu search — use of long term memory

Longer 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

Discussion

- short term memory strategies alone often perform astonishingly well
- additional intensification and diversification strategies can considerably improve performance
- several additional strategies available (e.g., strategic oscillation, path relinking, ejection chains, . . .)
- rather complex to use
- very good performance but often at the cost of 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, …*

$\leadsto$ 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

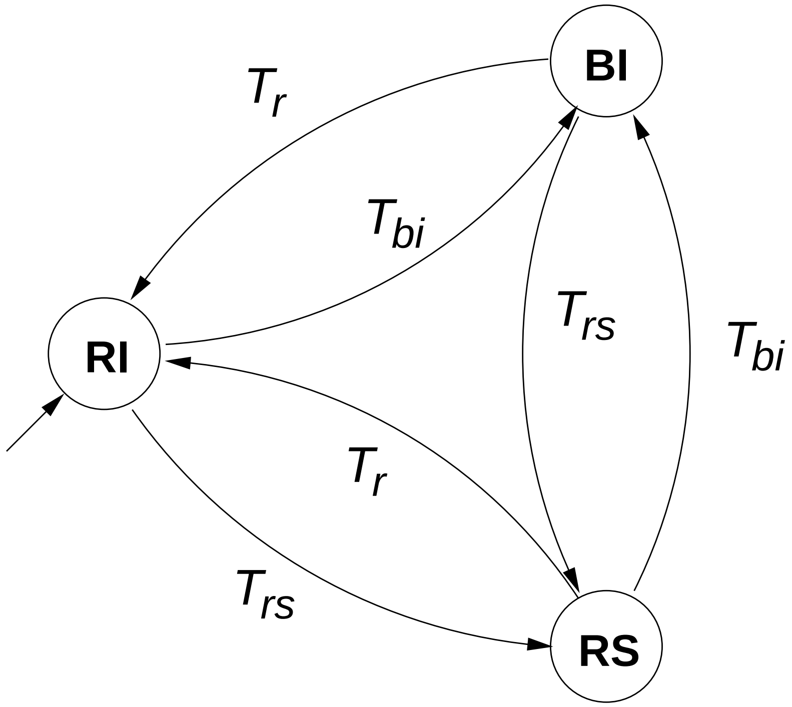
State transition types:

- deterministic: DET
- 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

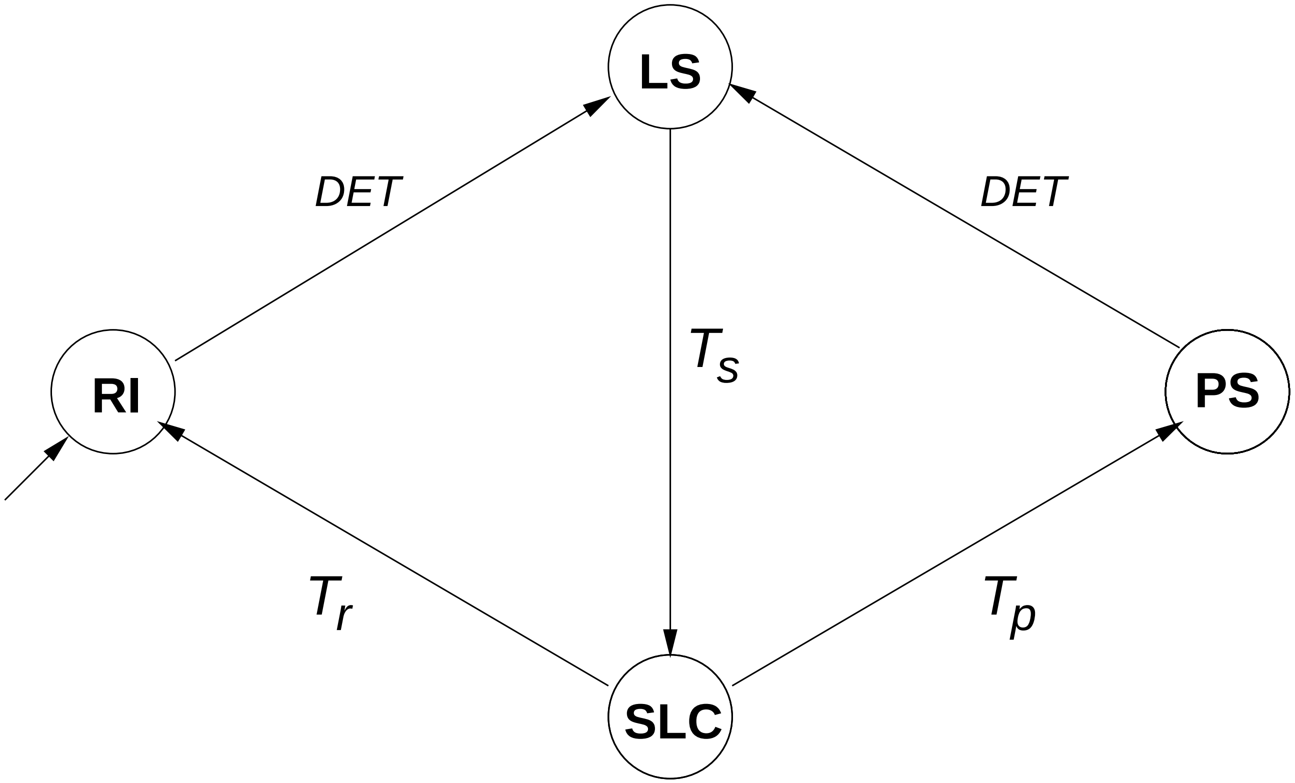


Iterated Local Search (ILS)

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



Why is ILS a good idea?

- speed – large number of local search applications in given time
- bias towards good solutions
- very easy to implement
- few parameters

Issues for Iterated Local Search applications

- choice of initial solution
- choice of solution modification
  - — Too strong: close to random restart
  - — Too weak: not enough to escape 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
- 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

- ILS is among the best available algorithms for the TSP

Other applications:

- Graph partitioning
- Quadratic Assignment Problem
- Scheduling problems

Related idea:

- Variable Neighbourhood Search

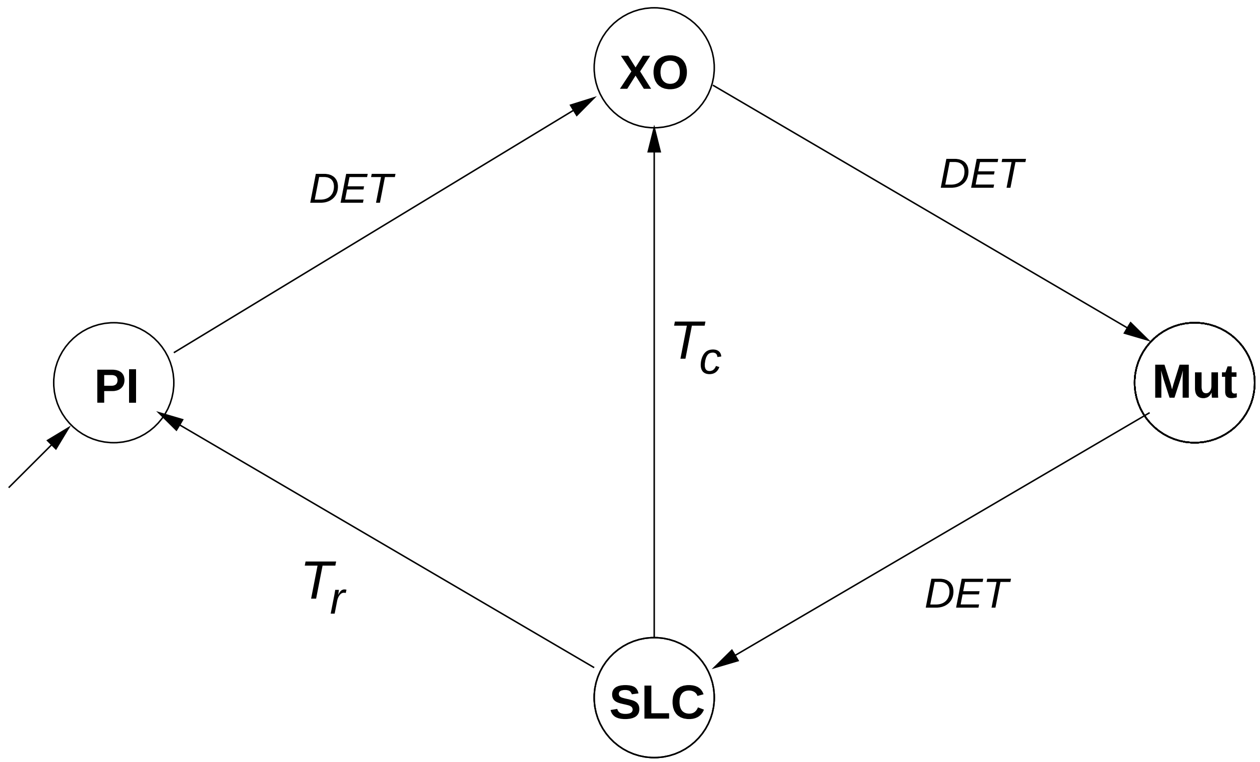
Evolutionary Algorithms

Combinatorial search technique inspired by the evolution of species.

