Simulated Annealing, Dynamic Local Search, GRASP, Iterated Greedy

—

an overview

Thomas Stützle

stuetzle@informatik.tu-darmstadt.de
http://www.intellektik.informatik.tu-darmstadt.de/~tom.

Darmstadt University of Technology
Department of Computer Science
Intellectics Group

Outline

- simulated annealing
  - basics
  - theory
  - applications
- dynamic local search
  - basics
  - applications
- greedy randomized adaptive search procedures (GRASP)
  - basics
  - applications
- iterated greedy
  - basics
  - applications

Escaping local optimality

- occasionally accept worse solutions
  - tabu search
  - simulated annealing
- modify evaluation function at run time
  - dynamic local search
- generate new solutions (for a local search)
  - iterated local search
  - memetic algorithms / EAs
  - ant colony optimization
  - GRASP
- constructive methods
  - ant colony optimization
  - iterated greedy

Notation

- $\mathcal{S}$: set of (candidate) solutions
- s: solution in $\mathcal{S}$
- f: cost function
- $f(s)$: cost function value of solution s
- $\mathcal{N}(s)$: neighborhood of s

here, we assume that we solve minimization problems

What is Simulated Annealing (SA)?

Simulated Annealing is an SLS method that tries to avoid local optima by accepting probabilistically moves to worse solutions.

- Simulated Annealing was one of the first SLS methods
- now a "mature" SLS method
  - many applications available (ca. 1,000 papers)
  - (strong) convergence results
- simple to implement
- inspired by an analogy to physical annealing

Physics analogy: annealing

- annealing is a thermal process for obtaining low energy states of a solid through a heat bath.
  1. increase the temperature of the solid until it melds
  2. carefully decrease the temperature of the solid to reach a ground state (minimal energy state, cristaline structure)
- computer simulations of the annealing process
  - models exist for this process based on Monte Carlo techniques
  - Metropolis algorithm
    simulation algorithm for the annealing process proposed by Metropolis et al. in 1953

Metropolis algorithm

- generates a sequence of states
  1. given state i with energy E_i, generate subsequent state j with energy E_j by some perturbation mechanism
  2. If $E_j - E_i \leq 0$, then accept j, otherwise accept j with probability
  $$\exp\left(-\frac{E_j - E_i}{k_B \cdot T}\right)$$
  k_B: Boltzmann constant, T: temperature
- if temperature is lowered slow enough, the solid may reach thermal equilibrium at each temperature
- thermal equilibrium characterized by Boltzman distribution

$$P_T(\mathbf{X} = i) = \frac{\exp\left(-E_i/k_B \cdot T\right)}{\sum_j \exp\left(-E_j/k_B \cdot T\right)}$$

Phys. annealing vs. optimization

physical system		comb. optimization problem
state	$\longleftrightarrow$	candidate solutions
energy of a state	$\longleftrightarrow$	cost function
ground state	$\longleftrightarrow$	optimum solutions
temperature	$\longleftrightarrow$	control parameter (temperature)
rapid quenching	$\longleftrightarrow$	iterative improvement

SA — high level procedure

- generate some neighboring solution $s' \in \mathcal{N}(s)$
- if $f(s') \leq f(s)$, then accept s'
- if $f(s') > f(s)$, then a probabilistic yes/no decision is made
  - if outcome is yes, then s' replaces s
  - if outcome is no, s is kept
- probabilisitic decision depends on
  - the difference $f(s') - f(s)$
  - a control parameter T, called temperature

Simulated Annealing — Procedural view

procedure *Simulated Annealing*

$n \leftarrow 0$; set initial temperature T_0

$s \leftarrow$ GenerateInitialSolution; $s_{best} \leftarrow s$

while outer loop criterion **do**

while inner loop criterion **do**

$s' \leftarrow \mathsf{GenerateSolution}(s)$

$s \leftarrow \mathsf{AcceptanceCriterion}(T_n, s, s')$

if $(f(s) < f(s_{best}))$

$s_{best} \leftarrow s$

end

$T_{n+1} \leftarrow \mathsf{UpdateTemp}(T_n)$, $n \leftarrow n + 1$

end

return s_{best}

end

SA — general issues

- generation of neighboring solutions
  - often: generate a random neighboring solution $s' \in \mathcal{N}(s)$
  - possibly better: systematically generate neighboring solutions
    $\rightsquigarrow$ at least one is sure to sample the whole neighbourhood if no move is accepted
- acceptance criterion
  - often used: Metropolis acceptance criterion
    - if $f(s') \leq f(s)$ then accept s'
    - if $f(s') > f(s)$ then accept it with a probability of

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

SA — cooling schedule

- open questions
  - how to define the control parameter?
  - how to define the (inner and outer) loop criteria?
- cooling schedule
  - initial temperature T_0

    (Example: base it on some statistics about cost values, acceptance ratios etc.)
  - temperature function — how to change the temperature

    (Example: geometric cooling, $T_{n+1} = \alpha \cdot T_n, n = 0, 1, \ldots, 0 < \alpha < 1$)
  - number of steps at each temperature (inner loop criterion)

    (Example: multiple of the neighbourhood size)
  - termination criterion (outer loop criterion)

    (Example: no improvement of s_{best} for a number of temperature values and acceptance rate below some critical value)

Simulated Annealing — Theory

- consider a variant of SA where
  - the temperature is fixed to $T > 0$
  - the number of steps is infinite
  - neighboring solutions are drawn randomly
- model this algorithm as a (homogeneous) Markov chain
- a Markov chain is a stochastic process, in which transition probabilities only depend on the current state
- probabilities of state transitions can be summarized in a matrix

Transition probabilities (1)

- how to compute the transition probabilities?
- decompose transition probability from s_i to $s_j \in \mathcal{N}(s_i)$ into
  - perturbation probability

$$(1) \qquad p_{ij} = \begin{cases} \frac{1}{|\mathcal{N}(s_i)|}, & \text{if } s_j \in \mathcal{N}(s_i); \\ \\ 0, & \text{otherwise;} \end{cases}$$

  - acceptance probability

$$(2) \quad q_{ij} = \begin{cases} 1, & \text{if } f(s_j) \leq f(s_i); \\ \\ \exp\left(-\frac{f(s_j)-f(s_i)}{T}\right) & \text{otherwise;} \end{cases}$$

Transition probabilities (2)

