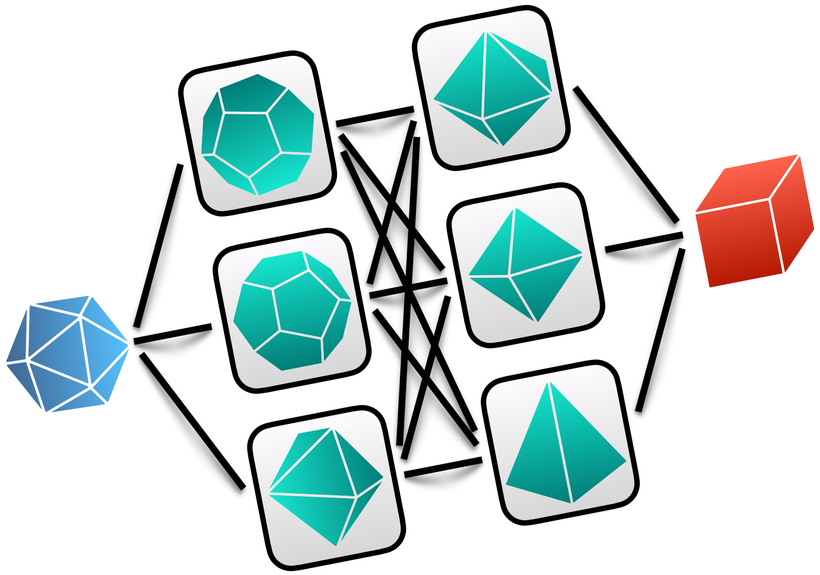
Did I forget to hit record? Please remind me!



Empirical Algorithmics

MIE1666: Machine Learning for Mathematical Optimization

Largely based on the book Stochastic Local Search by Hoos and Stützle and related slides by the authors

Some slides from Algorithm Design by Kleinberg and Tardos

Elias B. Khalil — 27/09/21

UNIVERSITY OF TORONTO

$$
\begin{aligned}
F := & (\neg x_1 \vee x_2) \\
& \wedge (\neg x_2 \vee x_1) \\
& \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \\
& \wedge (x_1 \vee x_2) \\
& \wedge (\neg x_4 \vee x_3) \\
& \wedge (\neg x_5 \vee x_3)
\end{aligned}
$$

$C := \{\top, \bot\}$ *the set of* truth values *(or propositional constants)* true *and* false, *and* $O := \{\neg, \wedge, \vee\}$ *the set of* propositional operators negation *('not')*, conjunction *('and') and* disjunction *('or')*.

EXAMPLE I.1 A Simple SAT Instance

See nice application in Product Configuration: https://www.cs.utexas.edu/~isil/cs389L/lecture4-6up.pdf

Let us consider the following propositional formula in CNF:

$$\begin{aligned} F := & (\neg x_1 \vee x_2) \\ & \wedge (\neg x_2 \vee x_1) \\ & \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \\ & \wedge (x_1 \vee x_2) \\ & \wedge (\neg x_4 \vee x_3) \\ & \wedge (\neg x_5 \vee x_3) \end{aligned}$$

For this formula, we obtain the variable set $Var(F) = \{x_1, x_2, x_3, x_4, x_5\}$; consequently, there are $2^5 = 32$ different variable assignments. Exactly one of these, $x_1 = x_2 = \top, x_3 = x_4 = x_5 = \bot$, is a model, rendering F satisfiable.

$C := \{\top, \bot\}$ *the set of* truth values *(or propositional constants)* true *and* false, *and* $O := \{\neg, \wedge, \vee\}$ *the set of* propositional operators negation *('not')*, conjunction *('and') and* disjunction *('or')*.

Example of a (greedy) construction heuristic for SAT

$$F := (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_4 \vee x_3) \wedge (\neg x_5 \vee x_3)$$

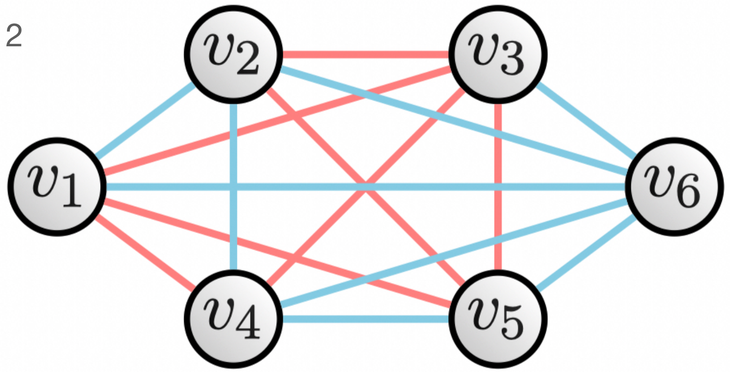
- start with an empty variable assignment
- in each step select an unassigned variable
  and set it to a truth value
  - if $\exists$ unsatisfied clause with only one unassigned variable,
    assign this variable to satisfy this clause
  - otherwise choose variable and truth value such that
    maximal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

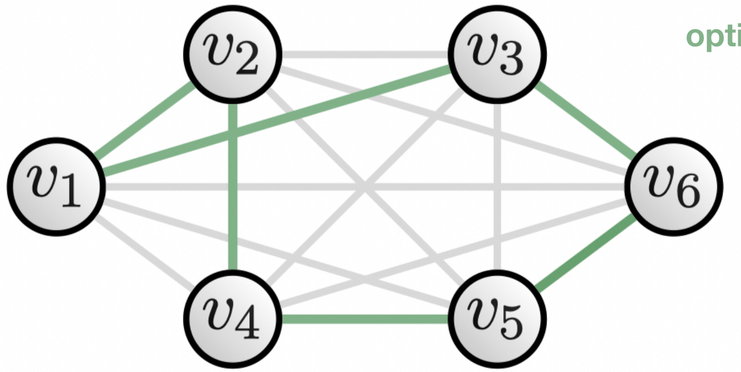
$C := \{\top, \bot\}$ *the set of* truth values *(or propositional constants)* true *and* false, *and* $O := \{\neg, \wedge, \vee\}$ *the set of* propositional operators negation *('not')*, conjunction *('and') and* disjunction *('or')*.

Graph Optimization

cost = 1

cost = 2





optimal solution, cost = 7

Travelling Salesperson Problem (TSP)

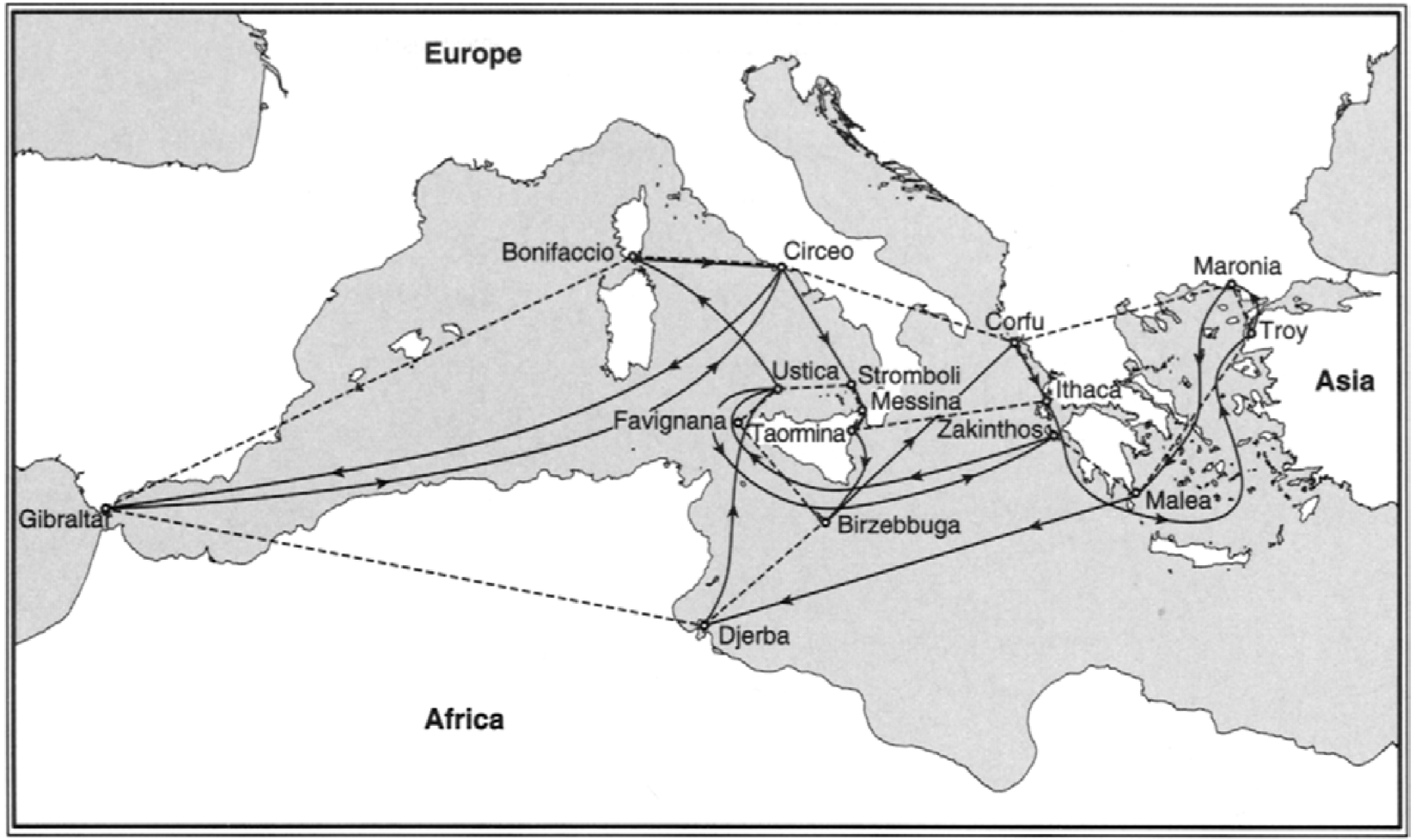
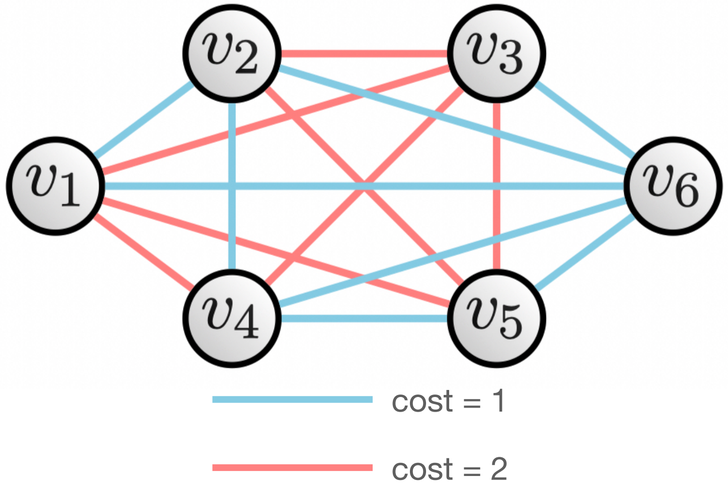


Figure 1.1 A graphic representation of the geographic TSP instance 'ulysses16' and its optimal solution (dashed line); the solid line and arrows indicate the sequence in which Homer's Ulysses supposedly visited the 16 locations. See Example 1.2 for details.