- population of strings which are manipulated via evolutionary operators and compete for survival
- population is manipulated via *evolutionary operators*
  - mutation
  - crossover
  - selection

- several variants have been developed
  - Genetic algorithms [Holland, 1975, Goldberg 1989]
  - Evolution strategies [Rechenberg, 1973, Schwefel, 1981]
  - Evolutionary Programming [Fogel, Owens, Walsh, 1966]
  - Genetic Programming [Koza, 1992]
- for combinatorial optimization the most widely used and most effective variant are genetic algorithms

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
  - how can meaningful information be given to the offspring
- mutation operator
  - background operator vs. driving the search

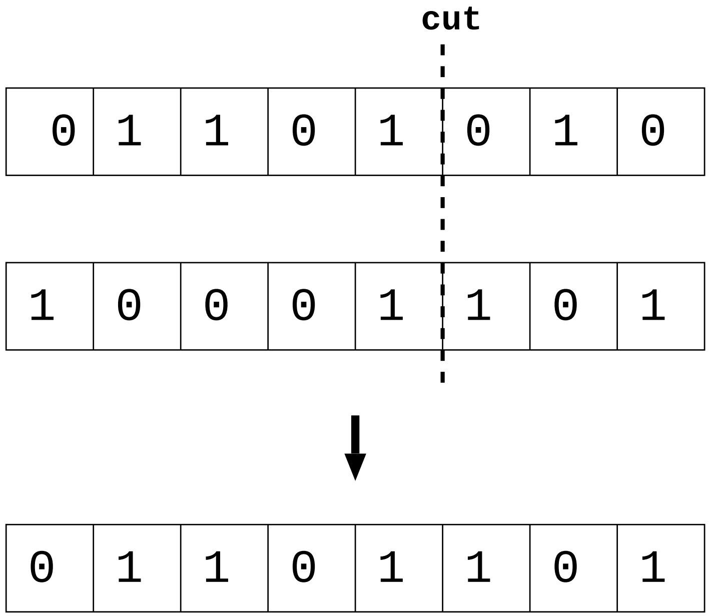
- selection scheme
  - prefer better solutions for survival
  - elitist strategies
  - maintenance of population diversity
- local search
  - often local search is useful to improve solutions
  - population based search in the space of local optima
    $\rightsquigarrow$ memetic algorithms
- stopping criteria
  - fixed number of generations
  - convergence of population

GA application to SAT

solution representation: a binary string

evaluation function: # of violated clauses

crossover operator:



mutation: with a fixed probability flip a variable's truth value

selection: choose best p strings for the next population avoiding duplicate solutions

additional local search: after each crossover or mutation apply a 1-opt local search

Discussion

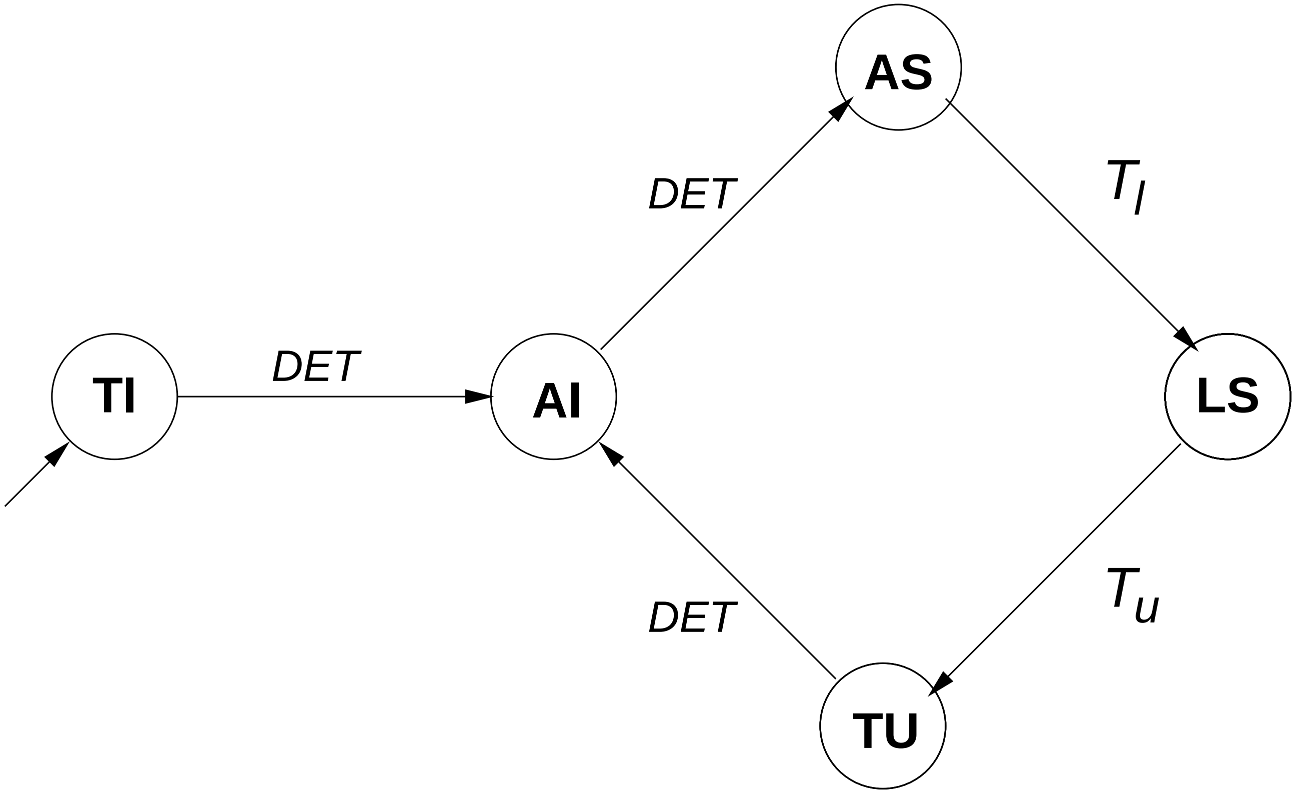
- population provides an easy way to increase search space exploration
- large number of different implementation choices
- best performance achieved when operators take problem characteristics into account
- good results for a wide variety of applications

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 “pheromone trails” (odorous substance)
- 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



Application principles:

- 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

ACO for the TSP – tour construction

- Solution components are “arcs”
- $\eta_{ij} = 1/d_{ij}$: Heuristic information, indicates the utility of going from node i to node j
- $\tau_{ij}(t)$: Intensity of the pheromone trail in iteration t on arc (i, j)
- Probabilistic selection of the next node according to:

$$p_{ij}(t) \sim (\tau_{ij}(t))^{\alpha} \cdot (\eta_{ij})^{\beta} \text{ if node } j \text{ allowed}$$

ACO for the TSP – Update of pheromone trails

- Parameter $0 < \rho < 1$, $1 - \rho$ represents pheromone evaporation
- Update of the pheromone trails according to (Ant System):

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

- $\Delta\tau_{ij}^{k} = 1/L_k$ if arc (i, j) is used by ant k on its tour
  L_k: Tour length of ant k
  m: Number of ants

Several improved extensions have been proposed.

Discussion

- General framework for ACO applications: ACO Metaheuristic [Dorigo, Di Caro 1999, Dorigo, Di Caro, Gambardella, 1999]
- Two main application fields:

  static problems $\longleftrightarrow$ dynamic problems
- considerable success despite being a very recent metaheuristic

Characteristics of Various Metaheuristics

Feature	SA	TS	ILS	GA	ACO
Trajectory	$+$	$+$	$-$	$-$	$-$
Population	$-$	$-$	$-$	$+$	$+$
Memory	$-$	$+$	$\exists$	$\exists$	$\exists$
Multiple neighborhoods	$-$	$-$	$+$	$\exists$	$-$
Sol. construction	$-$	$-$	$-$	$-$	$+$
Nature-inspired	$+$	$-$	$-$	$+$	$+$

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

Chance vs. Necessity

- randomisation of search algorithms often increases their performance and robustness
- stochastic search algorithms need to balance randomised and goal-directed search (exploration vs. exploitation, diversification vs. intensification)
- best performance will be achieved if good, problem specific balance is found
- different techniques can be used to achieve this balance
- peak performance obtained by successful exploitation of problem and algorithm specific knowledge
- this knowledge is often still poorly understood

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

$\leadsto$ empirical methods are used to analyse and characterise their 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...

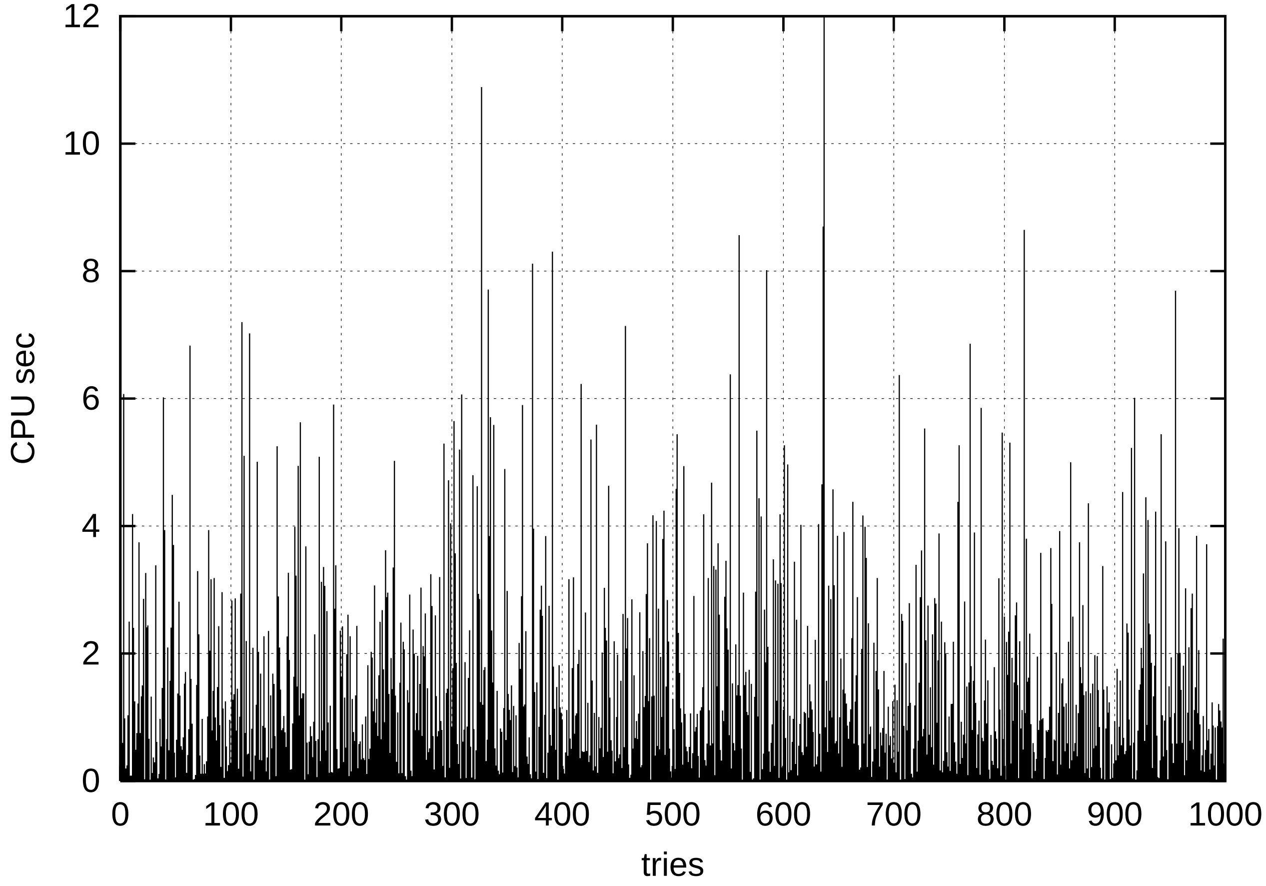
- help assessing suitability for applications
- provide basis for comparing algorithms
- characterise algorithm behaviour
- facilitate improvements of algorithms