- the transition probability between two solutions s_i and s_j can be computed as

$$\theta_{ij}(T) = \begin{cases} \frac{1}{|\mathcal{N}(s_i)|} & \text{if } f(s_j) \leq f(s_i), s_j \in \mathcal{N}(s \\ \frac{1}{|\mathcal{N}(s_i)|} \cdot \exp\left(-\frac{f(s_j)-f(s_i)}{T}\right) & \text{if } f(s_j) > f(s_i), s_j \in \mathcal{N}(s \\ 1 - \sum_{k,k\neq i} p_{ik} q_{ik} & \text{if } i = j; \\ 0 & \text{otherwise;} \end{cases}$$

- under some mild assumptions on the neighborhood structure, the resulting Markov chain is ergodic

Limiting state distribution

- let $\pi_{Tk}(s_i)$ be the probability that s_i is the current solution after k steps of the algorithm at temperature T
- state probability vector: $\pi_{Tk} = (\pi_{Tk}(s_1), \ldots \pi_{Tk}(s_i), \ldots)$
- for ergodic Markov chains, the state probability vector converges to a limiting probability vector

$$\lim_{k \mapsto \infty} \pi_{Tk} = \pi_T$$

- in particular, one can proof that

$$\lim_{k \mapsto \infty} \pi_{Tk}(s_i) = \frac{\exp(-f(s_i)/T)}{\sum_{s_j \in \mathcal{S}} \exp(-f(s_j)/T)}$$

(Boltzmann distribution)

Limiting distribution for $T \mapsto 0$

- consider two solutions s_i and s_j with $f(s_i) < f(s_j)$
- in this case we have

$$\frac{\pi_{Tk}(s_i)}{\pi_{Tk}(s_j)} \overset{k \mapsto \infty}{\longrightarrow} \frac{\exp(-f(s_i)/T)}{\exp(-f(s_j)/T)}$$

$$= \exp\left(\frac{f(s_j) - f(s_i)}{T}\right) \overset{T \searrow 0}{\longrightarrow} \infty$$

- the last assertion is due to the assumption $f(s_j) - f(s_i) > 0$
- since $\pi_{Tk}(s_i)$ is a probability, we have $\pi_{Tk}(s_i) \leq 1$
- convergence to ∞ is only possible if we have

$$\lim_{k \mapsto \infty} \lim_{T \searrow 0} \pi_{Tk}(s_j) = 0$$

Limiting distribution

- hence, we have proved that
  for a feasible solution s, $k \mapsto \infty$, and $T \searrow 0$ the probability $\pi_{Tk}(s)$ converges to 0, if s is not an optimal solution:

$$\lim_{k \mapsto \infty} \lim_{T \searrow 0} \pi_{Tk}(s) = 0$$

- additionally one can prove that if s is an optimal solution, then we have

$$\lim_{k \mapsto \infty} \lim_{T \searrow 0} \pi_{Tk}(s) = \frac{1}{|\mathcal{S}_{\text{opt}}|}$$

  where $\mathcal{S}_{\text{opt}}$ is the set of all optimal solutions

Observations

- if SA can be run long enough
  - with an infinite number of temperature values and
  - for each temperature value with an infinite number of steps

  one can be sure to be at an optimal solution at the end
- however, it is not clear what end means
- in addition, when run at a single temperature level long enough
  - we can be sure to find the optimum solution
  - hence, an optimal solution can be found without annealing
  - one only needs to store the best solution found and return it at the end

  BUT: we need $k \mapsto \infty$ to guarantee optimality

What do the proofs say?

- from the proofs we can also conclude that
  - better solutions are becoming more likely
- this gives evidence that after
  - a sufficient number of temperature values and
  - a sufficient number of steps at each temperature value

  chances are high to have seen a good solution
- however, it is unclear what sufficient means

remark: stronger results than the ones presented before are available. See Hajek's article from 1988

SA example: TSP (1) *Johnson, McGeoch 1997*

- simple implementation
  - start from a random initial solution
  - neighborhood: `2-opt`
  - simple cooling schedule
    - T_0 is chosen such that ca. 3% of the moves are rejected
    - geometric cooling with $\alpha = 0.95$
    - temperature length $n(n-1)$
    - outer loop criterion: 5 temperature values without improvement and acceptance rate below 2%

    $\rightsquigarrow$ relatively poor results (worse than `3-opt` at 300 times higher computation times)

SA example: TSP (2) *Johnson, McGeoch 1997*

- Improvements
  - look-up table for acceptance probabilities
  - restriction of the neighborhood to small candidate sets using nearest neighbor lists of length 20
  - good initial solution
  - low temperature starts
  - systematic scan of the neighborhood
  - inclusion of `3-opt` moves

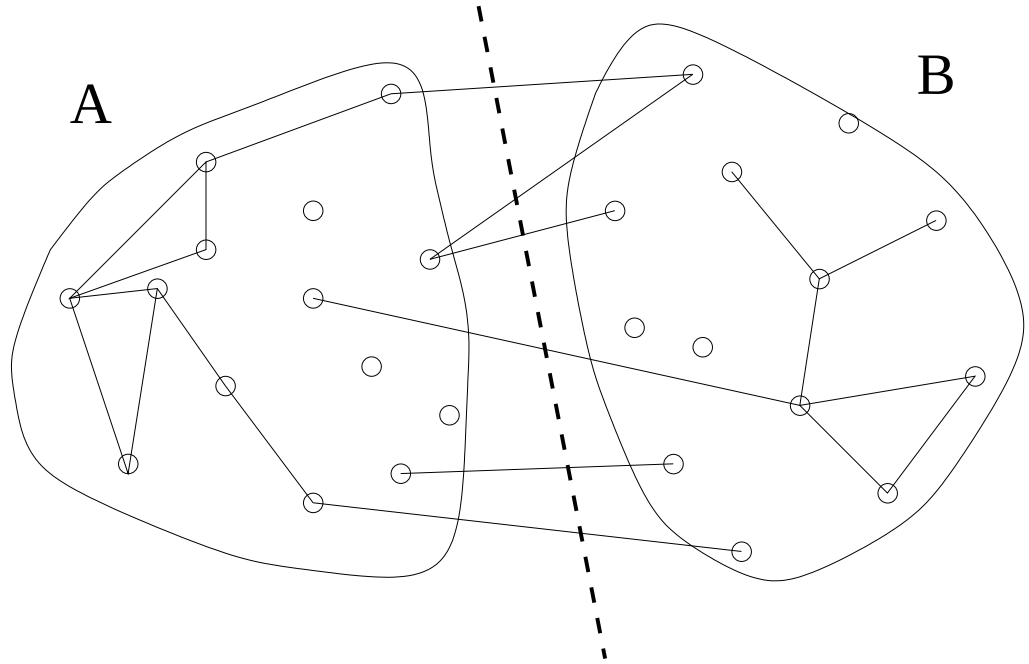
  $\rightsquigarrow$ significantly improved results, comparable to random-restart `LK` for same computation time
- comparison to other techniques

  $\rightsquigarrow$ SA quite far behind state-of-the-art of TSP solving

Graph bipartitioning

Given A graph $G = (V, E)$.

