Hybridisation of metaheuristic and exact optimisation techniques for Mixed Integer Programs

Matteo Fischetti, DEI, University of Padova

EuroGP-EvoCOP-EVOWorkshops 2005, Lausanne, March 29-April 1st, 2005

MIP solvers for hard optimization problems

- **Mixed-integer linear programming** (MIP) plays a central role in modelling difficult-to-solve (NP-hard) combinatorial problems

- General-purpose (exact) MIP solvers are very sophisticated tools, but in some hard cases they are **not adequate** even after clever tuning

- One is therefore tempted **to quit the MIP framework** and to design ad-hoc heuristics for the specific problem at hand, thus loosing the advantage of working in a generic MIP framework

- As a matter of fact, too often a MIP model is developed only "to better describe the problem" or, in the best case, to compute bounds for **benchmarking** the proposed ad-hoc heuristics

I MIP you

A neologism: To ***MIP something*** = translate into a MIP model and solve through a black-box solver

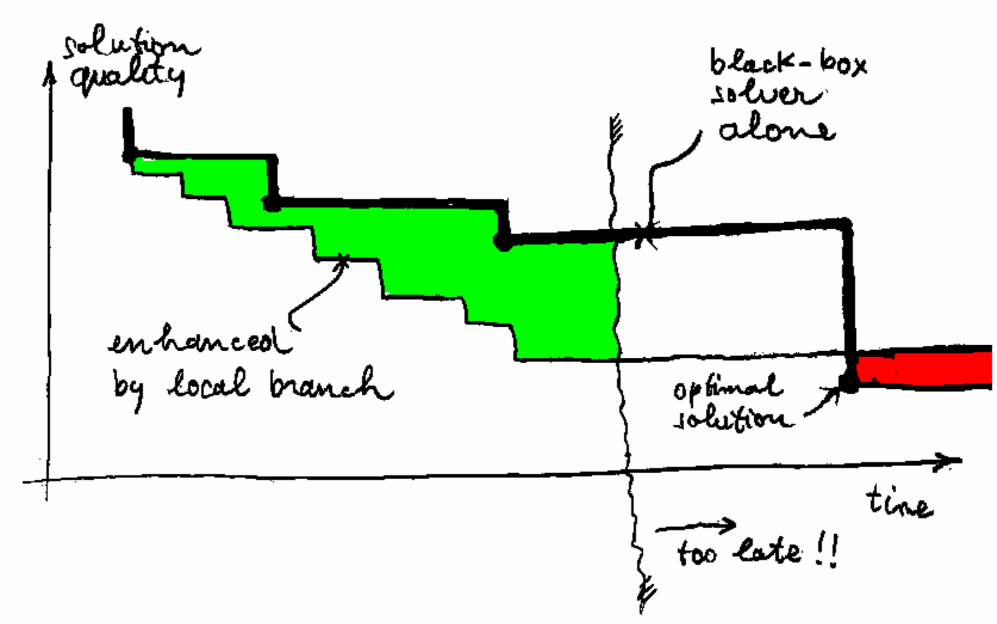
MIP solver enslaved to the local-search metaheuristic

- **MIPping local search:** use (as a black-box) a general-purpose MIP solver to explore large solution neighbourhoods defined through invalid linear inequalities called **local branching** cuts

Heuristic enslaved to the exact MIP solver

- **MIPping Chvàtal-Gomory cuts**: a “dual” scenario where a MIP heuristic is used to derive dual information (cutting planes) enhancing the convergence of an exact MIP solver