RTD-based empirical methodology:

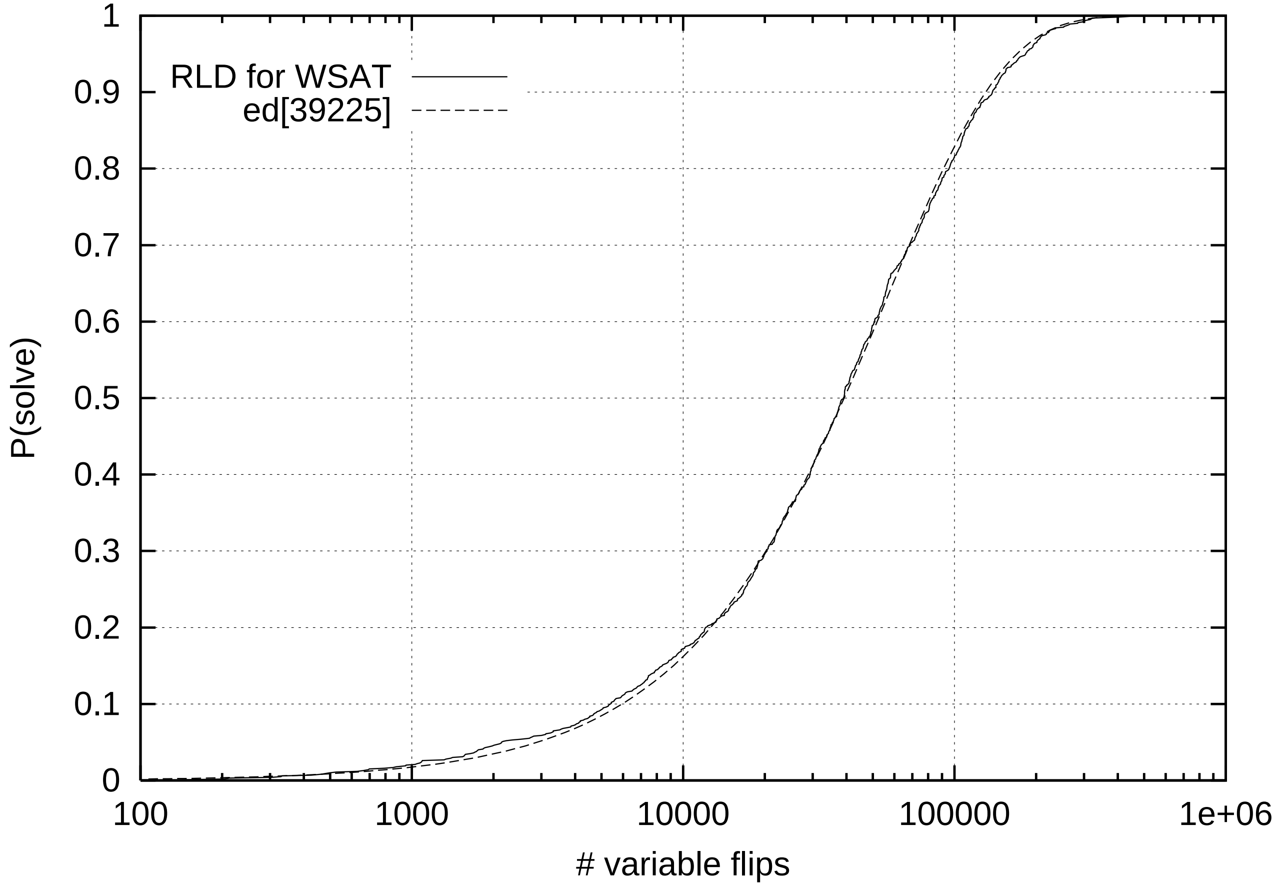
- run algorithm multiple times on given problem instance(s)
- estimate empirical run-time distributions (RTDs)
- get simple descriptive statistics (mean, stddev, percentiles, ...) from RTD data
- approximate empirical RTDs with known distribution functions
- check statistical significance using goodness-of-fit 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 $\rightarrow \infty$.

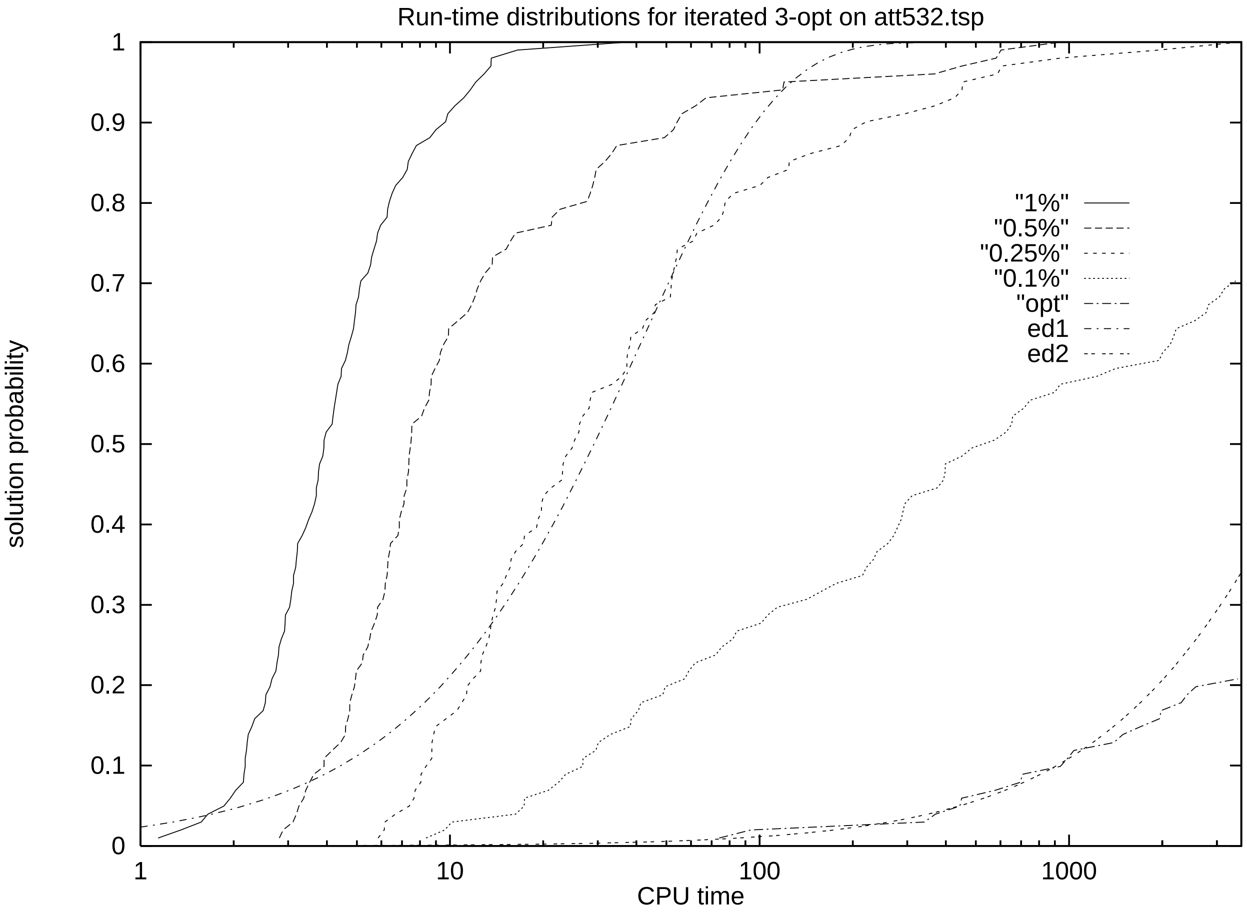
Some results [Hoos 1999]:

- 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 SAT algorithms known to date.

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



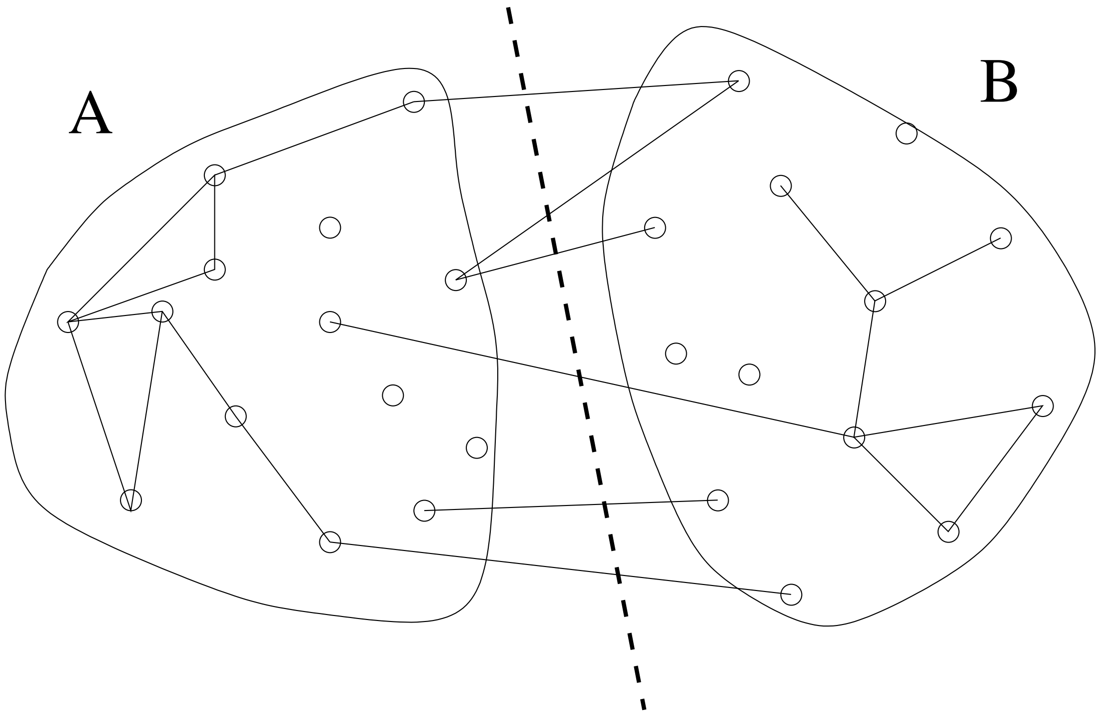
SQD-based methodology randomised optimisation algorithms:

- 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

Questions for SQD-based methodology:

- How do the SQDs scale with increasing run-time?
- What is the limiting shape of the SQDs with increasing instance size?