Goal Find a partition of the graph in two node sets V_1 and V_2 with $|V_1| = |V_2|$ and $V_1 \cup V_2 = V$, such that the number of edges with endnodes in the two different sets is minimized.



SA example: graph bipartitioning *Johnson et al. 1989*

- tests were run on random graphs ($G_{n,p}$) and random geometric graphs $U_{n,d}$
- modified cost function (α: imbalance factor)

$$f(V_1, V_2) = |\{(u, v) \in E \mid u \in V_1 \wedge v \in V_2\}| + \alpha(|V_1| - |V_2|)^2$$

$\rightsquigarrow$ allows infeasible solutions but punishes the amount of infeasibility

- side advantage: allows to use 1–exchange neighborhoods of size $\mathcal{O}(n)$ instead of the typical neighborhood that exchanges two nodes at a time and is of size $\mathcal{O}(n^2)$

SA example: graph bipartitioning *Johnson et al. 1989*

- initial solution is chosen randomly
- standard geometric cooling schedule
- experimental comparison to Kernighan–Lin heuristic
  - Simulated Annealing gave better performance on $G_{n,p}$ graphs
  - just the opposite is true for $U_{n,d}$ graphs
- several further improvements were proposed and tested

general remark: Although relatively old, Johnson et al.'s experimental investigations on SA are still worth a detailed reading!

SA example: course timetabling

- abstraction of a real course timetabling problem studied in the metaheuristics network
- problem
  - given is a set of events visited by a set of students
  - goal: assign events to timeslots and rooms subject to *hard* constraints and optimization 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 timeslot, there is at most one event in a room

SA example: course timetabling

- optimization criteria through *soft* constraints
  - student has event in last slot of a day
  - student has more than two events in a row
  - student has a single class on a day
- soft constraint violations are penalized
- objective
  - find a feasible solution with minimum number of soft constraint violations

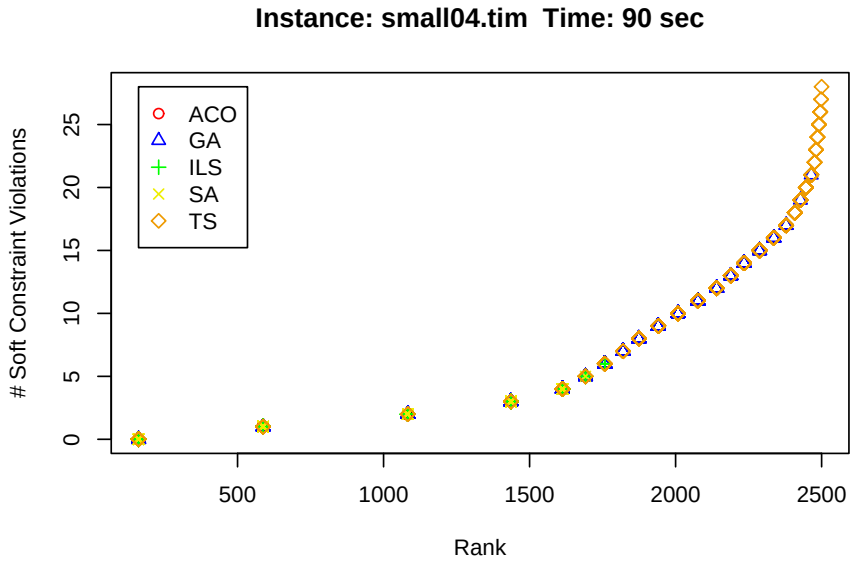
SA example: course timetabling

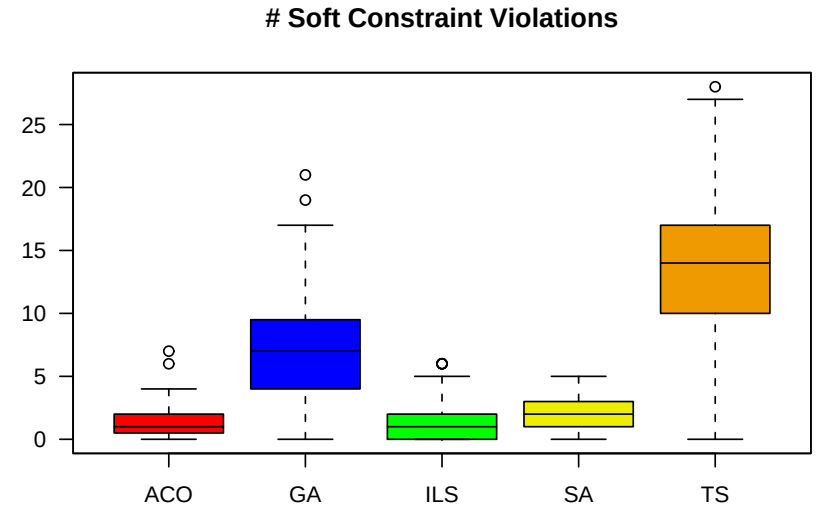
- this problem was attacked in the Metaheuristics Network
- and is part of the International Timetabling Competition!!
- implemented SLS methods
  - Ant Colony Optimization
  - Iterated Local Search
  - Simulated Annealing
  - Tabu Search
  - Evolutionary Algorithms
- all the SLS methods were implemented by the expert labs in the metaheuristics network
- and extensively evaluated on a set of benchmark problems (results courtesy of Michael Sampels)