Local Branching



(joint work with **Andrea Lodi, DEIS, University of Bologna)**

0-1 Mixed-Integer Programs

- We consider a MIP with 0-1 variables

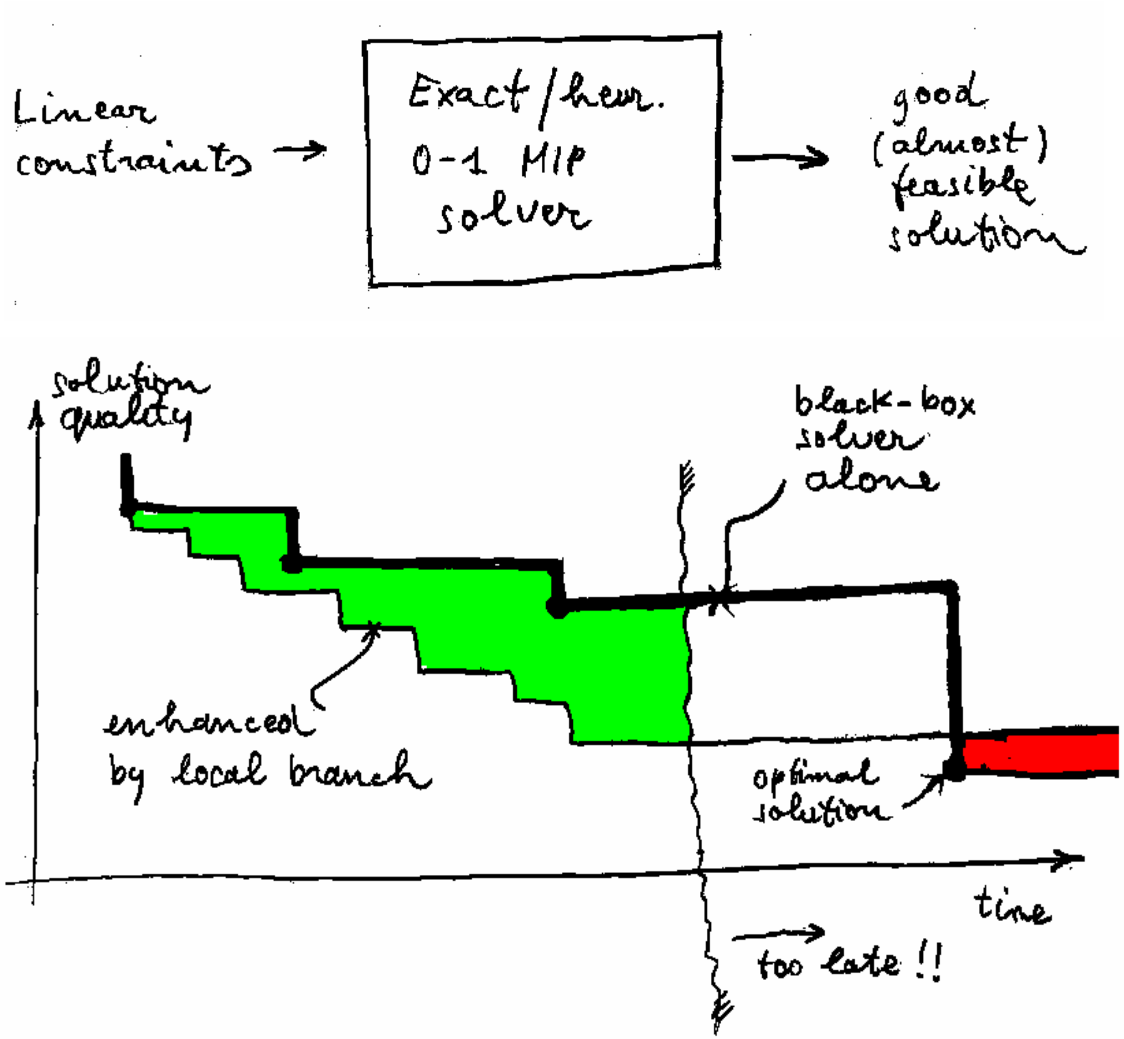
$$
\begin{aligned}
&\min c^T x \\
&Ax \geq b \\
&x_j \in \{0,1\} && \forall j \in \mathcal{B} \neq \emptyset \\
&x_j \geq 0, \text{ integer} && \forall j \in \mathcal{G} \\
&x_j \geq 0 && \forall j \in \mathcal{C}
\end{aligned}
$$

Relevant cases:

- 0-1 ILP's (generic or with a special structure)
- MIP's with no "general integer" variables
- MIP's with both general integer and binary variables, the latter being often used to activate/deactivate costs/constraints (possibly using BIG-M tricks…)