SQD analysis for Graph Partitioning (GP)



[Martin, Houdayer, Schreiber, 1999] studied best performing SLS algorithms for GP on ensemble of randomly generated graphs.

Results:

- SQD distributions approach a limiting Gaussian shape, both within individual graphs and across all graphs.

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

- Similar results also on the TSP.

Use solution quality distribution to estimate optimum

- given a sample of k feasible solutions and let x be the extreme value from the sample
  $\rightsquigarrow$ for increasing k, the distribution of x approaches a Weibull distribution with position parameter μ, where μ is optimal solution value [Dannenbring, 1977]

Estimation of optimum possible:

- generate m independent samples of x
- estimate parameters of Weibull distribution
- get confidence interval for optimum

Search Space Analysis

Intuition:

- algorithm searches in a *fitness landscape* of the problem to be solved
- fitness landscape can be imagined as a mountainous region with hills, craters, valleys ..

Goal: Findest lowest point (for minimization problems)

$\leadsto$ analyse structure of the fitness landscape

Fitness landscape is defined by:

- the set of all possible solutions (search space)
- an objective function that assigns to every solution a fitness (objective function) value
- a neighbourhood structure which induces a distance measure between solutions

Distance between solutions:

- defined as the minimum number of moves to convert one solution into another one
- often surrogate distance metrics are used (e.g., distance between tours in the TSP interpreted as number of different edges in two tours)

Important aspects of fitness landscapes:

- number of local optima
- ruggedness of the fitness landscape
- distribution of local minima and their relative location to the global minima
- topology of basins of attraction of local optima

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]

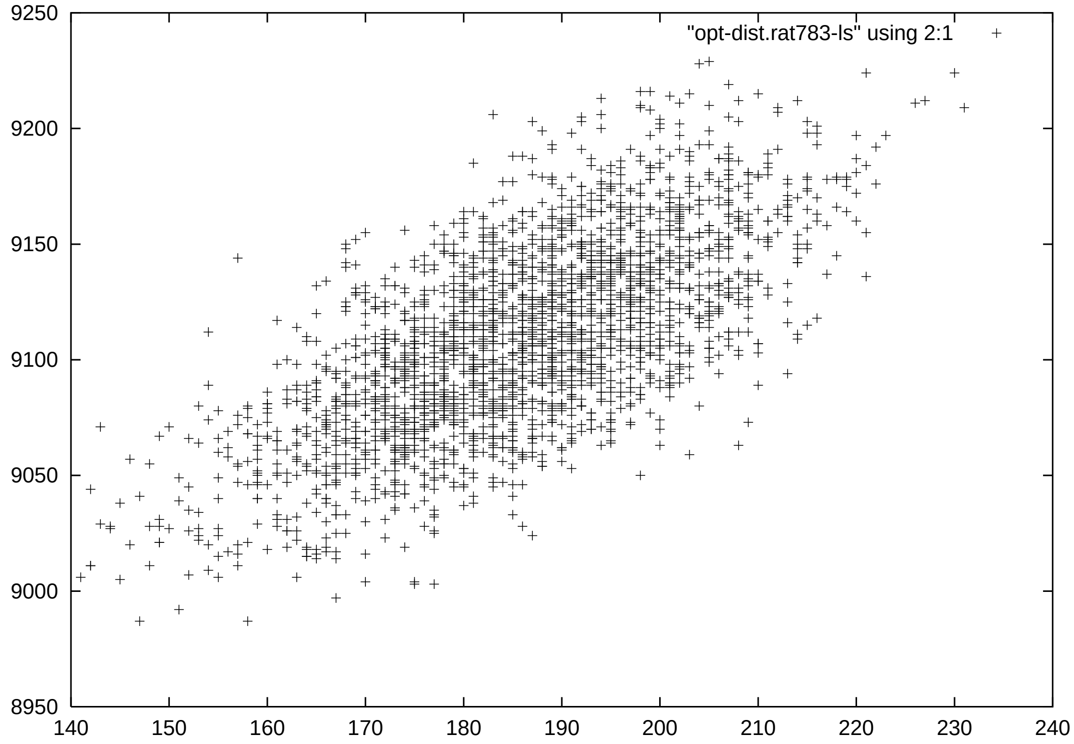
Measure for fitness-distance correlation:

- 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

Results:

- local minima are contained in a small subspace of the whole search space
- solution fitness guides SLS algorithms towards better solutions $\leadsto$ the shorter (on average) the tours are, the closer they are to the optimal one

Fitness-distance correlations have been analysed for several other problems.

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
  $\leadsto$ 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
  $\leadsto$ reactive search

Parallelising Stochastic Search

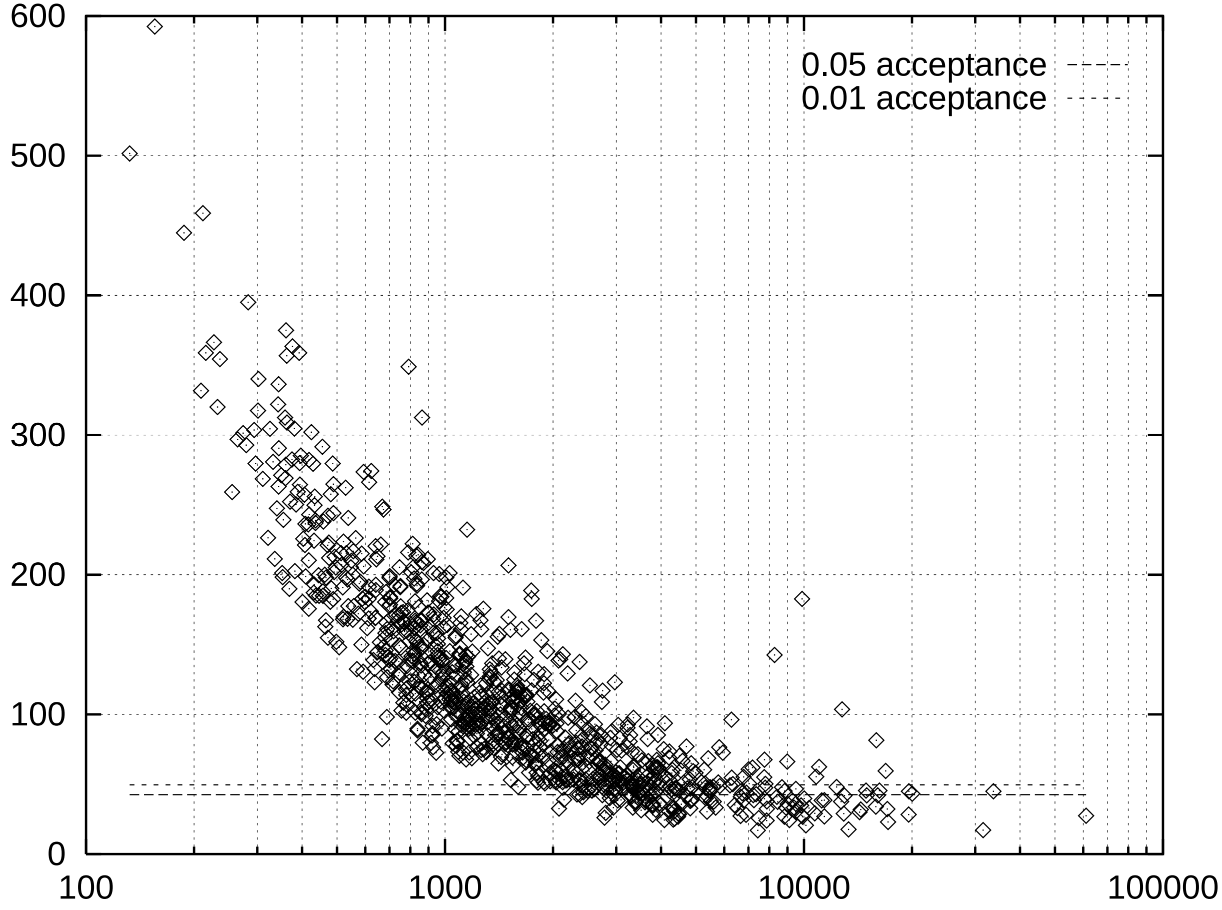
- randomised algorithms allow extremely easy and scalable parallelisation: multiple independent tries
- effectiveness depends on run-time behaviour of underlying algorithms
- under certain conditions, optimal speedup can be obtained

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

$\leadsto$ **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).