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



https://www.cs.princeton.edu/~wayne/kleinberg-tardos/

Polynomial running time

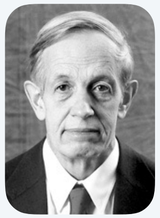
Desirable scaling property. When the input size doubles, the algorithm should slow down by at most some multiplicative constant factor C.

Def. An algorithm is poly-time if the above scaling property holds.

**There exist constants c > 0 and d > 0 such that,
for every input of size n, the running time of the algorithm
is bounded above by c n^d primitive computational steps.**

← corresponds to $C = 2^d$


von Neumann (1953)


Nash (1955)


Gödel (1956)


Cobham (1964)


Edmonds (1965)


Rabin (1966)

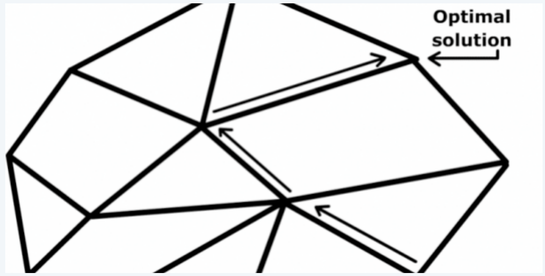
https://www.cs.princeton.edu/~wayne/kleinberg-tardos/

Worst-case analysis

Worst case. Running time guarantee for any input of size n.

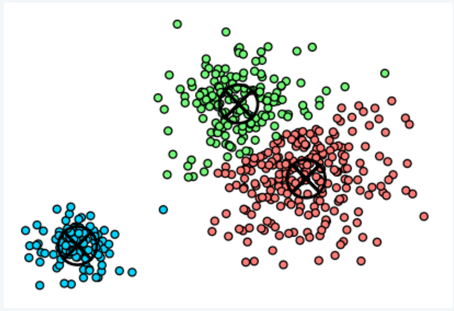
- Generally captures efficiency in practice.
- Draconian view, but hard to find effective alternative.

Exceptions. Some exponential-time algorithms are used widely in practice because the worst-case instances don't arise.



simplex algorithm

Linux grep



k-means algorithm

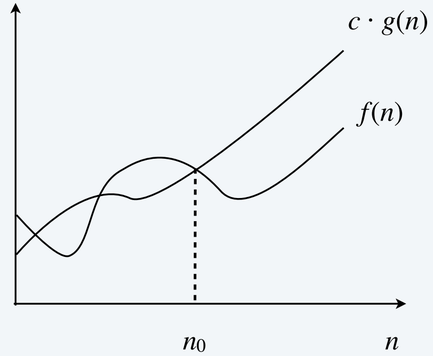
9

Big O notation

Upper bounds. $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

Ex. $f(n) = 32n^2 + 17n + 1$.

- $f(n)$ is $O(n^2)$. ← choose $c = 50, n_0 = 1$
- $f(n)$ is neither $O(n)$ nor $O(n \log n)$.



Typical usage. Insertion sort makes $O(n^2)$ compares to sort n elements.

https://www.cs.princeton.edu/~wayne/kleinberg-tardos/

Analysis of algorithms: quiz 1

Let $f(n) = 3n^2 + 17\, n \log_2 n + 1000$. Which of the following are true?

A. $f(n)$ is $O(n^2)$.

B. $f(n)$ is $O(n^3)$.

C. Both A and B.

D. Neither A nor B.

Polynomial time

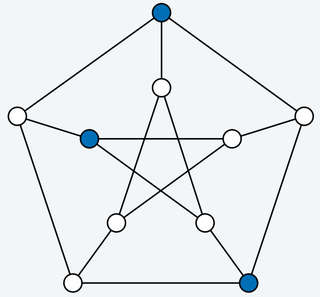
Polynomial time. Running time is $O(n^k)$ for some constant $k > 0$.

Independent set of size k. Given a graph, find k nodes such that no two are joined by an edge.

k is a constant

$O(n^k)$ algorithm. Enumerate all subsets of k nodes.

```
FOREACH subset S of k nodes:
    Check whether S is an independent set.
    IF (S is an independent set)
        RETURN S.
```



independent set of size 3

- Check whether S is an independent set of size k takes $O(k^2)$ time.
- Number of k-element subsets = $\binom{n}{k} = \frac{n(n-1)(n-2) \times \cdots \times (n-k+1)}{k(k-1)(k-2) \times \cdots \times 1} \leq \frac{n^k}{k!}$
- $O(k^2 \, n^k / k!) = O(n^k)$.

poly-time for $k = 17$, but not practical

Exponential time

Exponential time. Running time is $O(2^{n^k})$ for some constant $k > 0$.

Independent set. Given a graph, find independent set of max cardinality.

$O(n^2 2^n)$ algorithm. Enumerate all subsets of n elements.

$S^* \leftarrow \emptyset$.

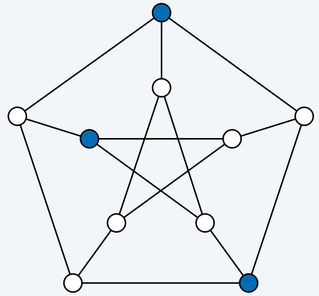
FOREACH subset S of n nodes:

  Check whether S is an independent set.

  IF (S is an independent set and $|S| > |S^*|$)

    $S^* \leftarrow S$.

RETURN S^*.



independent set of max cardinality

Exponential time

Exponential time. Running time is $O(2^{n^k})$ for some constant $k > 0$.

Euclidean TSP. Given n points in the plane, find a tour of minimum length.

$O(n \times n!)$ algorithm. Enumerate all permutations of length n.

$\pi^* \leftarrow \varnothing$.

FOREACH permutation π of n points:

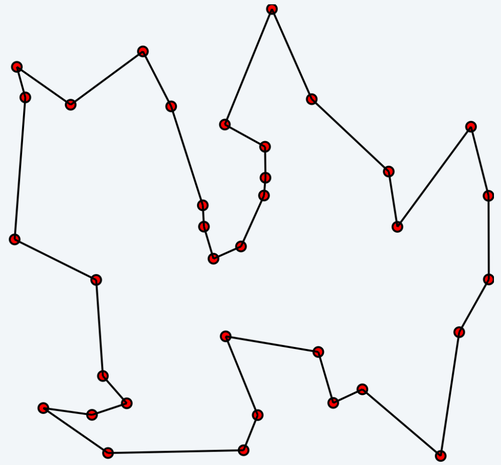
  Compute length of tour corresponding to π.

  IF (length(π) < length(π^*))

    $\pi^* \leftarrow \pi$.

RETURN π^*.

for simplicity, we'll assume Euclidean distances are rounded to nearest integer (to avoid issues with infinite precision)



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

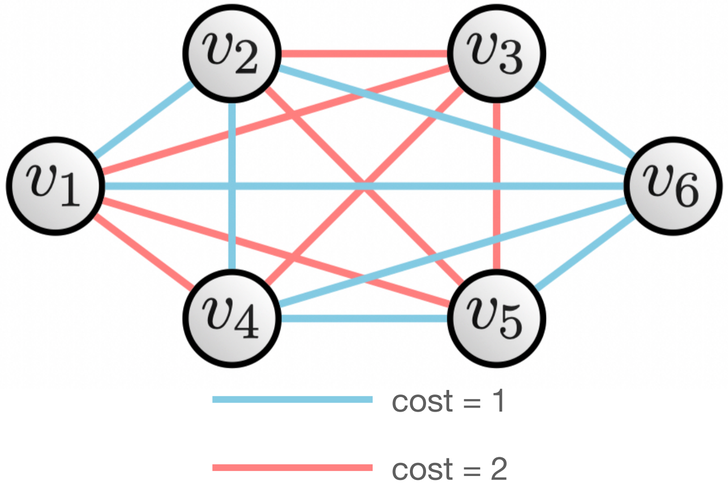
$$\begin{aligned} F := & (\neg x_1 \vee x_2) \\ & \wedge (\neg x_2 \vee x_1) \\ & \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \\ & \wedge (x_1 \vee x_2) \\ & \wedge (\neg x_4 \vee x_3) \\ & \wedge (\neg x_5 \vee x_3) \end{aligned}$$

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

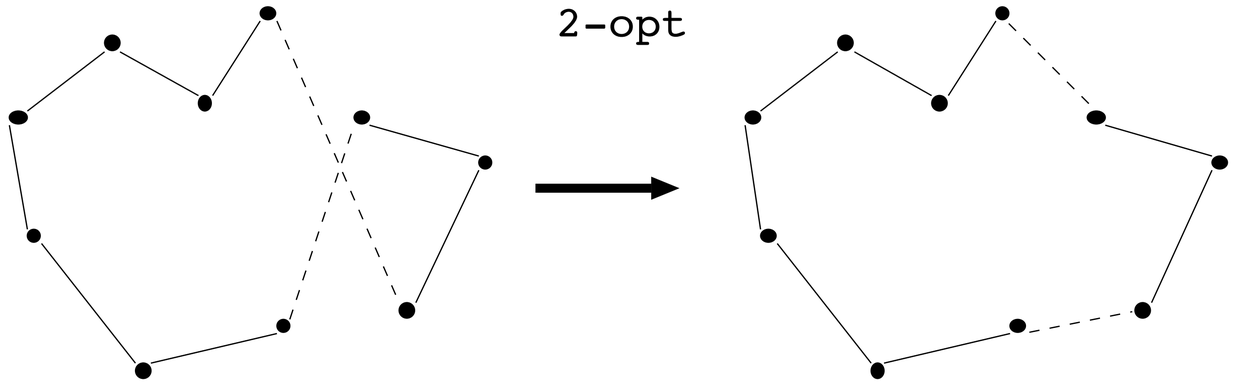
Nearest Neighbour heuristic for the TSP:

- 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 $\{d(\pi(i), j)\}; j \neq \pi(k), 1 \leq k \leq i$



Iterative Improvement for the TSP

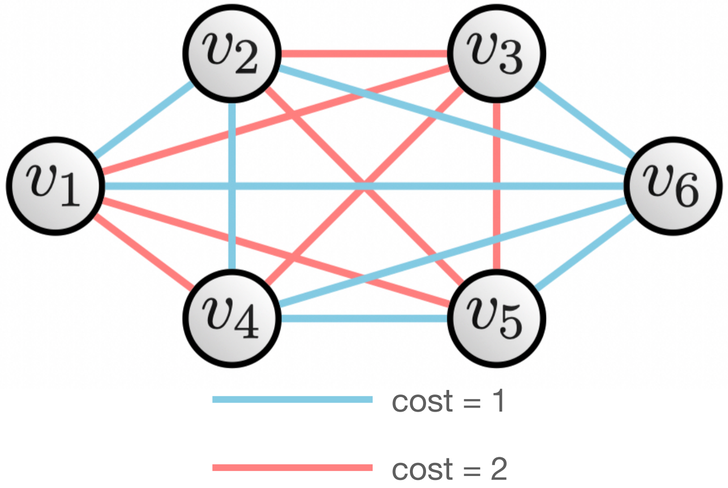
- initial solution is a complete tour
- k-opt neighbourhood: solutions which differ by at most k edges



- neighbourhood size $\mathcal{O}(n^k)$

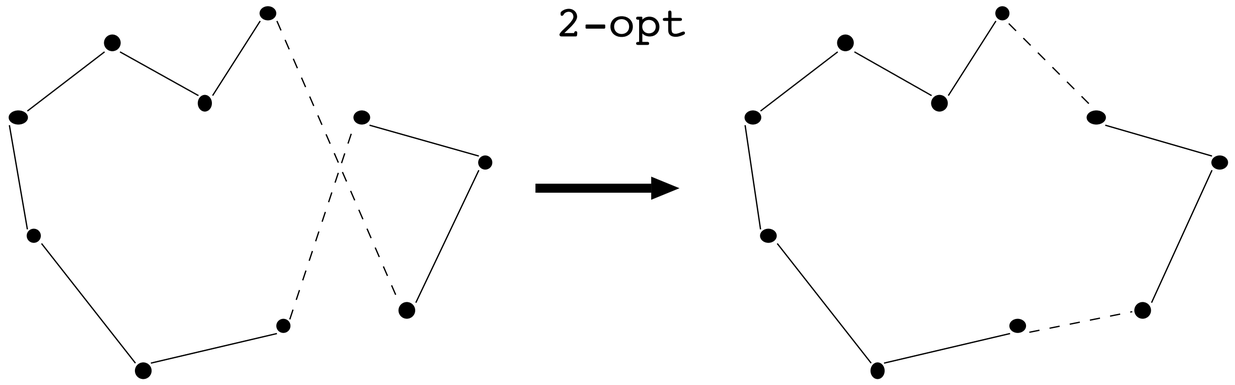
Nearest Neighbour heuristic for the TSP:

- 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 $\{d(\pi(i), j)\}; j \neq \pi(k), 1 \leq k \leq i$



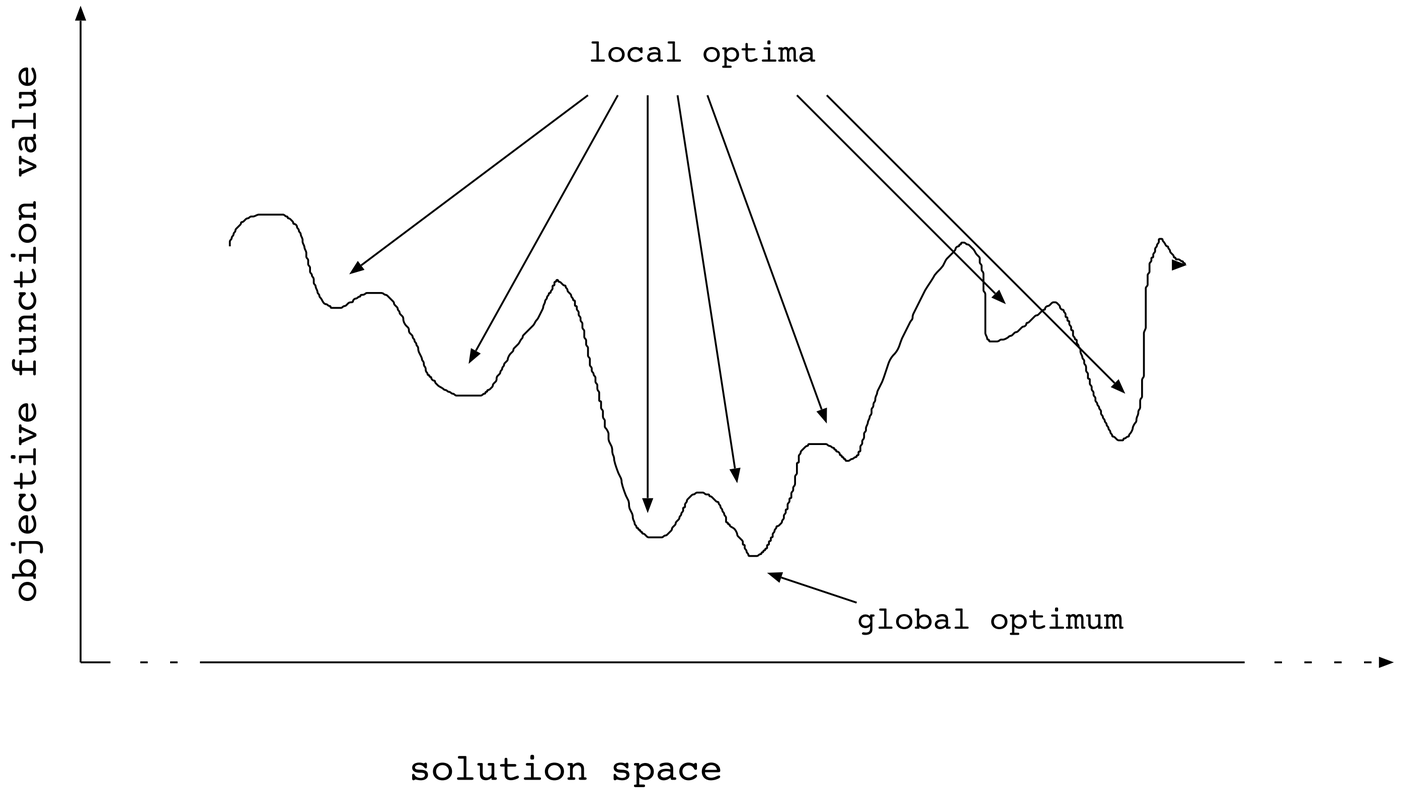
Iterative Improvement for the TSP

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



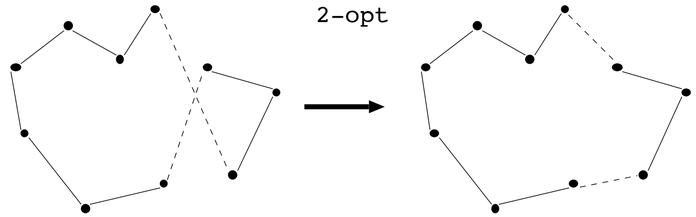
- neighbourhood size $\mathcal{O}(n^k)$

Problems with local search?



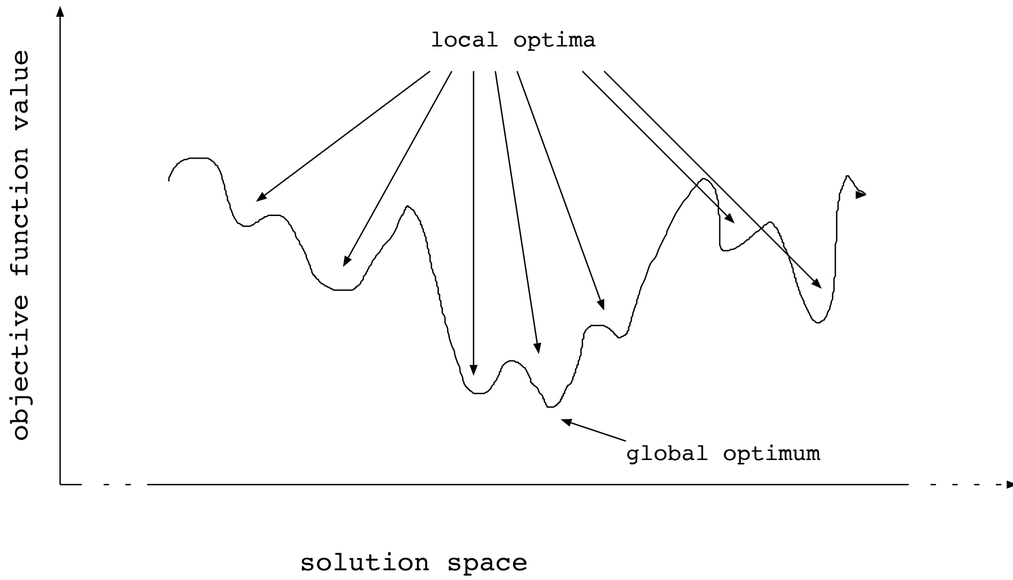
Iterative Improvement for the TSP

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



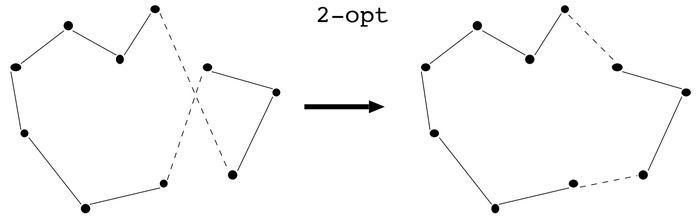
Stochastic Local Search:

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



Iterative Improvement for the TSP

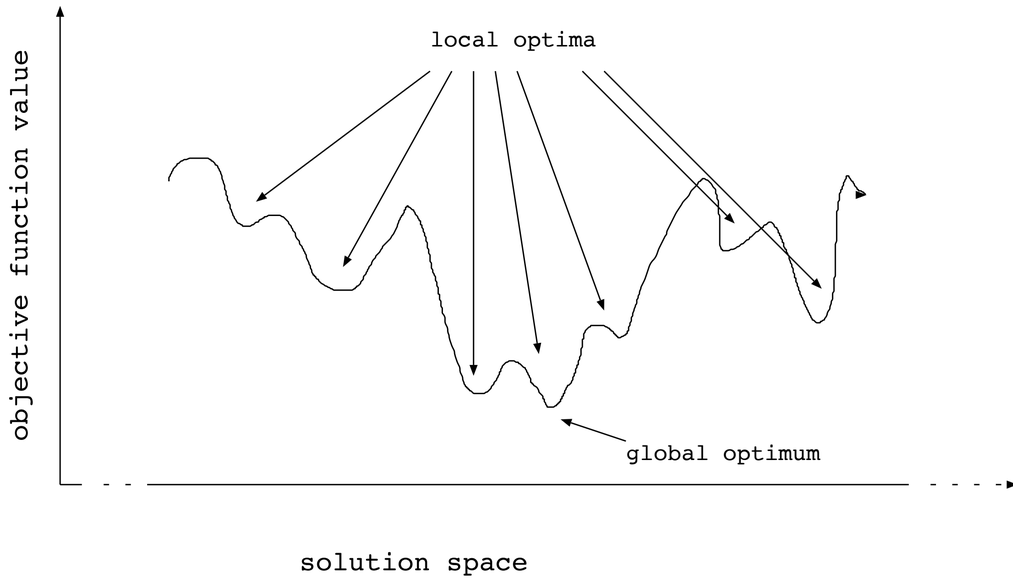
- initial solution is a complete tour
- k-opt neighbourhood: solutions which differ by at most k edges



Stochastic Local Search:

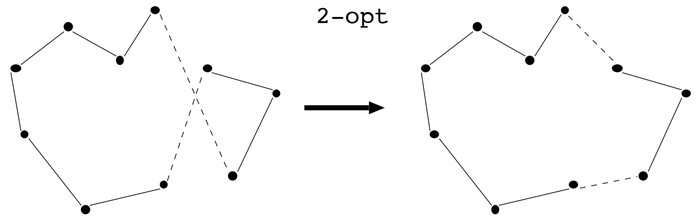
Problems with SLS?

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



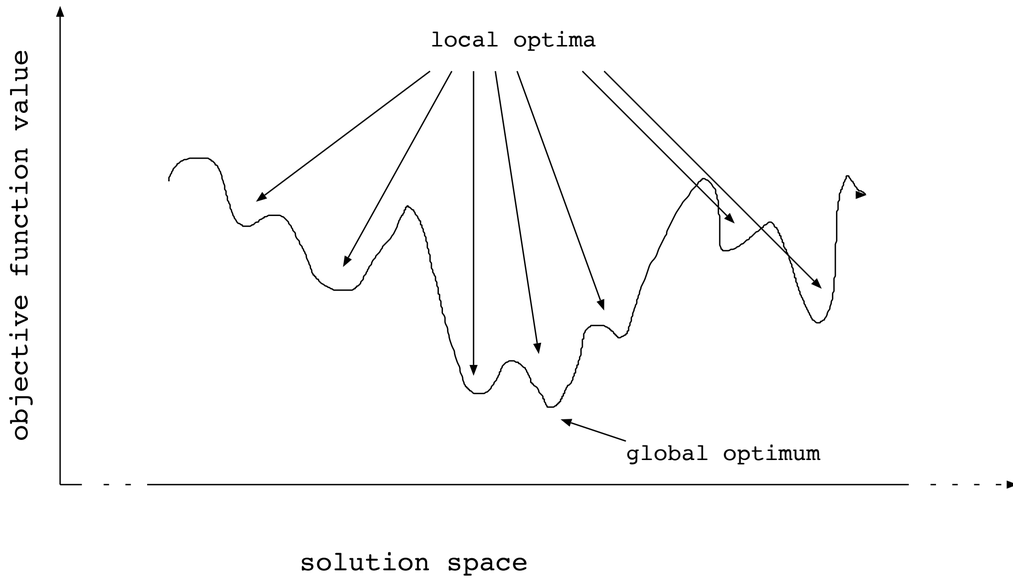
Iterative Improvement for the TSP

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



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

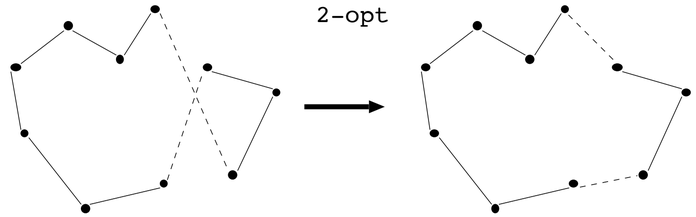


Stochastic Local Search:

- 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

Iterative Improvement for the TSP

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



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

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.

Holger Hoos, CPSC536H at UBC: http://www.cs.ubc.ca/labs/beta/Courses/CPSC536H-12/Slides/intro.pdf

The Scientific Method

make observations

formulate hypothesis/hypotheses (model)

While not satisfied (and deadline not exceeded) iterate:

1. design experiment to falsify model
2. conduct experiment
3. analyse experimental results
4. revise model based on results

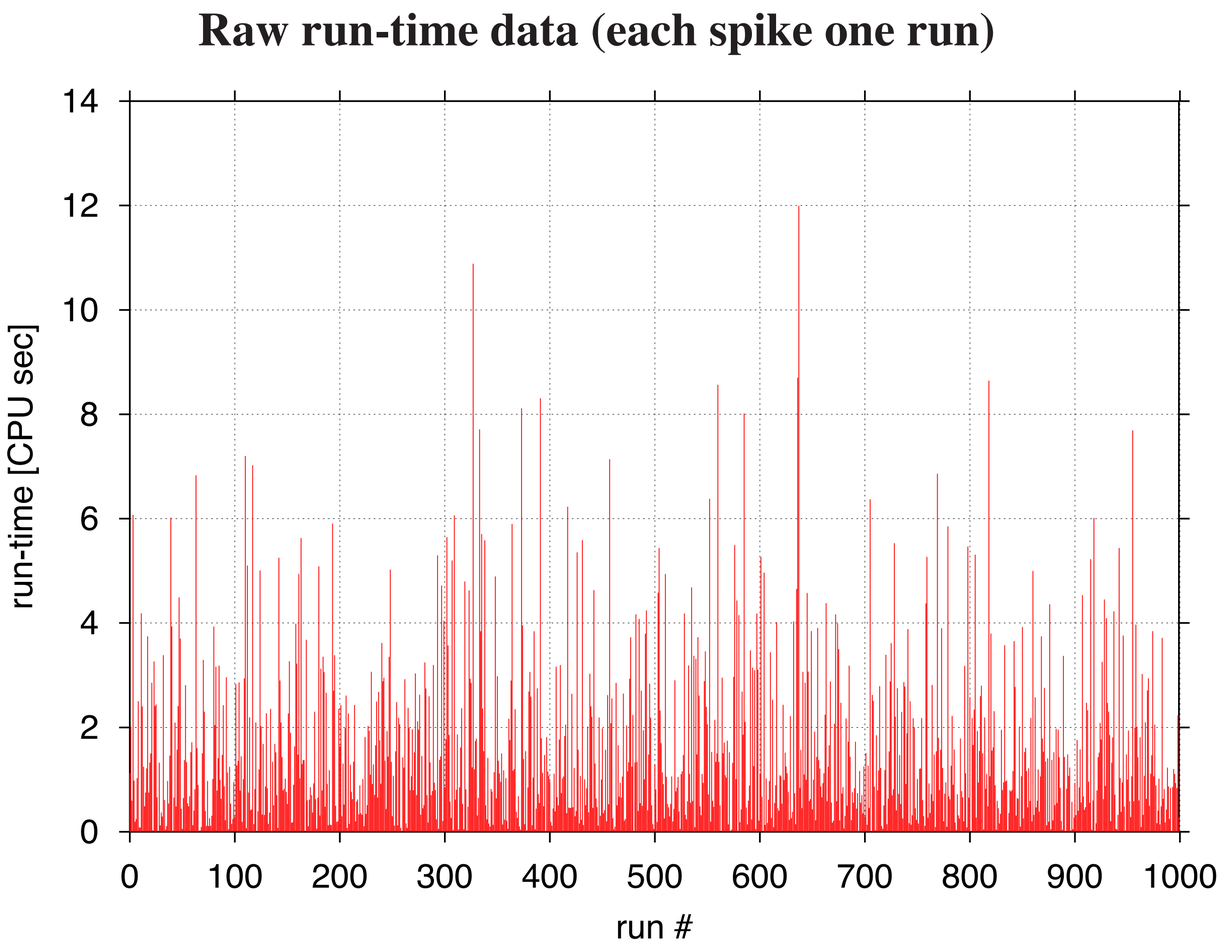
optimization vs decision (problem)