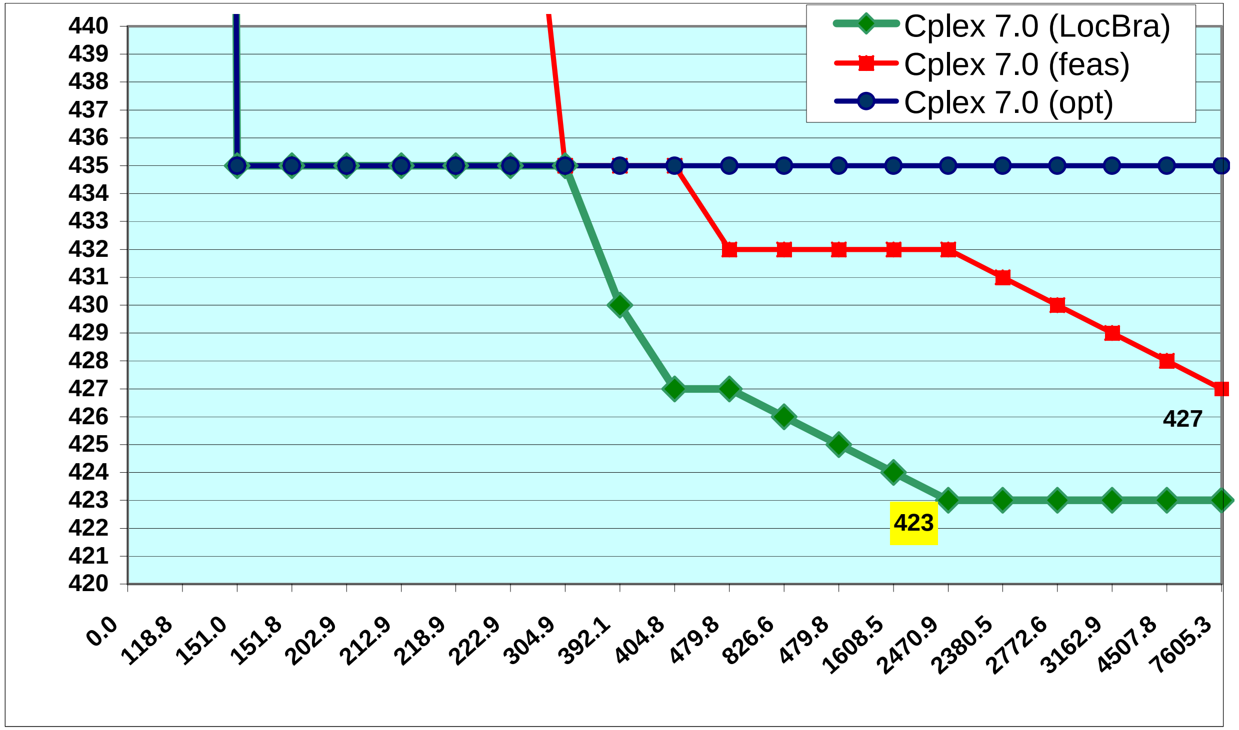
Assumption: once the binary variables have been fixed, the problem becomes (relatively) easy to solve

- We aim at embedding a **black-box** (general-purpose or specific) 0-1 MIP solver within an overall **heuristic framework** that “helps” the solver to deliver improved heuristic solutions



The desired “Italian flag”

An example: the hard MIPLIB problem seymour.lp



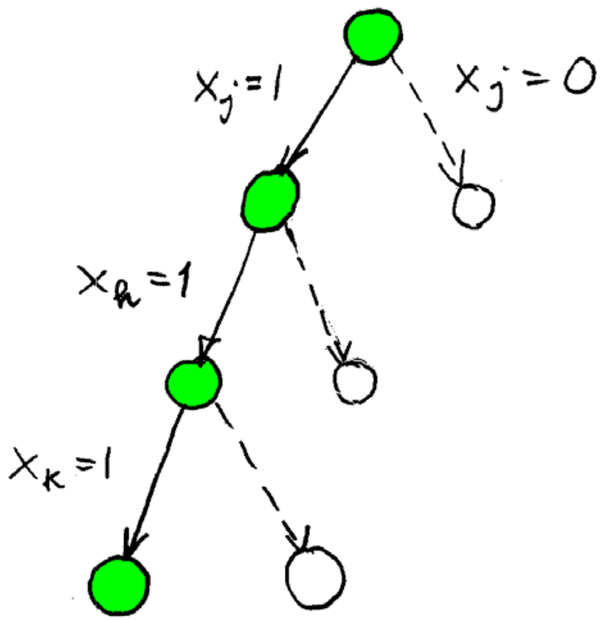
Local Branch heuristic on a hard MIPLIB problem (seymour.lp)

Variable-fixing strategy (hard version)

A commonly-used (often quite effective) **diving** heuristic framework:

- Let x^H be an (almost) feasible "target solution"

- Heuristic **depth-first search** of the branching tree:



- iteratively **fix to 1** certain "highly efficient" variables x_j such as $x_j^H = 1$ (**green nodes**)

- apply the **black-box module** to some **green nodes** only

- only limited **backtracking** allowed

Advantages:

- Problem size quickly reduced: the black-box solver can concentrate on smaller and smaller "**core problems**"
- The black-box solver is applied over and over on different subproblems (**diversification**)

Disadvantages:

- How to choose the "highly efficient" variables to be fixed?
- Wrong choices at early levels are typically very difficult to detect, even when **lower bounds** are computed along the way…

How to reach a sufficiently-deep branching level with a good lower bound?