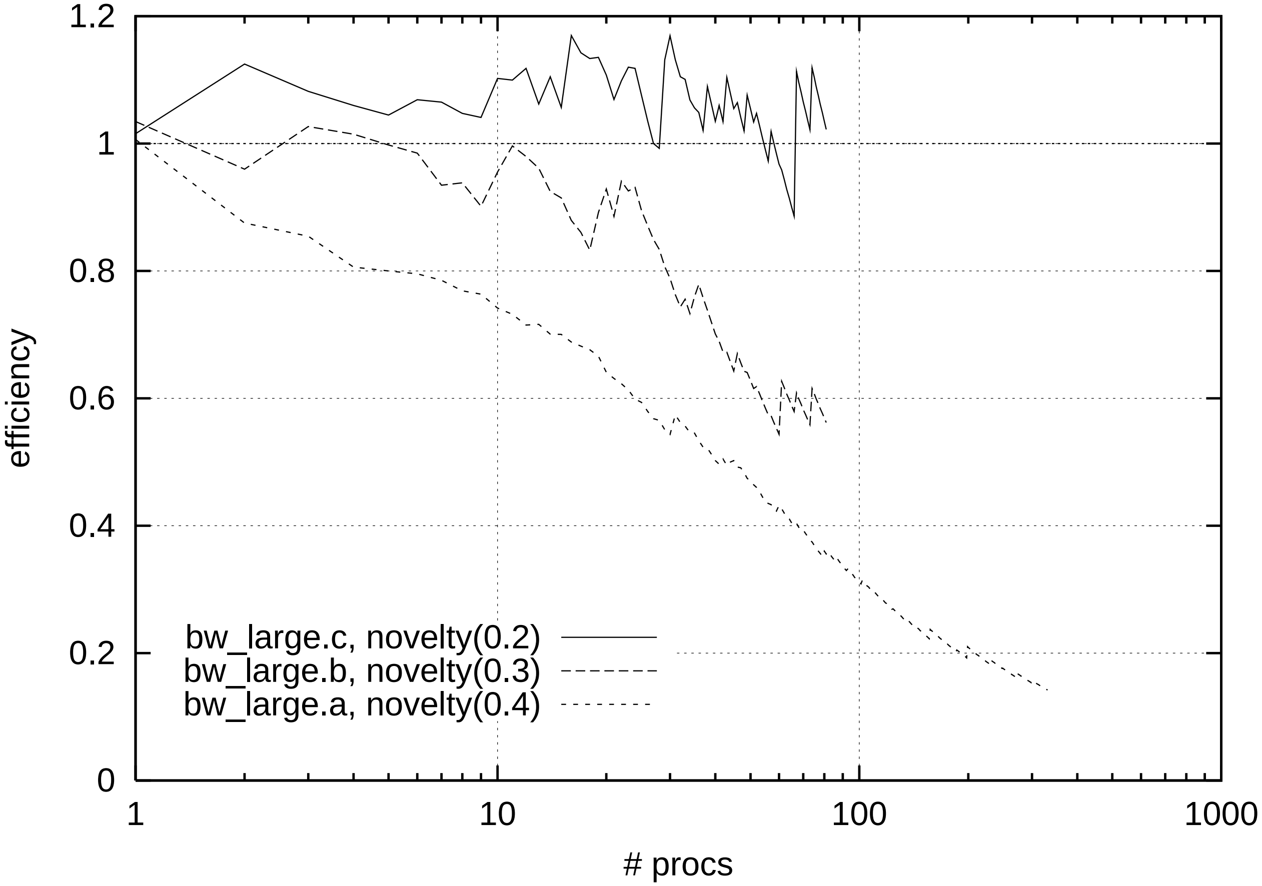
Goodness of fit for hard Random-3-SAT instances



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

Efficiency of multiple independent tries parallalelisation



ACO algorithms

- Fine-grained, not successful
- Coarse-grained, more succesful

Genetic Algorithms

- Work on *fine-grained* and *coarse-grained* parallel implementations

Simulated Annealing

- Parallel Runs
- Evaluation of moves

Tabu Search

- Exponential run-time distributions (QAP)
- Cooperative approaches [Crainic et al.]

Part IV

Applications

Applications of Stochastic Search

Some recent successful applications:

- SAT [e.g., Selman et al. 1992–1997; Hoos et. al. 1994–2000]
- TSP [e.g., Johnson & McGeoch 1997; Stützle & Hoos 1997–2001]
- Quadratic Assignment Problem [e.g., Stützle 1997-1999; Taillard 1995]
- Scheduling [e.g., den Besten, Stützle, Dorigo 2000]
- Planning [e.g., Kautz et al. 1996–1999; Brafman & Hoos 1998–2000]
- Combinatorial Auctions [Hoos & Boutilier 2000]

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]

SLS algorithms for SAT

- search through space of 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

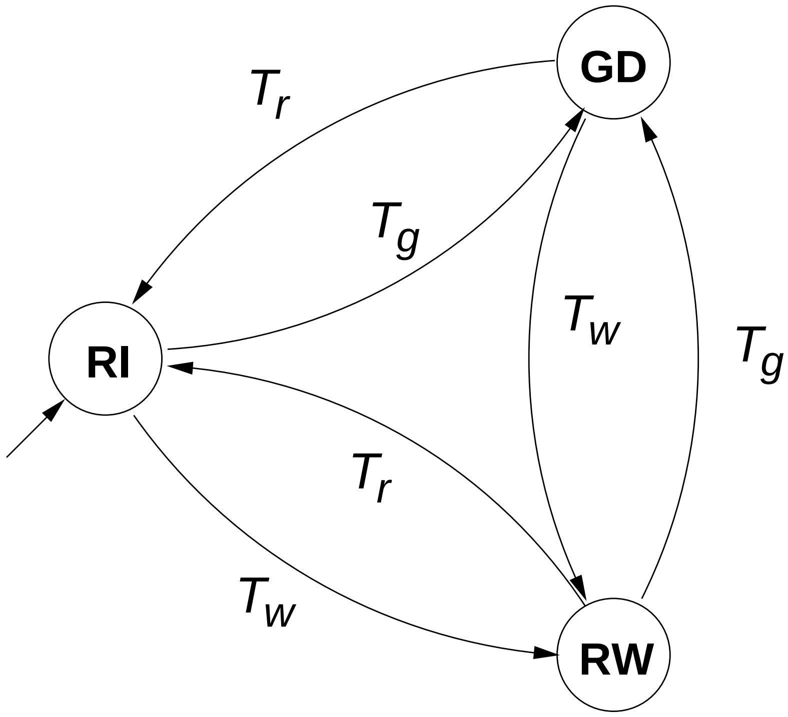
The GSAT Algorithm [Selman, Mitchell, Levesque, 1992]

- 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 a currently unsatisfied clause and a literal in that clause, flip the corresponding variable to satisfy this clause
- GWSAT steps: choose probabilistically between a GSAT step and a Random Walk step with 'walk probability' (noise) *wp*

GLSM representation of GWSAT



The WalkSAT algorithm family [McAllester et al., 1997]

- search initialisation: randomly chosen assignment
- search step
  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
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

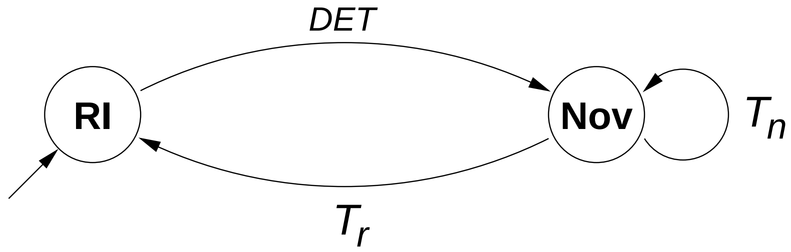
- Tabu, Novelty, and R-Novelty: essentially incomplete, i.e., can get stuck in non-solution areas of search space

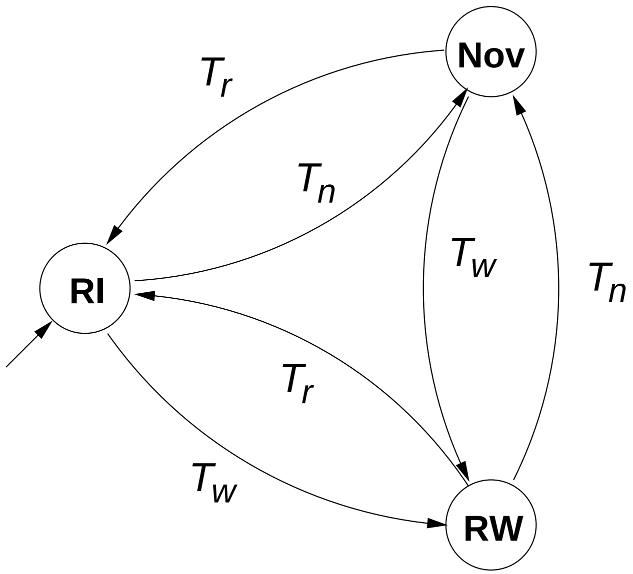
Novelty$^+$ and R-Novelty$^+$ [Hoos 1998]:

- 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

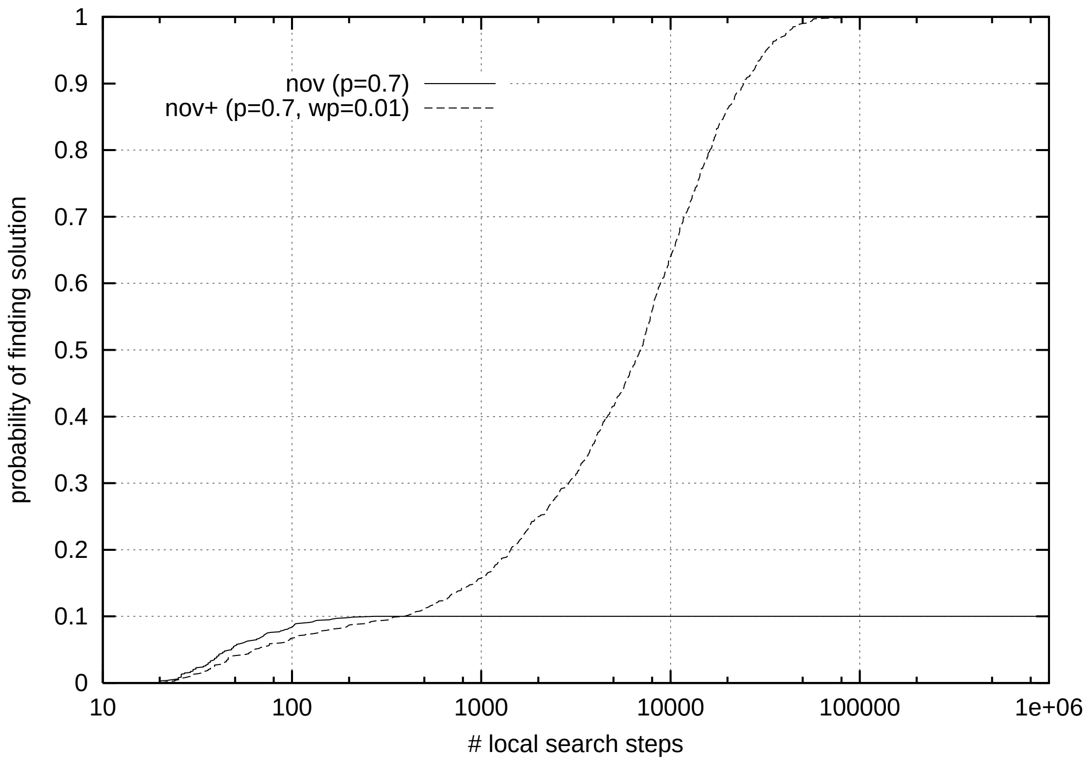
These algorithms are amongst the best-performing algorithms for SAT known today; they solve hard instances up to $> 10,000$ *variables and* $> 100,000$ *clauses*

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]
- REL_SAT [Bayardo & Schrag, 1997]
- Satz-Rand, REL_SAT-Rand [Gomes et al., 1998]

Similar SLS algorithms are successfully used to solve more general problems:

- MAX-SAT [e.g., Hansen & Jaumard, 1990; Battiti & Protasi, 1997]
- CSP [e.g., Minton et al., 1990–1992; Stützle, 1997; Galinier, Hao 1997]
- Dynamic SAT [Hoos & O'Neill, 2000]

The Travelling Salesman Problem (TSP)

The TSP ...