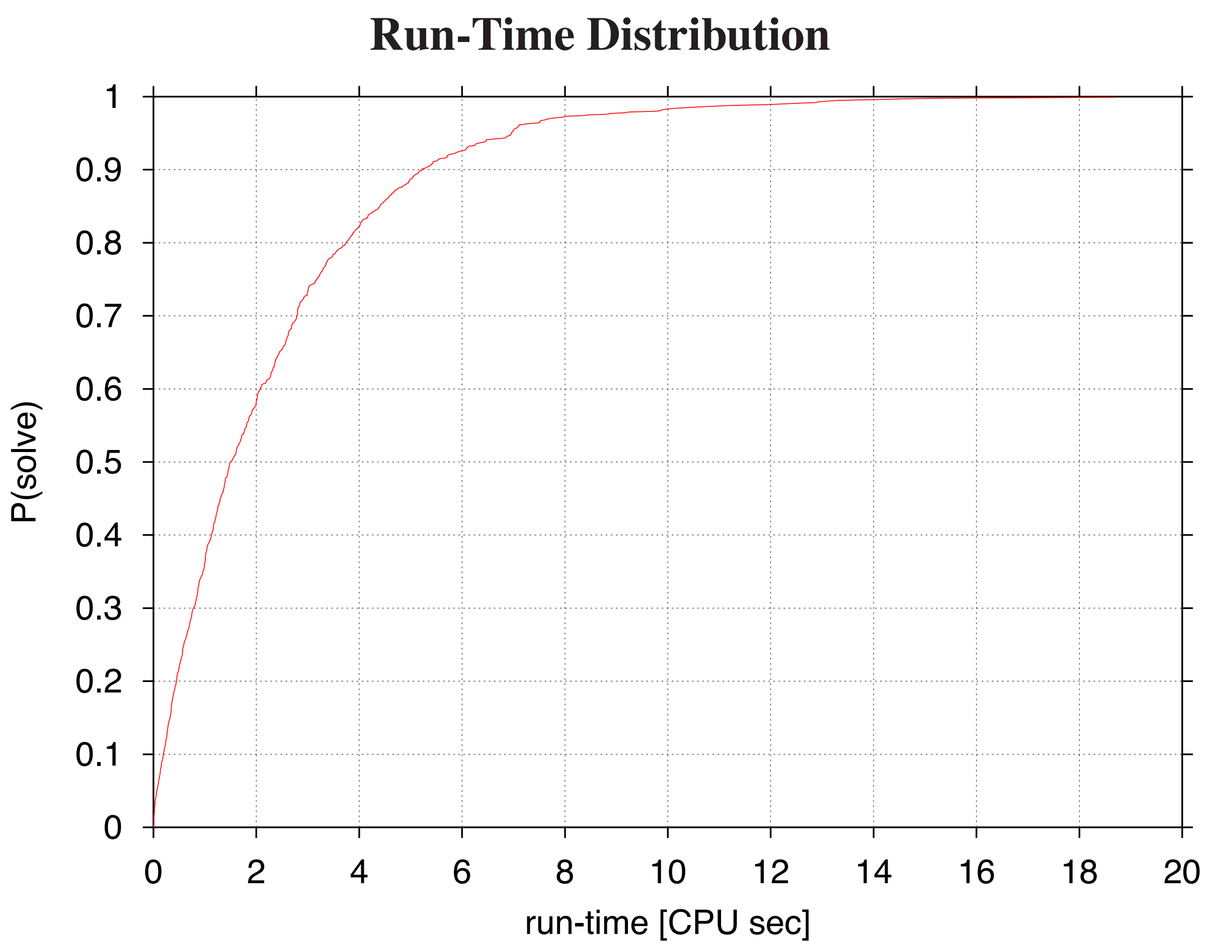
exact vs heuristic

complete vs incomplete

stochastic vs deterministic

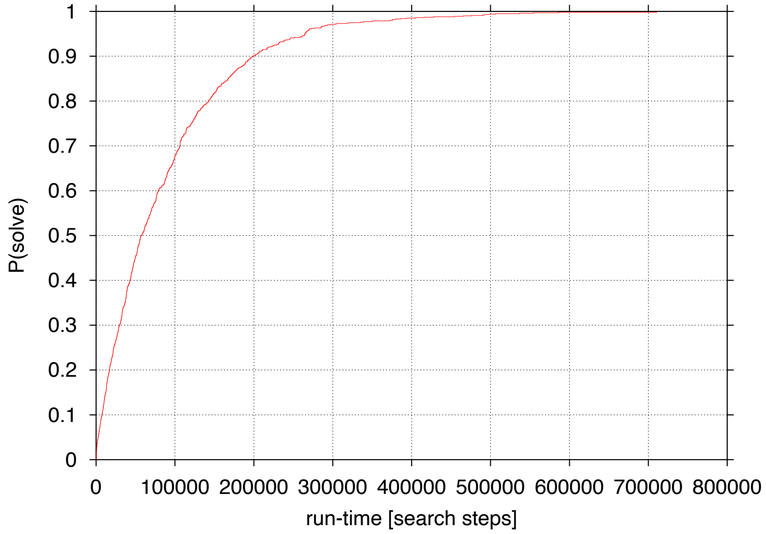
sequential vs parallel

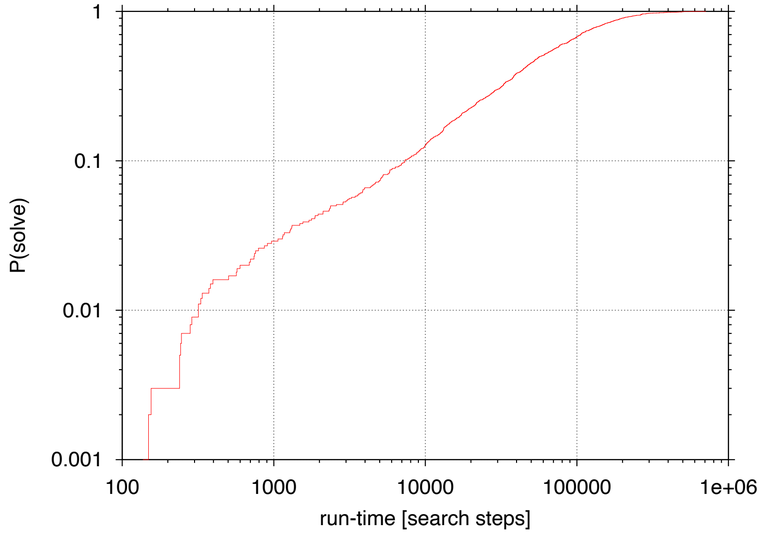


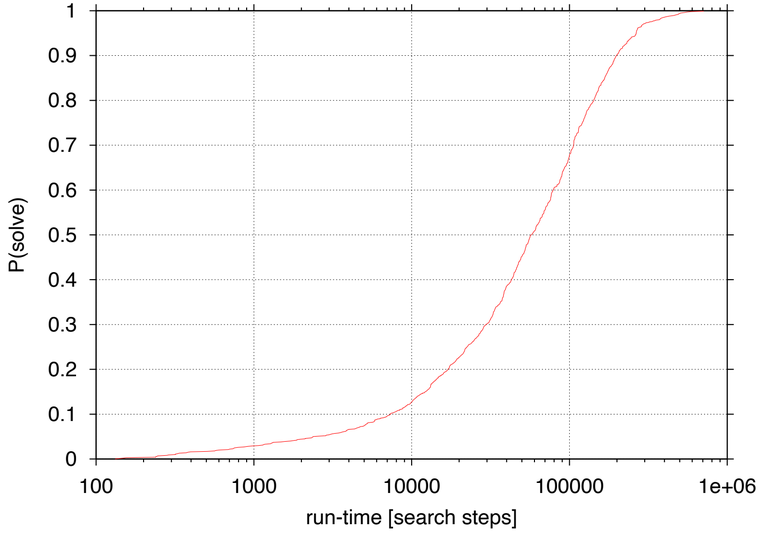


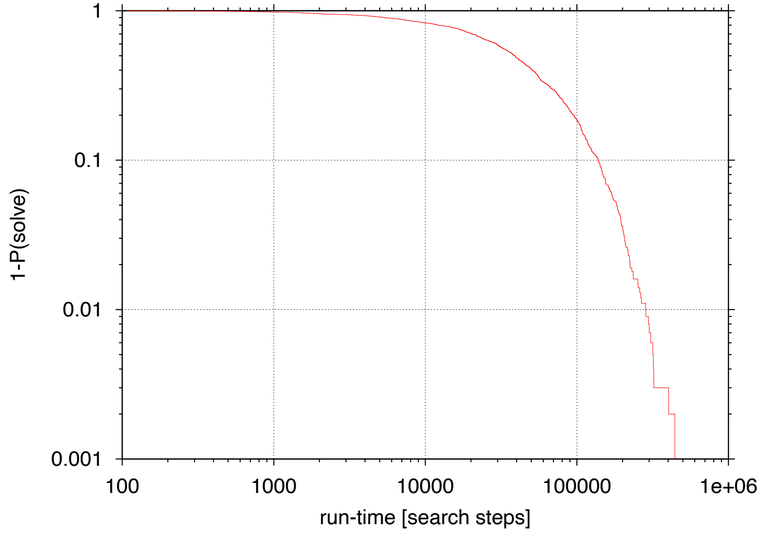
Holger Hoos, CPSC536H at UBC: http://www.cs.ubc.ca/labs/beta/Courses/CPSC536H-12/Slides/m3-all.pdf