Variable-fixing strategy (soft version): local branching

- **General idea:** don't decide the actual variables in $S^H := \{ j \in B : x_j^H = 1\}$ to be fixed (a difficult task!), but just their **number** $|S^H| - k$

- Introduce the **Local Branching** constraint

$$\Delta(x, x^H) := \sum_{j \in B : x_j^H = 1} (1 - x_j) \le k$$

or, more generally,

$$\Delta(x, x^H) := \sum_{j \in B : x_j^H = 0} x_j + \sum_{j \in B : x_j^H = 1} (1 - x_j) \le k$$

in the original MIP model, so as to define a convenient **k-OPT neighbourhood** $N(x^H, k)$ of the target solution x^H

"Akin to k-OPT for TSP"

Local branching in an exact solution framework

- Alternate between **strategic** and **tactical** branching decisions:

  - **STRATEGIC (high level) branching phase:**
    - concentrate on a convenient target solution and/or a certain neighbourhood size k

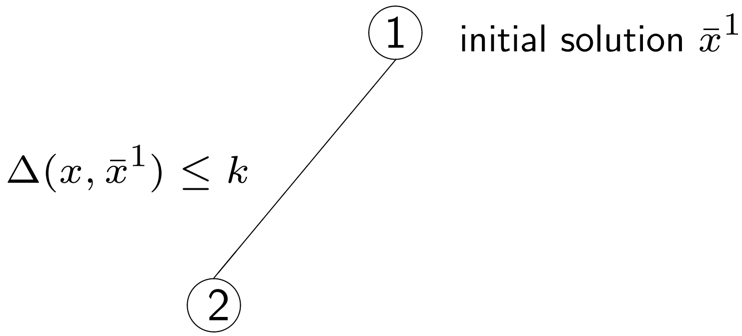
  - **TACTICAL (fine grain) branching phase:**
    - search $N(x^H, k)$ by means of the black-box module (e.g. a general-purpose MIP code using branching on variables…)

Conjecture: a small value of k drives the black-box solver towards integrality as effectively as fixing a large number of variables, but with a **much larger degree of freedom** ➔ better solutions can be found at early branching levels…