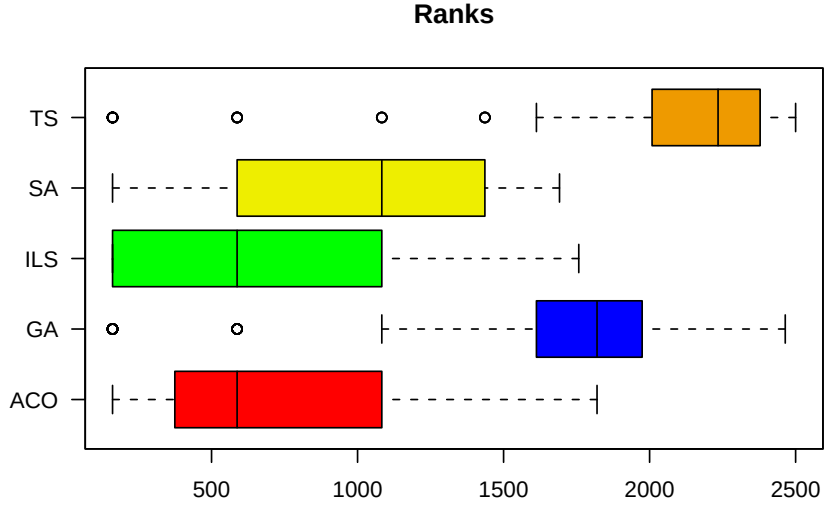
Course timetabling:

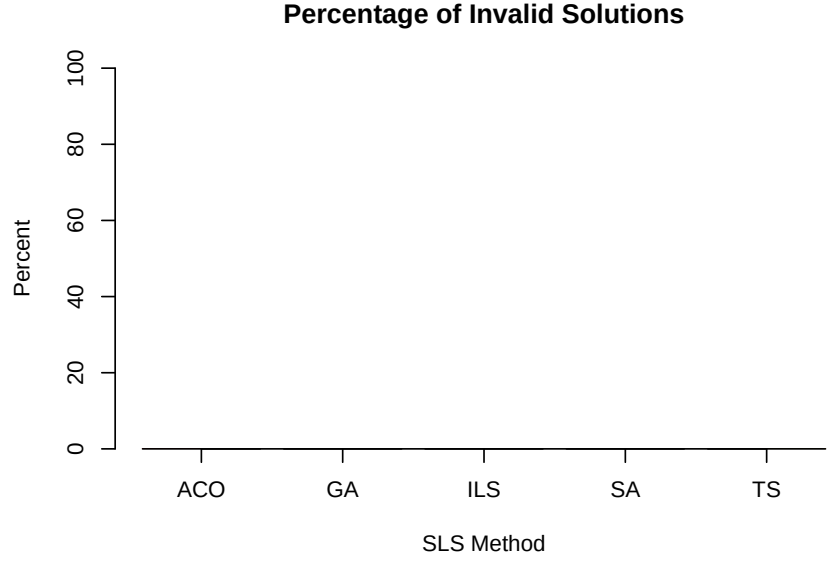
- implementations were done in two phases
- first phase
  - all labs were given a same local search
  - all labs were given one month of development time
  - then all algorithms had to be submitted and were evaluated
- here: results of this first phase
- second phase: more in depth studies and further developments
- the computational results were analyzed with non-parametric statistical tests based on ranks