RTD Graphs



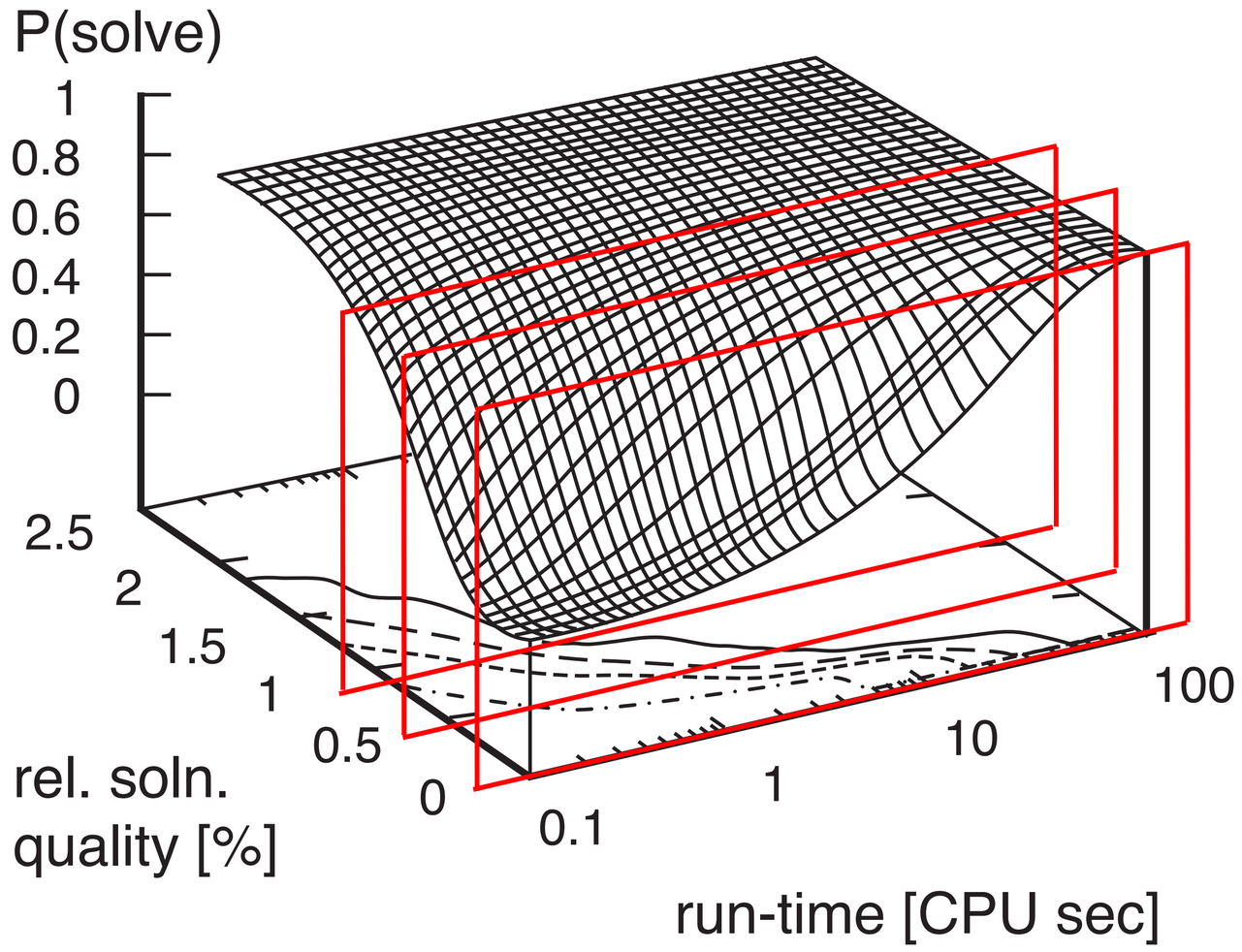




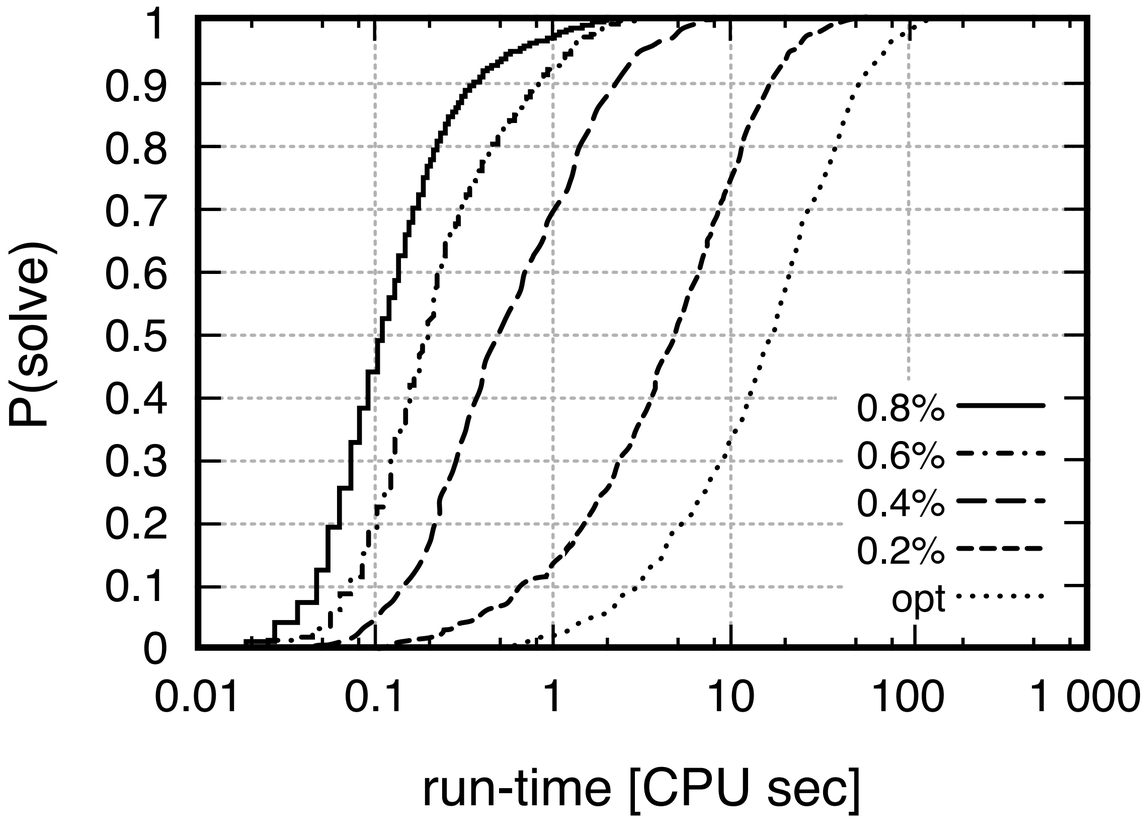


Holger Hoos, CPSC536H at UBC: http://www.cs.ubc.ca/labs/beta/Courses/CPSC536H-12/Slides/m3-all.pdf

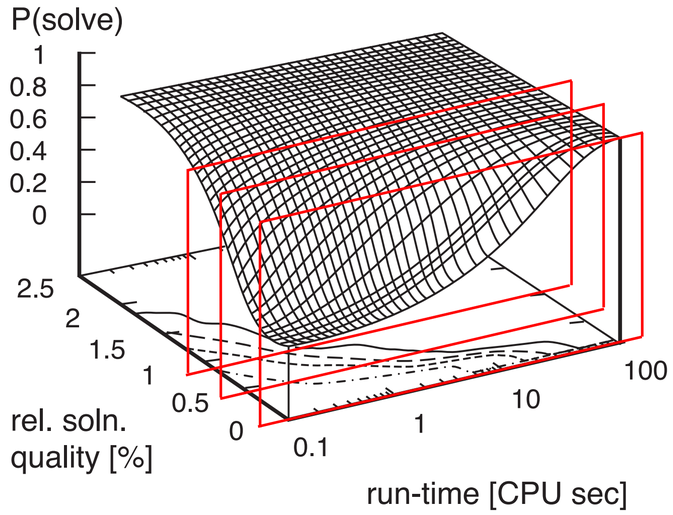
Typical run-time distribution for SLS algorithm applied to hard instance of combinatorial optimisation problem:



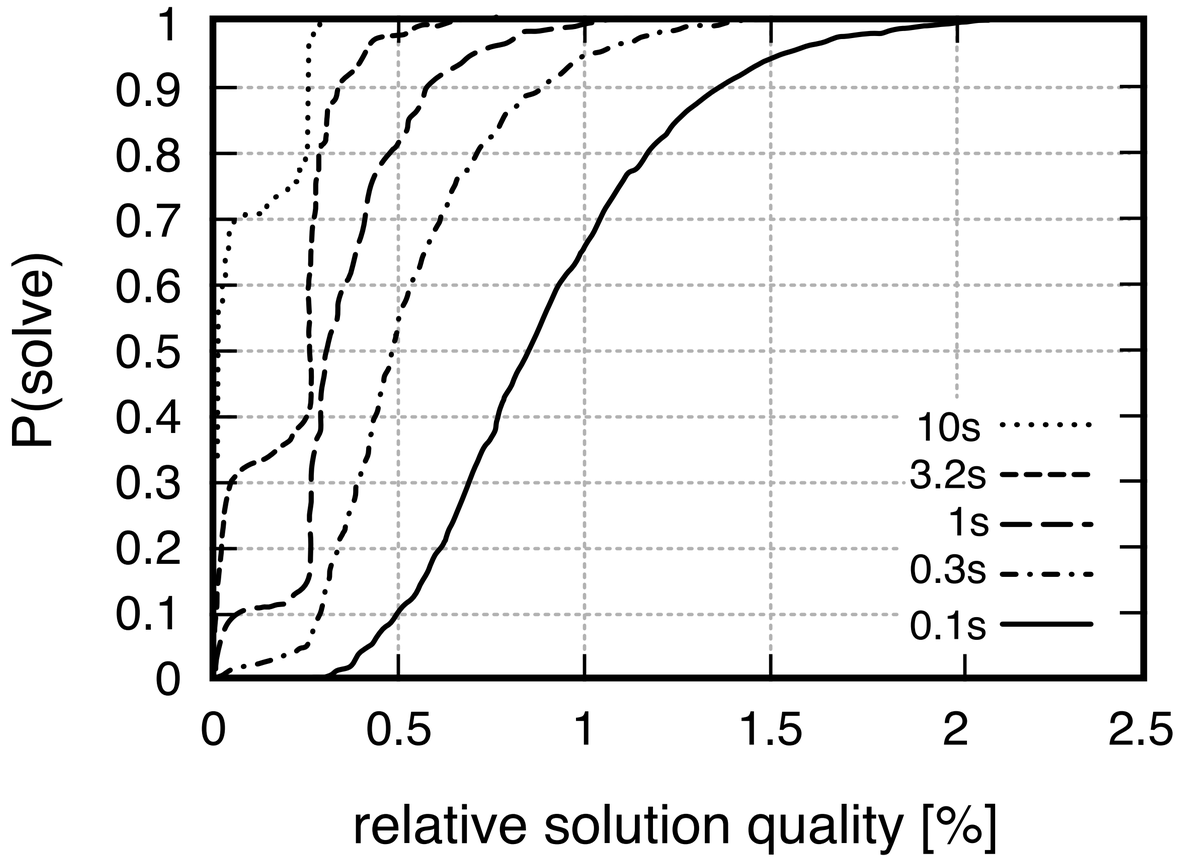
Qualified RTDs for various solution qualities:



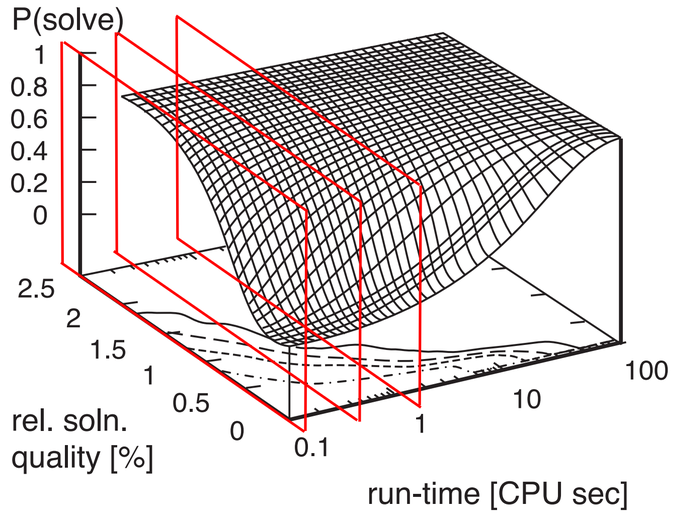
Typical run-time distribution for SLS algorithm applied to hard instance of combinatorial optimisation problem:



Solution quality distributions for various run-times:



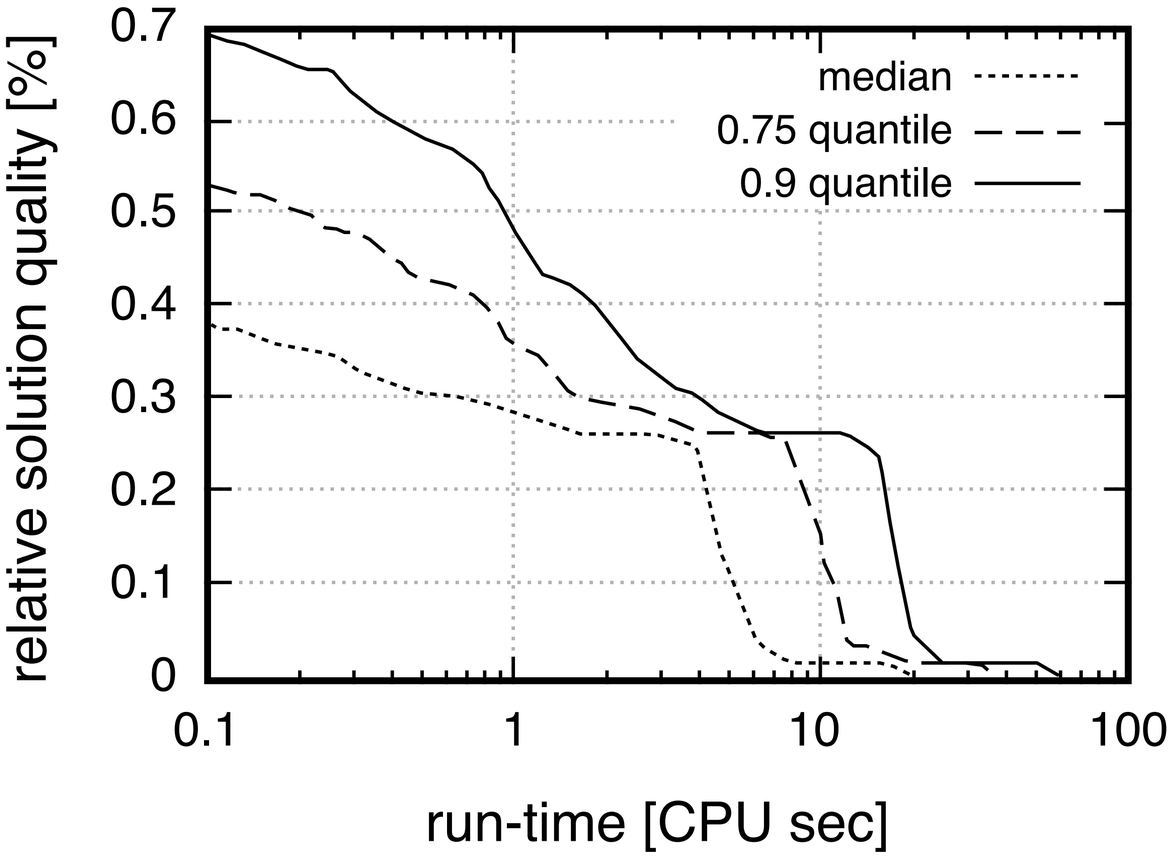
Typical solution quality distributions for SLS algorithm applied to hard instance of combinatorial optimisation problem:



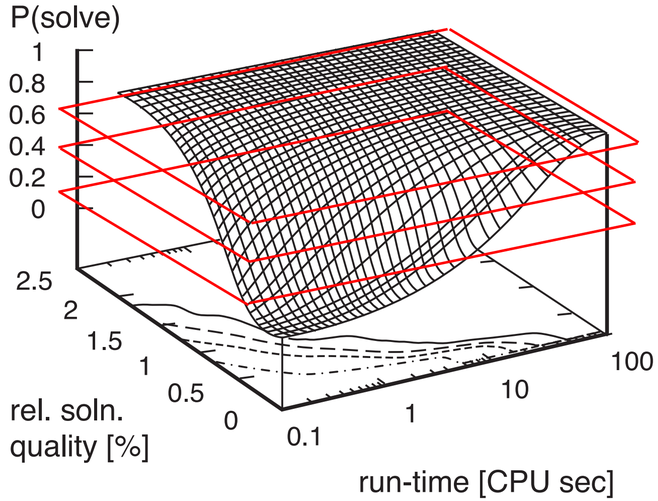
SQT: Solution Quality over Time

Holger Hoos, CPSC536H at UBC: http://www.cs.ubc.ca/labs/beta/Courses/CPSC536H-12/Slides/m5-all.pdf

Typical SQT curves for SLS optimisation algorithms applied to instance of hard combinatorial optimisation problem:



Typical SQT curves for SLS optimisation algorithms applied to instance of hard combinatorial optimisation problem:



Mixed Integer (Linear) Program

$$\max\ c^\intercal x + h^\intercal y$$

$$Ax + Gy \leq b$$

$$x \geq 0 \text{ and integer}, y \geq 0$$

$$A \in \mathbb{R}^{m \times n}, G \in \mathbb{R}^{m \times p}, b \in \mathbb{R}^{m \times 1}, c \in \mathbb{R}^{n \times 1}, h \in \mathbb{R}^{p \times 1}$$

$$x \in \mathbb{R}^{n \times 1}, y \in \mathbb{R}^{p \times 1}$$

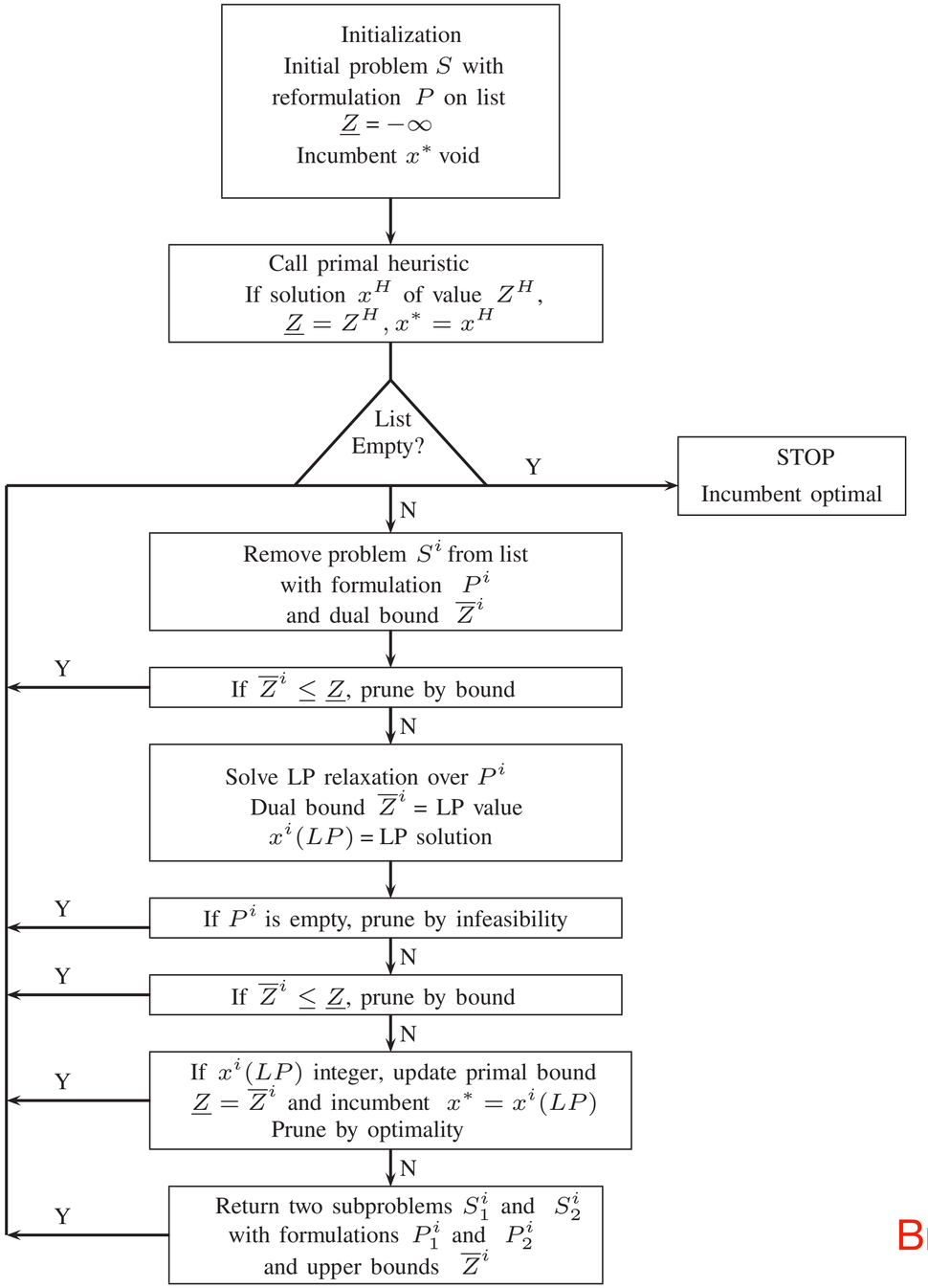


Figure 7.10 Branch-and-bound flow chart.