- has been a source of inspiration for new algorithmic ideas
- is a standard test-bed for exact and approximate algorithms

Recent contributions to TSP solving:

- finding approximate solutions of very high quality
- pushing the frontier of tractable instance size
- understanding of algorithm behaviour

Approaches for the TSP:

- exact algorithms: Branch & Cut
  $\leadsto$ largest TSPLIB instance solved has 13,509 cities!
- SLS algorithms:
  - construction heuristics
  - iterative improvement algorithms
  - metaheuristics

Results for SLS algorithms:

- construction heuristics and iterative improvement can be applied to very large instances ($> 10^6$ cities) with considerable success [Applegate et al. 2000; Johnson, McGeoch 1997]

- best performance results w.r.t. solution quality obtained with iterated local search, genetic algorithms, or more TSP-specific approaches [Applegate et al., 2000; Johnson, McGeoch, 1997; Merz, Freisleben, 1997; Stützle, Hoos, 1999]

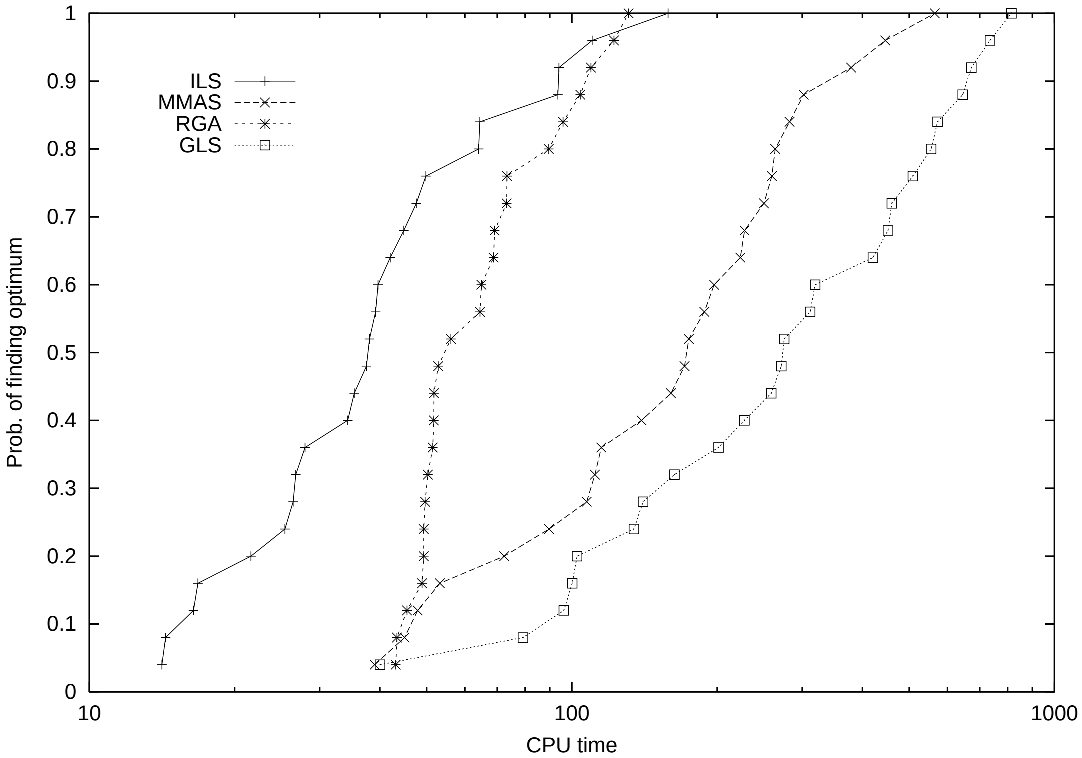
- best performing hybrid algorithms use 3-opt or Lin-Kernighan local search

- 8th DIMACS Implementation Challenge on the TSP gives state–of–the–art results

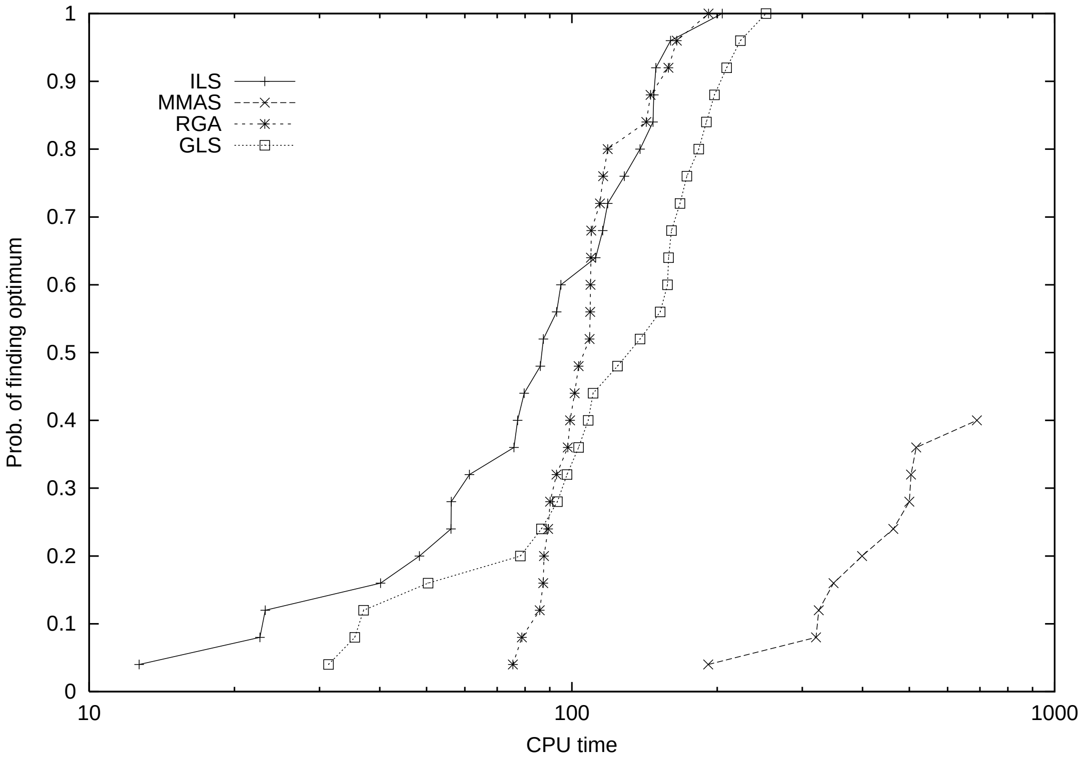
Comparison of SLS algorithms for the TSP [Stützle, Grün, Linke, Rüttgers, 2000]:

- TSP algorithms compared:
  - Iterated Local Search with diversification (ILS)
  - Genetic Algorithm with DPX-crossover (DPX)
  - Repair-based genetic algorithm (RGA)
  - ACO algorithm (MMAS)
- all algorithms use same 3-opt local search and same data structures
- all algorithms are run on a same machine
- comparative analysis based on RTDs

Comparison of TSP algorithms on instance rat783



Comparison of TSP algorithms on instance pr1002



Results:

- very good performance by all algorithms
- no algorithm dominates all others on all instances
- ILS algorithm performs best on most instances
- all algorithms need effective diversification features to achieve best performance
- additional experiments with other 3-opt implementations of 3-opt showed that specific implementation details can result in significant performance difference
- some RTDs are “steeper” than an exponential distribution

The Quadratic Assignment Problem (QAP)

Given:

- n items to be assigned to n locations
- a_{ij} : Flow from item i to item j
- d_{rs} : Distance between location r and location s

Objective: Minimise sum of products flow $\times$ distance

The QAP ...

- is a generic problem in layout design
- has applications in hospital layout, typewriter keybord design, ..
- is considered to be one of the "hardest" optimization problems for exact solution

$\leadsto$ *need for approximate solution*

Local search algorithms for the QAP:

- search space given by all possible assignments, represented by permutations
- assignments are neighboured if they differ in the location of exactly two items

SLS approaches for the QAP:

- SA, TS, ILS, EA, ACO have all been to the QAP
- several additional approaches to the QAP
- relative performance depends strongly on instance characteristics

Four classes of QAP instances in QAPLIB [Taillard, 1995]:

- randomly generated instances according to uniformly distributed matrix entries (class I)
- random flows on grids (class II)
- real-life problems (class III)
- randomly generated problems which resemble the structure of real-life problems (class IV)

Indication of type:

- flow and/or distance dominance

$$fd(A) = 100 \cdot \frac{\sigma}{\mu}, \text{where}$$

$$\mu = \frac{1}{n^2} \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}; \; \sigma = \sqrt{\frac{1}{n^2 - 1} \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} (a_{ij} - \mu)}$$

- sparsity of flow and/or distance matrix

Fitness-Distance Analysis:

- instances of class I show no significant fitness-distance correlations
- most of the instances of classes II and III do have significant fitness-distance correlations; the largest correlations are observed for instances of classes III and IV
- local minima are spread over the whole search space
- some instances have many optimal solutions
- dominance and sparsity give indication of instance type