Results: Small size instance

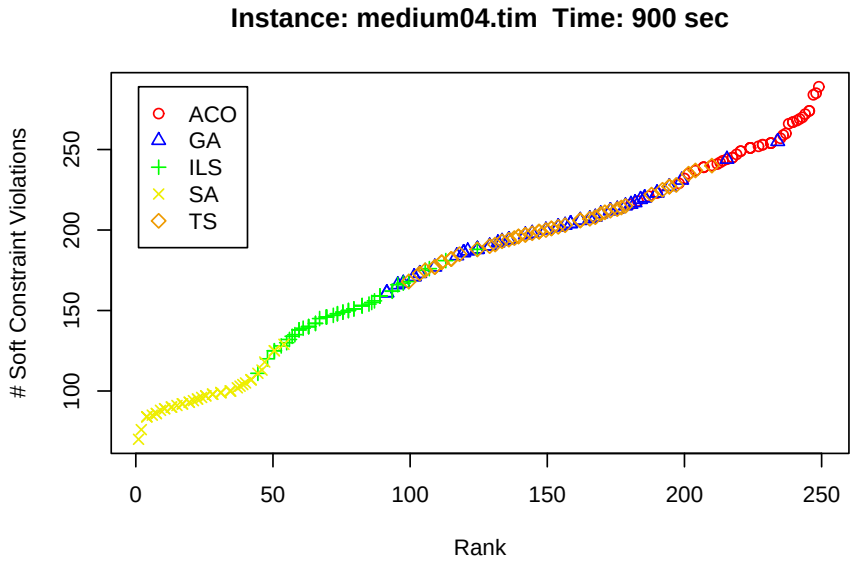


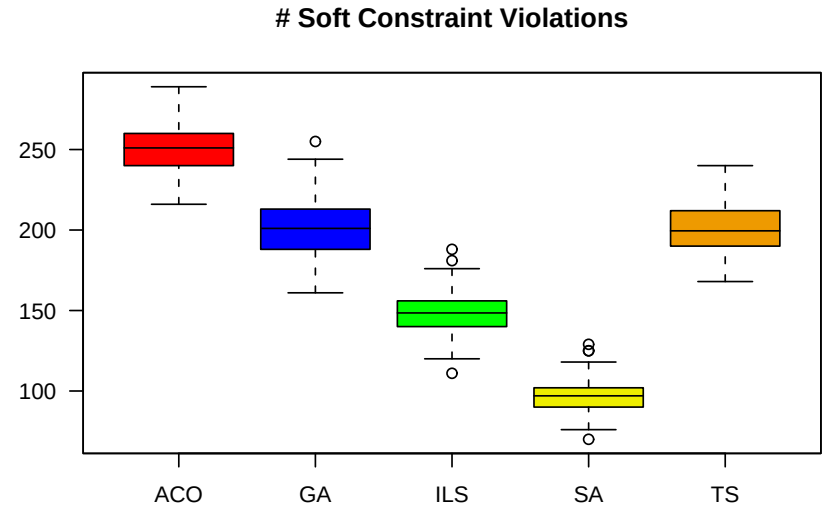


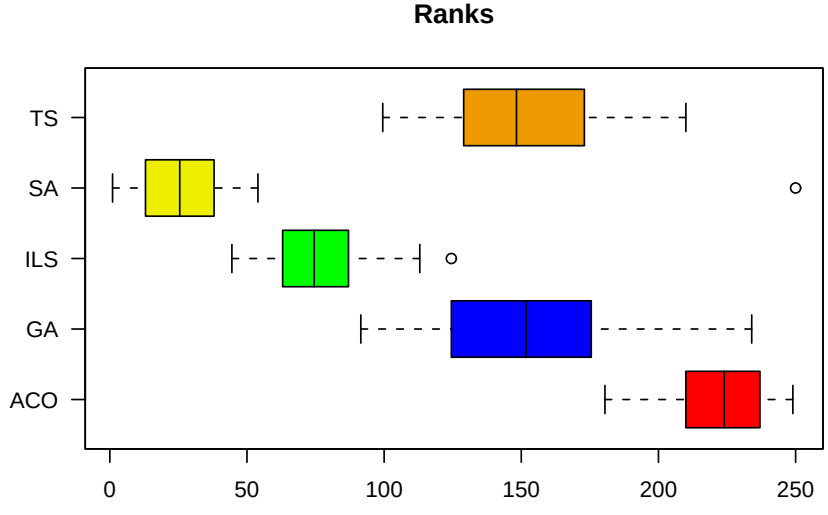


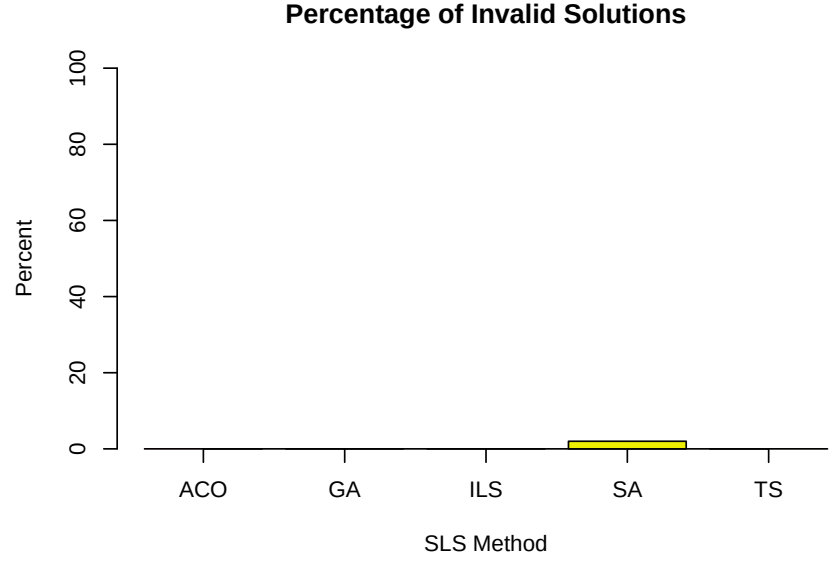


Medium size instance

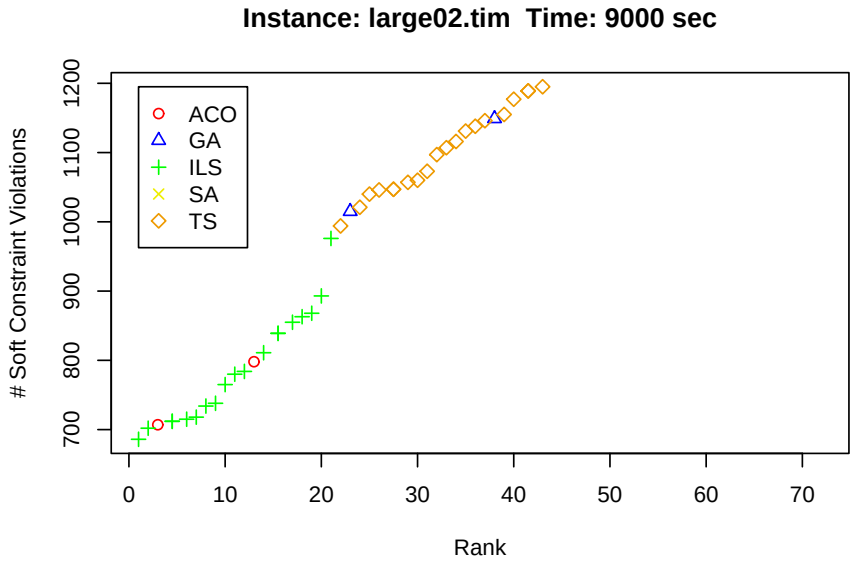


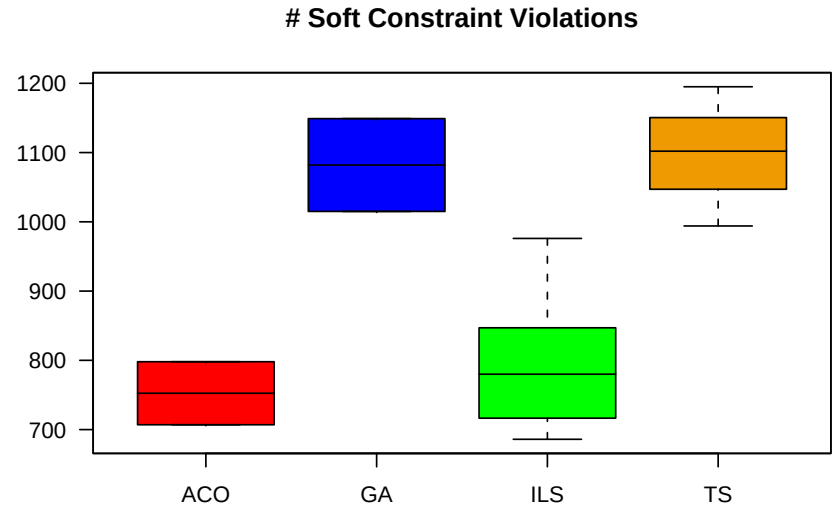


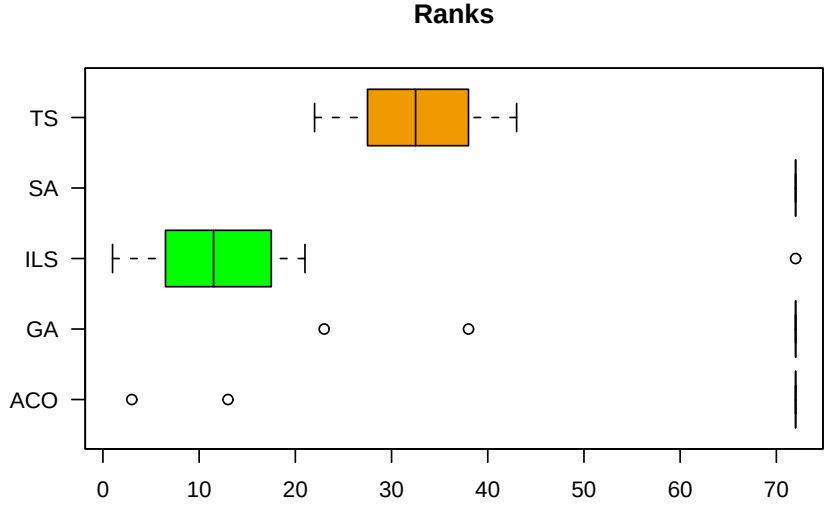


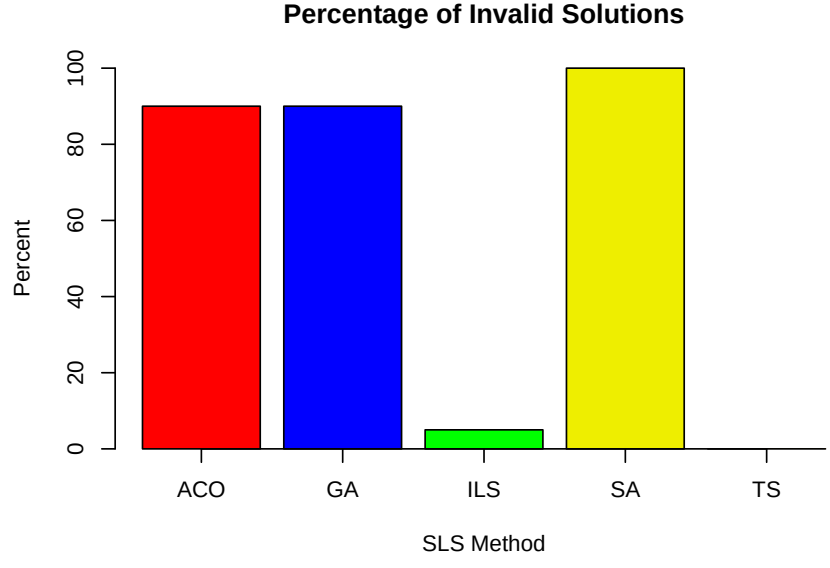


Large size instance









Additional issues . . . not covered here

- Non-monotone cooling schedules
- parallelization
- inhomogeneous theory
- results on speed of convergence
- optimal cooling schedules
- related approaches (threshold accepting etc.)

Summary

- Simulated Annealing is historically one of the first SLS methods
- very easy to implement
- interesting for
  - practitioners: short development times
  - mathematicians: convergence
- good results but often at the cost of substantial computation times

Literature

• E.H.L. Aarts, J.H.M. Korst, and P.J.M. van Laarhoven. Simulated Annealing. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 91–120. John Wiley & Sons, 1997.

• V. Cerný. A Thermodynamical Approach to the Traveling Salesman Problem. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

• B. Hajek. Cooling Schedules for Optimal Annealing. *Mathematics of OR*, 13:311–329, 1988.

• D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by Simulated Annealing: An Experimental Evaluation: Part I, Graph Partitioning. *Operations Research*, 37(6):865–892, 1989.

• D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by Simulated Annealing: An Experimental Evaluation: Part II, Graph Coloring and Number Partitioning. *Operations Research*, 39(3):378–406, 1991.

• D.S. Johnson and L.A. McGeoch. The Travelling Salesman Problem: A Case Study in Local Optimization. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, 1997.

• S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

• F. Romeo and A. Sangiovanni-Vincentelli. A Theoretical Framework for Simulated Annealing. *Algorithmica*, 6:302–345, 1991.

• W.M. Spears. Simulated Annealing for Hard Satisfiability Problems. Technical report, Naval Research Laboratory, Washington D.C., 1993.

What is dynamic local search?

Dynamic local search is a collective term for a number of approaches that try to escape local optima by iteratively modifying the evaluation function value of solutions.

- different concept for escaping local optima
- several variants available
- promising results

Dynamic local search

- guide the local search by a dynamic evaluation function
- evaluation function $h(s)$ composed of
  - cost function $f(s)$
  - penalty function