Primal heuristic

Termination criterion

Node selection

Pruning rule

Solving LP Relaxation

Pruning rule

Pruning rule

Pruning rule

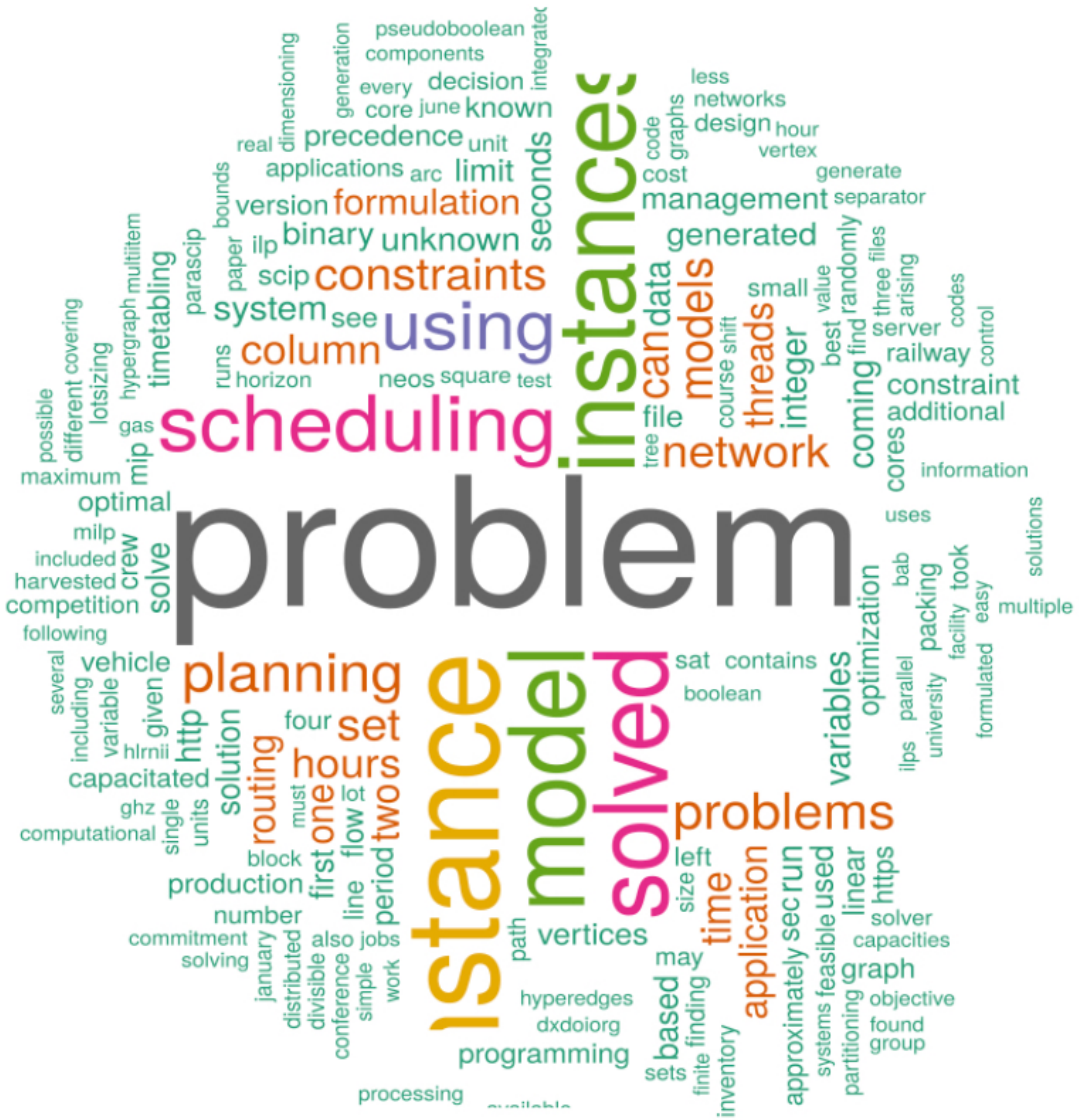
Branching variable selection

Not Secure | miplib2017.zib.de

MIPLIB 2017 About Benchmark Collection Download Help

MIPLIB 2017 -- The Mixed Integer Programming Library

In response to the needs of researchers for access to real-world mixed integer programs, Robert E. Bixby, E.A. Boyd, and R.R. Indovina created in 1992 the MIPLIB, an electronically available library of both pure and mixed integer programs. Since its introduction, MIPLIB has become a standard test set used to compare the performance of mixed integer optimizers. Its availability has provided an important stimulus for researchers in this very active area. The library has now been released in its sixth edition as a collaborative effort between Arizona State University, COIN-OR, CPLEX, FICO, Gurobi, MathWorks, MIPCL, MOSEK, NUOPT, SAS, and Zuse Institute Berlin. Like the previous MIPLIB 2010, two main sets have been compiled from the submissions. The **Benchmark Set** contains 240 instances that are solvable by (the union of) today's codes. For practical reasons, the benchmark instances were selected subject to various constraints regarding solvability and numerical stability. The much larger **Collection Set** represents a diverse selection regardless of the above, benchmark-relevant criteria. Download the instance sets as well as supplementary data, run scripts and the solution checker from our Download page.



TUTORIALS IN
OPERATIONS RESEARCH
INFORMS 2013

 | ISBN 978-0-9843378-4-2
http://dx.doi.org/10.1287/educ.2013.0112

Performance Variability in Mixed-Integer Programming

Andrea Lodi
Department of Electrial, Electronic and Information Engineering, University of Bologna, Bologna, Italy, andrea.lodi@unibo.it

Andrea Tramontani
CPLEX Optimization, IBM, Bologna, Italy, andrea.tramontani@it.ibm.com

Abstract The performance of mixed-integer programming solvers is subject to some unexpected variability that appears, for example, when changing from one computing platform to another, when permuting rows and/or columns of a model, when adding seemingly neutral changes to the solution process, etc. This phenomenon has been observed for decades, but only recently has it started to be methodologically analyzed with the two possible aims of either reducing or exploiting it, ideally both. In this tutorial we discuss the roots of performance variability, we provide useful tips to recognize it, and we point out some severe misinterpretations that might be generated by not performing/analyzing benchmark results carefully. Finally, we report on the most recent attempts to gain from variability.

Keywords mixed-integer programming; computation; branch and bound; benchmarking; erraticism; floating-point precision

TABLE 1. Impact of disabling zero-half cuts in CPLEX 12.5 (correct model grouping).

		Default	No zero-half cuts				Affected	
Class	#models	#tilim	#tilim	#wins	#losses	Time	#models	Time
All	3,136	88	106	395	370	1.03	1,406	1.07
[0, 10k]	3,065	17	35	395	370	1.03	1,406	1.07
[1, 10k]	1,875	17	35	376	359	1.05	1,096	1.09
[10, 10k]	1,103	17	35	272	274	1.08	715	1.13
[100, 10k]	598	17	35	168	184	1.13	422	1.19
[1k, 10k]	243	17	35	69	85	1.23	188	1.31

TABLE 2. Impact of disabling zero-half cuts in CPLEX 12.5 (biased model grouping).

		Default	No zero-half cuts				Affected	
Class	#models	#tilim	#tilim	#wins	#losses	Time	#models	Time
All	3,136	88	106	395	370	1.03	1,406	1.07
[0, 10k]	3,065	17	35	395	370	1.03	1,406	1.07
[1, 10k]	1,846	17	34	376	333	1.03	1,068	1.06
[10, 10k]	1,072	17	34	272	243	1.03	684	1.05
[100, 10k]	549	17	33	168	135	0.97	373	0.95
[1k, 10k]	207	17	25	69	50	0.82	153	0.76

TABLE 3. Comparison of CPLEX 12.5 using two different random seeds (correct model grouping).

		Seed 1	Seed 2				Affected	
Class	#models	#tilim	#tilim	#wins	#losses	Time	#models	Time
All	3,121	87	105	581	588	1.00	2,264	1.01
[0, 10k]	3,051	17	35	581	588	1.00	2,264	1.01
[1, 10k]	1,864	17	35	558	558	1.01	1,693	1.01
[10, 10k]	1,110	17	35	398	396	1.01	1,056	1.01
[100, 10k]	596	17	35	238	237	1.02	585	1.02
[1k, 10k]	236	17	35	101	100	1.08	235	1.08

Learning to Branch in Mixed Integer Programming, Khalil et al., AAAI-16,
https://ekhalil.com/files/papers/KhaLebSonNemDil16.pdf

	CPLEX-D	SB	PC	SB+PC	SB+ML
CPLEX-D		1.39 (389)	0.64 (449)	0.84 (452)	0.97 (463)
SB	0.72 (389)		0.47 (389)	0.61 (388)	0.76 (389)
PC	1.56 (449)	2.11 (389)		1.34 (445)	1.59 (450)
SB+PC	1.20 (452)	1.63 (388)	0.75 (445)		1.22 (454)
SB+ML	1.03 (463)	1.32 (389)	0.63 (450)	0.82 (454)	

Table 3: Ratios for the shifted geometric means (shift 10) over nodes on instances solved by both strategies. The first value in a cell in row $\mathcal{A}$ and column $\mathcal{B}$ is the ratio of the average number of nodes used by $\mathcal{A}$ to that of $\mathcal{B}$. The second value is the number of instances solved by both $\mathcal{A}$ and $\mathcal{B}$.

Learning to Branch in Mixed Integer Programming, Khalil et al., AAAI-16, https://ekhalil.com/files/papers/KhaLebSonNemDil16.pdf

	CPLEX-D	SB	PC	SB+PC
CPLEX-D				
SB	5/264/0/125/123			
PC	8/164/0/285/63	68/63/0/326/5		
SB+PC	8/227/0/225/60	72/66/7/315/6	15/320/0/125/12	
SB+ML	8/267/0/196/49	82/96/7/286/5	21/355/0/95/7	17/300/58/96/6

Table 4: Win-tie-loss matrix for the number of nodes. A quintuple in a cell in row $\mathcal{A}$ and column $\mathcal{B}$ has: the number of absolute wins, wins, ties, losses and absolute losses for $\mathcal{A}$ against $\mathcal{B}$, w.r.t. the number of nodes.