Case study: ACO algorithm ($\mathcal{MM}$AS) for the QAP

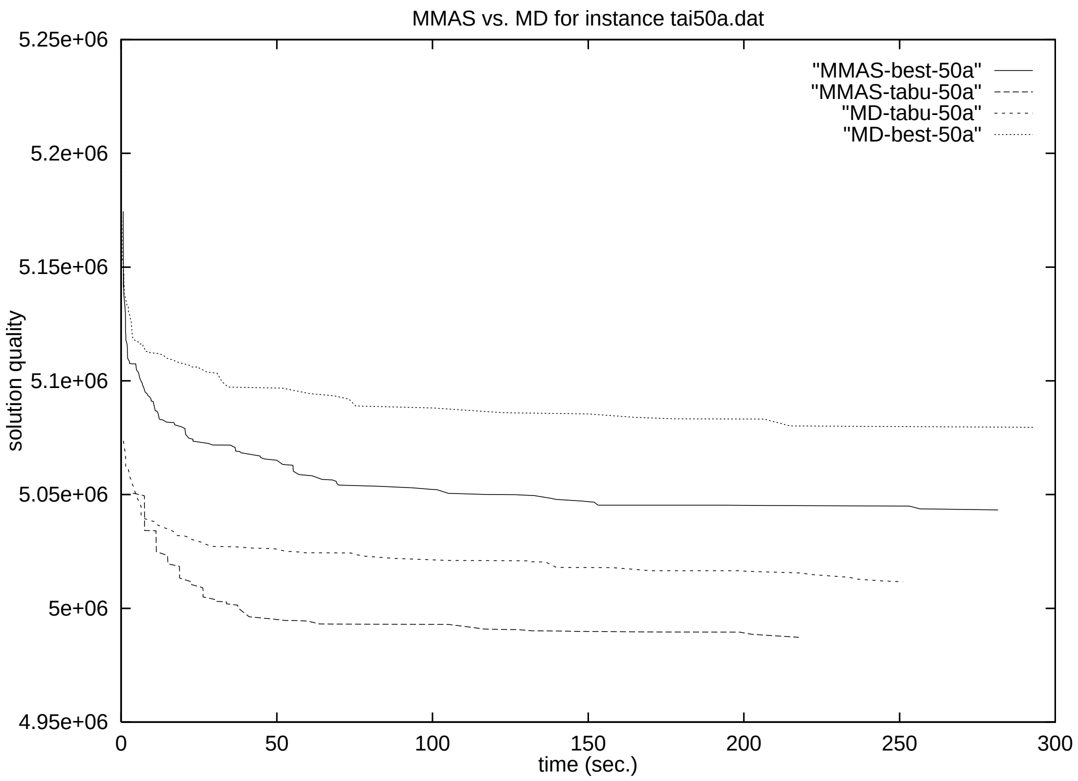
- ACO algorithm constructs candidate solutions by iteratively assigning items to locations
- pheromone trails τ_{ij} refer to the desirability of assigning item i to location j
- candidate solution construction only uses pheromone trails
- candidate solutions are improved by local search

Which type of local search to be used?

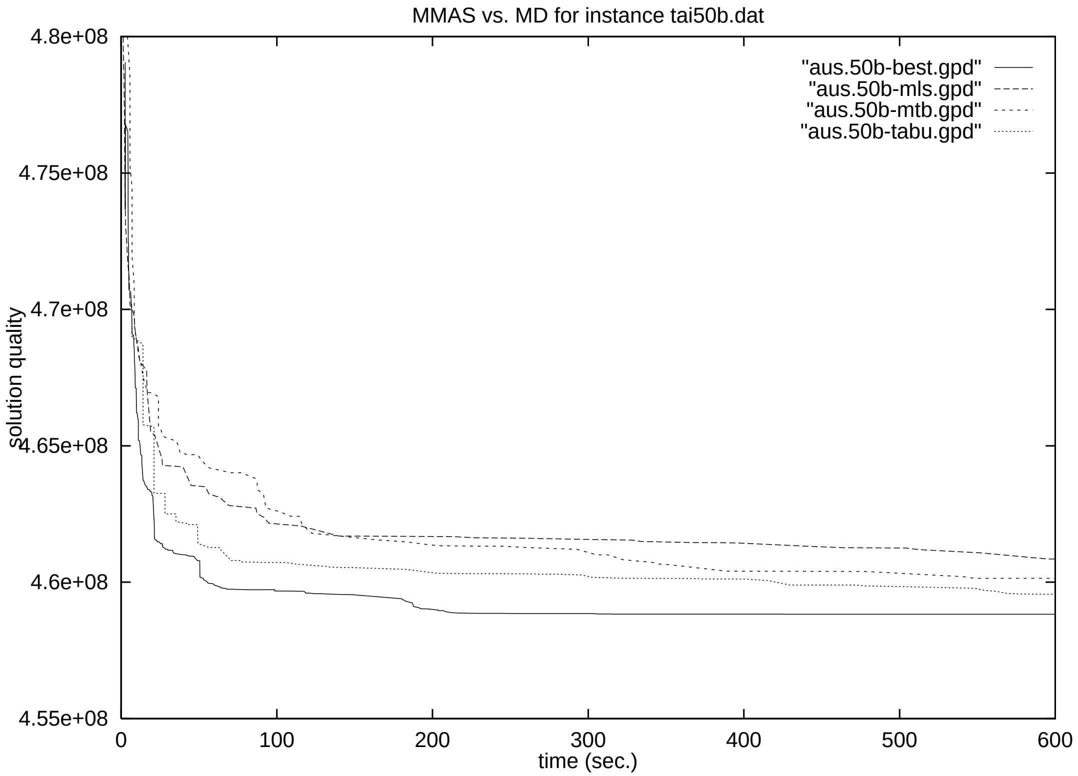
- 2-opt local search algorithm ($\mathcal{MM}$AS$_{\texttt{2-opt}}$)
  $\leadsto$ needs only few steps to reach local minimum
- short tabu search runs ($\mathcal{MM}$AS$_{\text{TS}}$)
  $\leadsto$ more run-time intensive than 2-opt but finds candidate solutions of better quality than 2-opt

Tradeoff: Speed vs. Quality

Solution quality development on class I instance



Solution quality development on class IV instance



Computational comparison of QAP algorithms:

- algorithms compared are among the best known for the QAP
  - robust tabu search (Ro-TS) [Taillard, 1991, 1995]
  - genetic hybrids (GH) [Fleurent, Ferland, 1994]
  - hybrid ant system (HAS) [Gambardella et al., 1999]
  - $\mathcal{MM}$AS$_{\text{TS}}$ and $\mathcal{MM}$AS$_{\texttt{2-opt}}$ [Stützle, Hoos, 1997-2000]
- algorithms use approx. same computation time
- results are given as the average deviation from best known candidate solutions over 10 independent algorithm runs

instance	Ro-TS	GH	HAS	$\mathcal{MM}\mathrm{AS}_{\mathrm{TS}}$	$\mathcal{MM}\mathrm{AS}_{2-\mathtt{opt}}$
uniform, random instances; class I					
`tai40a`	0.990	**0.884**	1.989	0.933	1.131
`tai50a`	1.125	**1.049**	2.800	1.236	1.900
`tai60a`	1.203	**1.159**	3.070	1.372	2.484
`tai80a`	0.900	**0.796**	2.689	1.134	2.103
random flows on grids; class II					
`nug30`	0.013	**0.007**	0.098	0.026	0.042
`sko72`	0.146	0.143	0.277	**0.109**	0.243
`sko81`	0.136	0.136	0.144	**0.071**	0.223
`sko90`	**0.128**	0.196	0.231	0.192	0.288

instance	Ro-TS	GH	HAS	$\mathcal{MM}AS_{TS}$	$\mathcal{MM}AS_{2-\mathtt{opt}}$
real-life instances; class III					
`bur26a-h`	0.002	0.043	**0.0**	0.006	**0.0**
`kra30a`	0.268	**0.134**	0.630	**0.134**	0.314
`kra30b`	**0.023**	0.054	0.071	**0.023**	0.049
`ste36a`	0.155	n.a.	n.a.	**0.036**	0.181
`ste36b`	0.081	n.a.	n.a.	**0.0**	**0.0**
real-life like instances; class IV					
`tai40b`	0.531	0.211	**0.0**	0.402	**0.0**
`tai50b`	0.342	0.214	0.192	0.172	**0.002**
`tai60b`	0.417	0.291	0.048	**0.005**	**0.005**
`tai80b`	1.031	0.829	0.667	0.591	**0.096**

Results:

- relative performance of algorithms depends strongly on instance class
- frequent application of a fast local search performs better on more structured instances
- $\mathcal{MM}$AS among best algorithms for the QAP

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, . . .

The Single Machine Total Weighted Tardiness Problem

Often, single machine is the bottleneck in production environment.

Given (for each item):

- due date d_j given for each task
- weight (importance) w_j for each single task
- processing time p_j for each task

Objective: minimise sum of weighted tardiness

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

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

Local search algorithms for SMTWTP:

- several construction heuristics available
- search space given by all possible task sequences
- possible neighbourhoods:
  - swaps of two neighboring jobs at position i and $i + 1$
    swap-neighbourhood
  - interchanges of jobs at ith and jth position
    interchange-neighbourhood
  - removals of job at ith position and insertion in jth position
    insert-neighbourhood

More effective local search:

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

Idea: concatenate local search in different neighbourhoods
this idea is systematically exploited in variable neighbourhood search [Mladenovic, Hansen, 1995–2000]

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

Instances: 125 randomly generated benchmark instances with 100 tasks; 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.	inter+insert			insert+inter		
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
  $\leadsto$ more complex neighbourhood yields more effective local search
- other local search extension:
  Dynasearch [Congram, Potts, van de Velde 1998]
- final solution quality depends strongly on initial candidate solution
- improvements possible by hybrid algorithms
  $\leadsto$ extensions using ILS and ACO

ILS algorithm:

- candidate solution modification: variable number of random left-insert moves
- acceptance criterion: Apply candidate solution modification to best candidate solution since start of the algorithm
- local search: several possibilities examined, best results with interchange-insert concatenation

ACO algorithm:

- candidate solutions are constructed starting from an empty sequence by iteratively appending tasks
- pheromone trail τ_{ij} refers to desirability of assigning task i to position j
- candidate solution construction uses pheromone trails and heuristic information
- local search: half the colony applies insert-interchange, the other half interchange-insert local search

Summary of results:

- excellent performance of both approaches
- they consistently find best-known candidate solutions on all known benchmark instances in reasonable time (few seconds up to some minutes)
- heterogeneous colony in ACO improved robustness
- ILS on most instances faster than ACO
- benchmark instances obtained with specific parameter settings for instance generator are very easily solved
- search space analysis could identify reasons for good performance

Planning

Given: set of actions (operators), initial and goal state

Objective: find sequence (or partially ordered collection) of actions which allow reaching goal state from initial state

Some prominent AI planning problems:

- blocks world
- logistics
- configuration

Planning is an $\mathcal{NP}$-hard combinatorial problem

Some algorithms for planning:

- classic approaches: forward planning / regression planning
- GraphPlan [Blum & Furst, 1995]
- SATPLAN, Blackbox [Kautz & Selman, 1996-1999]
- LPSP [Brafman & Hoos, 1998-1999]
- Fast Forward [Hofmann & Nebel, 2000-2001]

The Planning-as-SAT approach:

- encode planning instance into SAT
  (given fixed/max plan length)
- use polynomial preprocessing to simplify SAT instance
  (unit propagation, subsumption, etc.)
- use SAT algorithm to solve simplified SAT instance
  (WalkSAT, satz, etc.)
- decode propositional model into plan

Advantages:

- highly competitive with best known general purpose planners [Kautz & Selman, 1996]
- makes use of very advanced propositional reasoning technology
- benefits from any improvements in SAT solving (software & hardware)

Potential disadvantages:

- potentially more difficult to integrate domain-specific knowledge
- very large, explicit intermediate representations (CNF formulae)
- overhead through encoding / decoding

Lifting SLS techniques to plan level:

- search directly in a space of candidate plans
- use SLS techniques for SAT at plan level
- add new, planning specific SLS strategies (GLSM states)

$\rightsquigarrow$ LPSP algorithm for linear planning [Brafman & Hoos, 1998–1999]

Search initialisation: Bi-directional stochastic search (construction heuristic)

Search steps:

Best-Replacement: replace single action such that a maximal number of flaws (unsatisfied preconditions of actions) is removed from the current plan

Flaw-Repair: randomly chose a flaw from the current plan and repair it by replacing a single action

Reorder: reorder current plan heuristically, trying to minimise number of flaws and throwing out superfluous actions

Additionally: use reachability analysis as preprocessing step to prune out impossible actions

Run-time of LPSP vs. SATPLAN

instance	plan length	LPSP		WalkSAT		satz
		mean	vc	mean	vc	
bw_large.a	6	0.096	0.53	0.73	n/a	0.56
bw_large.b	9	1.22	0.82	3.61	0.88	1.44
bw_large.c	14	4.62	0.61	62.96	0.99	5.82
bw_large.a	18	31.18	0.60	152.05	0.94	> 60
log.a	6	0.068	0.48	4.33	n/a	4.33
log.b	10	0.69	1.37	11.56	n/a	11.58
log.c	16	13.50	1.11	1,075.17	0.67	> 60

Advantages:

- excellent performance for linear planning
- facilitates combination of general-purpose and domain-specific SLS strategies
- avoids large intermediate representations
- easier to analyse / understand

Disadvantages:

- more complex, planning specific SLS algorithm
- more difficult to implement
- allows less use of “off-the-shelve” solver technology

Combinatorial Auctions

Auctions are attractive as mechanisms for ...

- E-commerce (business transactions, sales)
- resource allocation
  (radio frequencies, pollution rights, scheduling)
- coordinating agents in multi-agent systems

$\leadsto$ increasing interest in CS / AI

Standard, single item auctions:

- winner determination is easy
- cannot express complementarities of goods

Combinatorial auctions (CAs):

- bidding for bundles
- allows expressing preferences more directly
- minimises buyer's risk of getting incomplete combinations

Example: (fictitious — at least for now)

Ferry tickets (restricted ferry capacity):
people bid for the guarantee to get on a ferry on certain days.

Bidder A wants to go from Seattle to Victoria (BC) over the weekend, thus has to bid for Fri and Sun sailing

$\leadsto$ *A needs both access permits (=bundle of items)*

Winner determination for CAs

Given:

- set of goods to be auctioned
- set of bids for combinations (sets) of these goods

Problem:

Determine assignment of goods to bidders maximising total revenue (sum of prices for satisfied bids)

SLS algorithms for CA

search space: space of feasible, partial assignments

solution set: assignments with optimal / given revenue

neighbourhood relation: assignments reachable by reassigning set of goods such that one formerly unsat bid becomes satisfied

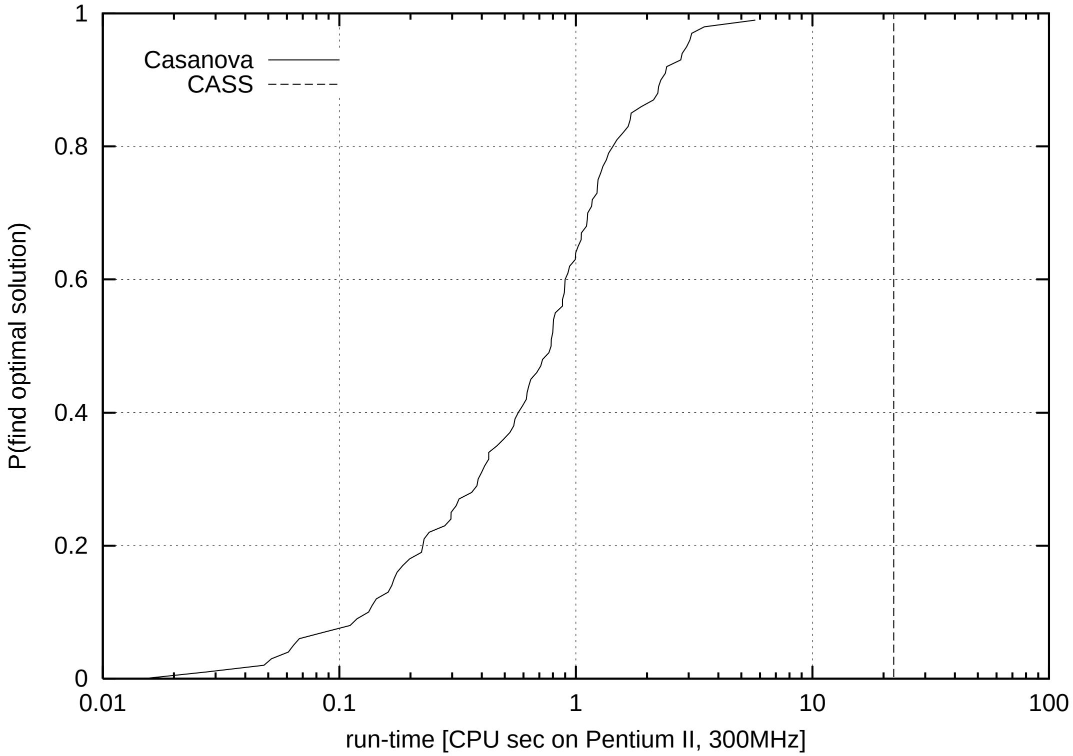
search initialisation function: start with empty assignment

search step function: picks a neighbour according to stochastic heuristic h

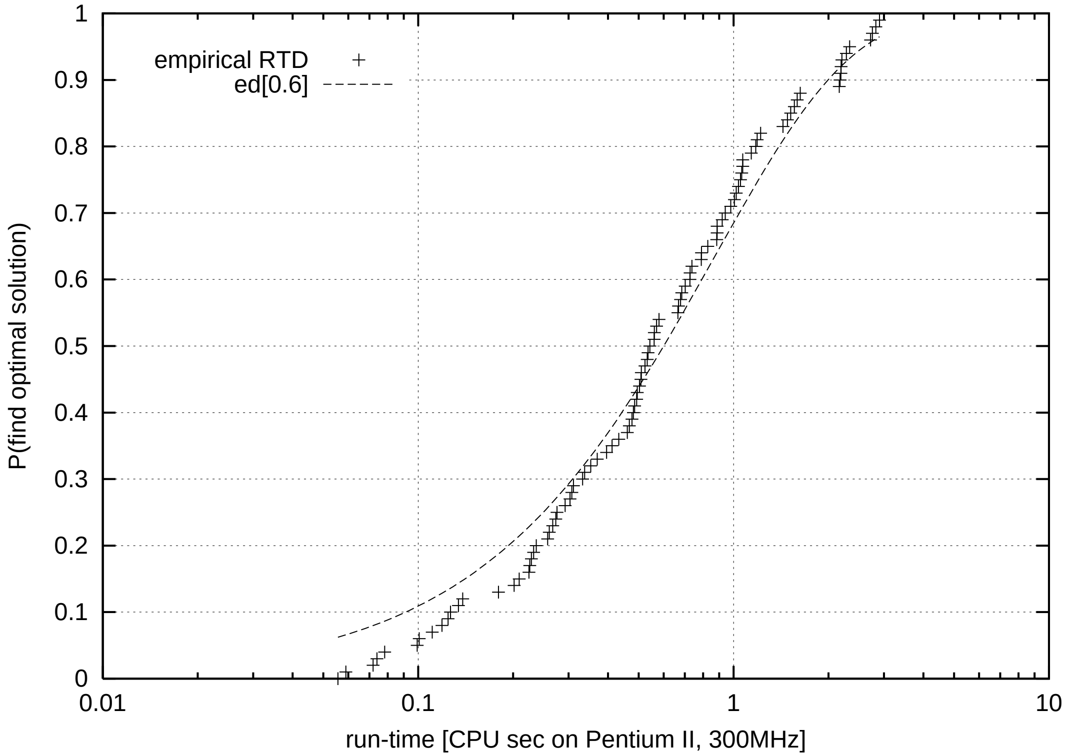
Casanova, one algorithm of this family ...

- was designed based on principles and methods developed and tested for SLS algorithms for SAT
- outperforms best known algorithm for CA (CASS) for large problems with fixed cutoff times significantly
- although incomplete, finds optimal solutions to small CA instances, often much faster than systematic search
- scales better with growing problem size than other methods
- can be easily parallelised with optimal speedup

Run-time distributions for finding optimal solution for a hard instance with 100 goods, 100 bids



Run-time distributions for finding optimal solution for a hard instance with 200 goods, 200 bids



Average Revenue for CASS vs. Casanova (fixed run-time)

problem type	# goods	# bids	CASS	Casanova	Gain
simple uniform (3)	200	10000	281413	286164	1.69%
simple binomial (0.01)	500	5000	583279	616708	5.73%
simple exponential (5)	500	5000	647629	655329	1.19%
cnf uniform (3)	100	100	104868	127011	21.12%
cnf poisson (2)	100	500	101568	135973	33.87%
k-of uniform (2,4,2)	100	100	48812	59938	22.79%

Conclusions and Issues for 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
  ($\leadsto$ search space analysis, new theoretical & empirical results)

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

- application to real-world problems
  (e.g., intelligent systems, e-commerce, bioinformatics)

Stochastic search is a general approach for solving combinatorial problems with significant research and practical potential, *but:* it is certainly no panacea.