- penalty function is adapted at computation time to guide the local search
- penalties are associated to solution features
- related approaches
  - long term strategies in tabu search
  - noising method
  - usage of time-varying penalty functions for (strongly) constrained problems
  - etc.

Issues in dynamic local search

- timing of penalty modifications
  - at every local search step
  - only when trapped in a local optimum w.r.t. h
  - long term strategies for weight decay
- strength of penalty modifications
  - additive vs. multiplicative penalty modifications
  - amount of penalty modifications
- focus of penalty modifications
  - choice of solution attributes to be punished

Example: Guided Local Search

PhD thesis Voudouris; Voudouris, Tsang, Mills 1995 – . . .

Guided local search

- guided local search (GLS)
  - modifies penalties when trapped in local optima of h
  - variants exist that use occasional penalty decay
  - uses additive penalty modifications
  - chooses few among the many solution components for punishment

Guided local search — Procedural view

procedure *Guided Local Search*

$s \leftarrow$ GenerateInitialSolution

InitializePenalties

while (termination condition not met) **do**

$h \leftarrow$ ComputeAugmentedObjectiveFunction

$\hat{s} \leftarrow$ LocalSearch$(\hat{s}, h, f)$

UpdatePenalties$(\hat{s})$

end while

return s_{best}

end procedure

Attention: to get s_{best}, check in LocalSearch solutions also w.r.t. the cost function f

GLS — details

- penalties are associated to solution attributes
  - cost contribution $f_i(s)$ for solution attribute i
  - penalty costs p_i for solution attribute i
  - an indicator function $I_i(s)$ says whether solution attribute i occurs in solution s
- evaluation function $h(s)$ becomes

$$h(s) = f(s) + \lambda \cdot \sum_{i=1}^{M} p_i \cdot I_i(s)$$

M: number of solution attributes
λ: determines the influence of the penalty costs

Guided local search — details

- LocalSearch
  - uses $h(s)$ for evaluating solutions
  - runs until stuck in a local optimum $\hat{s}$ w.r.t. h
  - once stuck, penalties are modified
- modification of penalties
  - define the utility of solution attributes as

$$Util(\hat{s}, i) = \frac{f_i(\hat{s})}{1 + p_i}$$

  - for all solution attributes with maximum utility set $p_i \leftarrow p_i + 1$

Propositional Satisfiability Problem (SAT)

- Simple SAT instance (in CNF):

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

- 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 – model finding variant: For a given propositional formula Φ, if Φ is satisfiable, find a model, otherwise declare Φ unsatisfiable.

GLS example: SAT/MAX-SAT *Mills, Tsang, 2000*

- best-improvement 1–opt local search (GSAT architecture)
- uses in additions a special tie-breaking criterion that favors flipping a variable that was flipped the longest time ago (taken from HSAT)
- if in x consecutive iterations no improved solution is found, then modify penalties
- solution attributes are clauses
- when trapped in a local optimum, add penalties to clauses of maximum utility

GLS example: SAT/MAX-SAT

- computational experience
  - good results especially for very hard SAT instances
  - currently one of the best available algorithms for weighted MAX-SAT
- further applications
  - TSP
  - QAP
  - Vehicle Routing
  - Constraint Satisfaction Probleme
  - workforce scheduling

Literature

• A. Davenport, E. Tsang, C.J. Wang, and K. Zhu. GENET: A Connectionist Architecture for Solving Constraint Satisfaction Problems by Iterative Improvement. In *Proceedings of the 14th National Conference on Artificial Intelligence*. MIT press, 1994.

• F Hutter, D. Tompkins, and H.H. Hoos. Scaling and Probabilistic Smoothing: Efficient Dynamic Local Search for SAT. In *Proc. CP'02*, 2002.

• P. Mills and E. Tsang. Guided local search for solving SAT and weighted MAX-SAT problems. In I. Gent, H. van Maaren, and T. Walsh, editors, *SAT'2000*. IOS Press, pp. 89–106, 2000.

• P. Morris. The Breakout Method for Escaping from Local Minima. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 40–45. MIT press, 1993.

• D. Schuurmans, and F. Southey, and R.C. Holte. The Exponentiated Subgradient Algorithm for Heuristic Boolean Programming. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 334–341. Morgan Kaufmann, San Francisco, USA.

• C. Voudouris, and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113, pages 469–499, 1999.

• Z. Wu and W. Wah. An efficient global-search strategy in discrete lagrangian methods for solving hard satisfiability problems. In *Proceedings of the 17th National Conference on Artificial Intelligence*, pages 310–315. AAAI Press / The MIT Press, Menlo Park, CA, USA, 2000.

What is GRASP?

Greedy Randomized Adaptive Search Procedures (GRASP) is an SLS method that tries to construct a large variety of good initial solutions for a local search algorithm.

- predecessors: semi-greedy heuristics
- tries to combine the advantages of random and greedy solution construction

Greedy construction heuristics

- iteratively construct solutions by choosing at each construction step one solution component
  - solution components are rated according to a greedy function
  - the best ranked solutions component is added to the current partial solution
- examples: Kruskal's algorithms for minimum spanning trees, greedy heuristic for the TSP, . . .
- advantage: generate good quality solutions; local search runs fast and finds typically better solutions than from random initial solutions
- disadvantage: do not generate many different solutions; difficulty of iterating

Random vs. greedy construction

- random construction
  - high solution quality variance
  - low solution quality
- greedy construction
  - good quality
  - low (no) variance
- goal: exploit advantages of both

Semi-greedy heuristics

- add at each step not necessarily the highest rated solution
- repeat until a full solution is constructed:
  - rate solution components according to a greedy function
  - put high rated solution components into a *restricted candidate list* (RCL)
  - choose one element of the RCL randomly and add it to the partial solution
  - adaptive element: greedy function depends on the partial solution constructed so far

Hart, Shogan, 1987

Generation of the RCL

- mechanisms for generating RCL
  - cardinality based: include the k best rated solution components into RCL
  - value based: include all solution components with greedy values better than a given threshold
- $\min \max \alpha$ based RCL
  - let f_{min} (f_{max}) be greedy values of best (worst) ranked solution component
  - include solution components e with greedy values

$$f(e) \leq f_{min} + \alpha \cdot (f_{max} - f_{min})$$

  $\alpha \in [0, 1]$ is a parameter
  - $\alpha = 0$ corresponds to a greedy construction heuristic
  - $\alpha = 1$ corresponds to a random solution construction

GRASP

- GRASP tries to capture advantages of random and greedy solution construction
- iterate through
  - randomized solution construction exploiting a greedy probabilistic bias to construct feasible solutions
  - apply local search to improve over the constructed solution
- keep track of the best solution found so far and return it at the end

GRASP — local search

- local search from random solutions
  - high variance
  - best solution quality often better than greedy (if not too large instances)
  - average solution quality worse than greedy
  - local search requires many improvement steps
- local search from greedy solutions
  - average solution quality better than random
  - local search typically requires only a few improvement steps
  - low (no) variance

GRASP — Procedural view

procedure *GRASP*
  Initialize Parameter
  while (termination condition not met) **do**
    $s =$ ConstructGreedyRandomizedSolution$()$
    $s' =$ LocalSearch(s)
    if $f(s') < f(s_{best})$
      $s_{best} = s'$
  end
  return s_{best}
end *GRASP*

GRASP Example: SAT

- solution components are value assignment to variables
- greedy-Function
  - number of still unsatisfied clauses that would become satisfied by a value assignment
  - Φ_i^+: set of additionally satisfied clauses if $x_i = \texttt{true}$
  - Φ_i^-: set of additionally satisfied clauses if $x_i = \texttt{false}$
- $\min \max \alpha$ based RCL
  - Let $\Phi^* = \max\{|\Phi_i^+|, |\Phi_i^-|\}$ over all free variables x_i
  - $x_i = \texttt{true} \in$ RCL if $|\Phi_i^+| \geq \alpha \cdot \Phi^*$
    $x_i = \texttt{false} \in$ RCL if $|\Phi_i^-| \geq \alpha \cdot \Phi^*$

GRASP Example: SAT

- variable selection
  - if an unsatisfied clause contains only one single still uninstantiated variable, try to satisfy this clause
  - otherwise choose randomly an element from the RCL
- local search
  - best-improvement 1-opt local search (GSAT architecture)
- performance
  - at the time the research was done reasonably good performance
  - however, nowadays by far outperformed by more recent local search algorithms for SAT
  - the same is true for weighted MAX-SAT

GRASP extensions

- convergence of GRASP (not guaranteed if $\alpha \neq 1$)
- introduction of a bias when choosing elements from the RCL
  - different possibilities of using, e.g. ranks (e.g. $\text{bias}(r) = 1/r$)
  - choose a solution component with a probability proportional to bias
- reactive GRASP (tuning of α)
- addition of a type of long term memory to bias search
  - path relinking
  - use of previous elite solutions to guide construction
- parallelization of GRASP

GRASP — concluding remarks

- straightforward extension of construction heuristics
- easy to implement
- few parameters
- many different applications available
- several extensions exist
- can be used to generate initial population in population-based methods
- however, as a stand-alone procedure often not state-of-the-art results

Literature

• T.A. Feo and M.G.C. Resende. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operations Research Letters*, 8:67–71, 1989.

• P. Festa and M. G. C. Resende. GRASP: An annotated bibliography. In P. Hansen and C. C. Ribeiro, editors, *Essays and Surveys on Metaheuristics*. Kluwer Academic Publishers, 2001.

• J. P. Hart and A. W. Shogan. Semi-greedy Heuristics: An Empirical Study. *Operations Research Letters*, 6:107–114, 1987.

• M.G.C. Resende, T.A. Feo. A GRASP for Satisfiability. In D.S. Johnson and M.A. Trick, editors. *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*. pages 499–520. American Mathematical Society, 1996.

• M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Approximate solution of weighted MAX-SAT problems using GRASP. In J. Gu and P.M. Pardalos, editors, *Satisfiability problems*, volume 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 393–405. American Mathematical Society, 1997.

• M. G. C. Resende and C. C. Ribeiro, Greedy randomized adaptive search procedures, in Handbook of Metaheuristics, F. Glover and G. Kochenberger, eds., Kluwer Academic Publishers, pp. 219-249, 2002

What is Iterated Greedy?

Iterated Greedy is an SLS method that builds a sequence of solutions by iterating over greedy construction heuristics through destruction and construction phases.

- straightforward extension of iterated local search to the context of greedy construction heuristics
- very good results in a variety of applications

Greedy — procedural view

procedure *Greedy Construction Heuristic*
  $s_p = \textit{empty solution}$
  while s_p is not a complete solution s **do**
    $c =$ GreedyComponent(s_p)
    $s_p = s_p \otimes c$
  end while
  $s = s_p$
  return s
end procedure

Greedy construction heuristics

- give seed solutions to local search / EAs etc.
- sometimes additional features applied
  - use look-ahead
  - use local search on partial solutions
- construction heuristics also used inside several SLS methods like ACO, rollout/piloting method, GRASP
- different approach:
  - destruct part of the solution
  - reconstruct a full solution
  - iterate through these two phases

$\rightsquigarrow$ iterated greedy (IG)

IG — procedural view

procedure *Simple Iterated Greedy*
  $s =$ GenerateInitialSolution
  repeat
    $s_p =$ DestructionPhase(s)
    $s^{'} =$ ConstructionPhase(s_p)
    $s =$ AcceptanceCriterion$(s, s^{'})$
  until termination condition met
end

closely related to iterated local search but using as an underlying heuristic

a greedy construction one

IG — algorithm

- destruction phase
  - fixed vs. variable size of destruction
  - stochastic vs. deterministic destruction
  - uniform vs. biased destruction
- construction phase
  - not every construction heuristic is trivially applicable

    e.g. nearest neighbor construction heuristic for TSP would need some adaptations
  - typically, adaptive construction heuristics preferable
  - speed of the construction heuristic is an issue
- acceptance criterion
  - very much the same issue as in ILS

IG — enhancements

- usage of history information to bias destructive/constructive phase
- use lower bounds on the completion of a solution in the constructive phase
- combination with local search in the constructive phase
- use local search to improve full solutions
  $\rightsquigarrow$ destruction / construction phases can be seen as a perturbation mechanism in ILS
- exploitation of constraint propagation techniques

IG example: Set covering

- given:
  - finite set $\mathbf{A} = \{a_1, \ldots, a_m\}$ of objects
  - family $\mathbf{B} = \{B_1, \ldots B_n\}$ of subsets of $\mathbf{A}$ that covers $\mathbf{A}$
  - weight function $w : \mathbf{B} \mapsto \mathbb{R}^+$
- $\mathbf{C} \subseteq \mathbf{B}$ *covers* $\mathbf{A}$ if every element in $\mathbf{A}$ appears in at least one set in $\mathbf{C}$, i.e. if $\bigcup \mathbf{C} = \mathbf{A}$
- goal: find a subset $\mathbf{C}^* \subseteq \mathbf{B}$ of minimum total weight that covers $\mathbf{A}$.
- interest: arises in many applications, $\mathcal{NP}$-hard

IG example: Set covering

- IG approach by Brusco and Jacobs from 1995
- assumption: all subsets are ordered according to nondecreasing costs
- construct initial solution using a greedy heuristic based on two steps
  - randomly select a uncovered object a_i
  - add the lowest cost subset that covers a_i
- DestructionPhase removes a fixed number of $k_1|\mathbf{C}|$ subsets; k_1 is a parameter

IG example: Set covering

- ConstructionPhase proceeds as
  - build a candidate set containing subsets with cost of less than $k_2 \cdot f(\mathbf{C})$
  - compute the cover value $\gamma_j = w_j/d_j$
    d_j: number of objects covered when adding subset b_j
  - add a subset with minimum cover value
- complete solution is post-processed by removing redundant subsets
- AcceptanceCriterion: Metropolis acceptance criterion from SA
- computational experience
  - good performance with this simple approach
  - more recent IG variants are state-of-the-art algorithms for SCP

IG — concluding remarks

- simple principle
- analogous extension to greedy heuristics as ILS to local search
- not a very strongly explored SLS method
- provides an additional tool to SLS researchers
- for some applications so far excellent results
- can give place to more effective combinations of tree search and local search heuristics