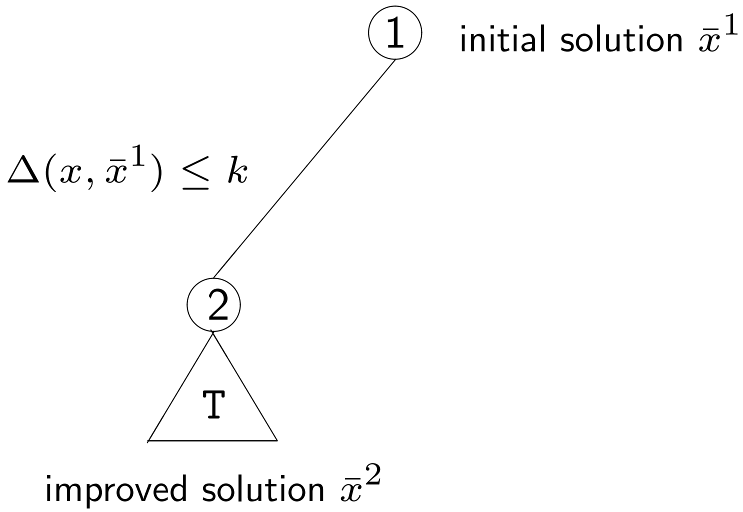
The basic local branching scheme

1 initial solution $\bar{x}^1$

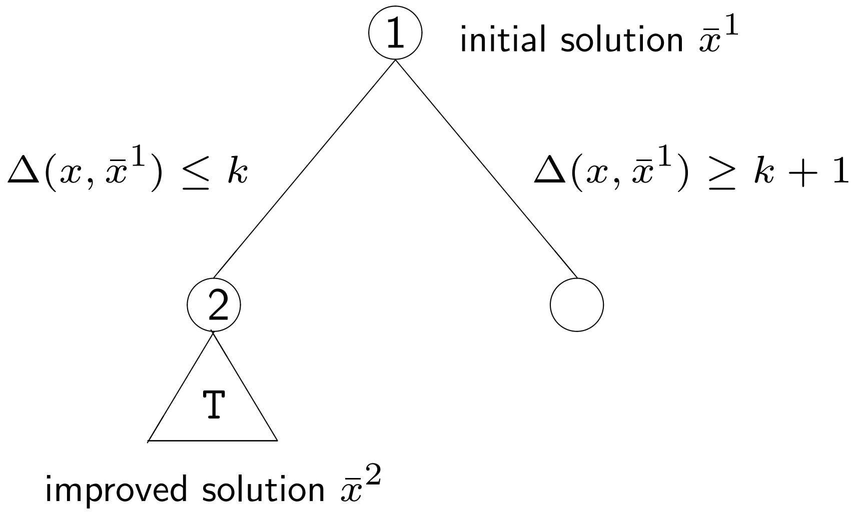
The basic local branching scheme



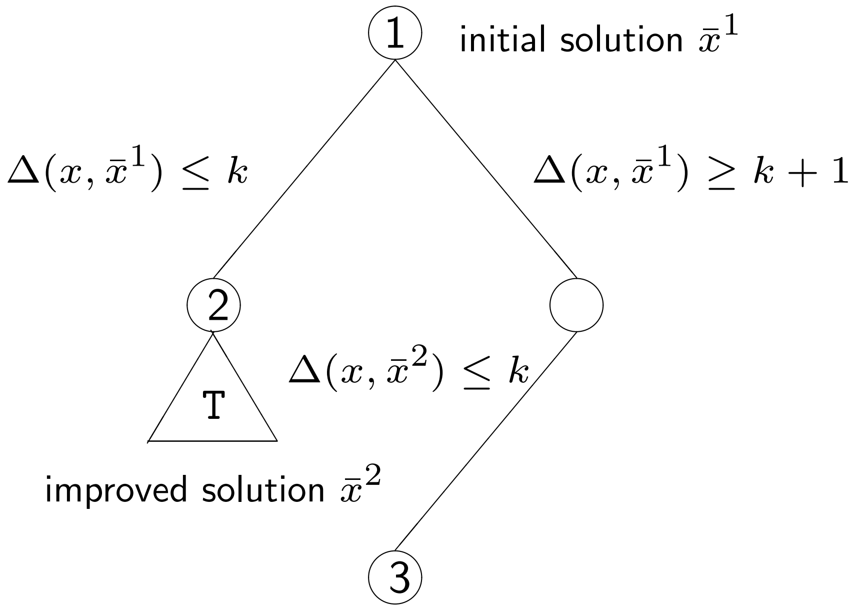
The basic local branching scheme



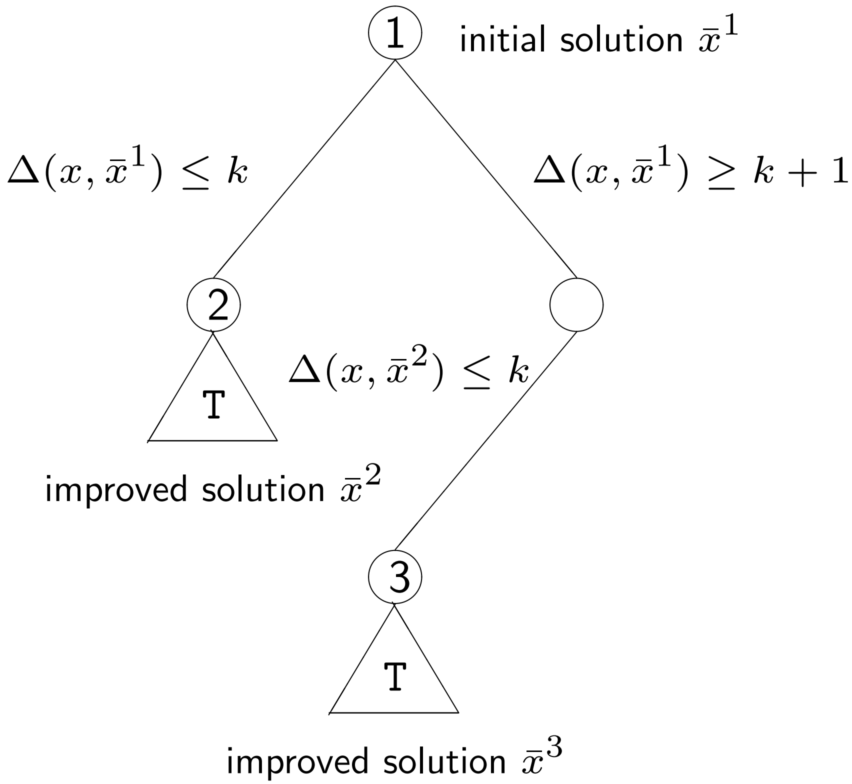
The basic local branching scheme



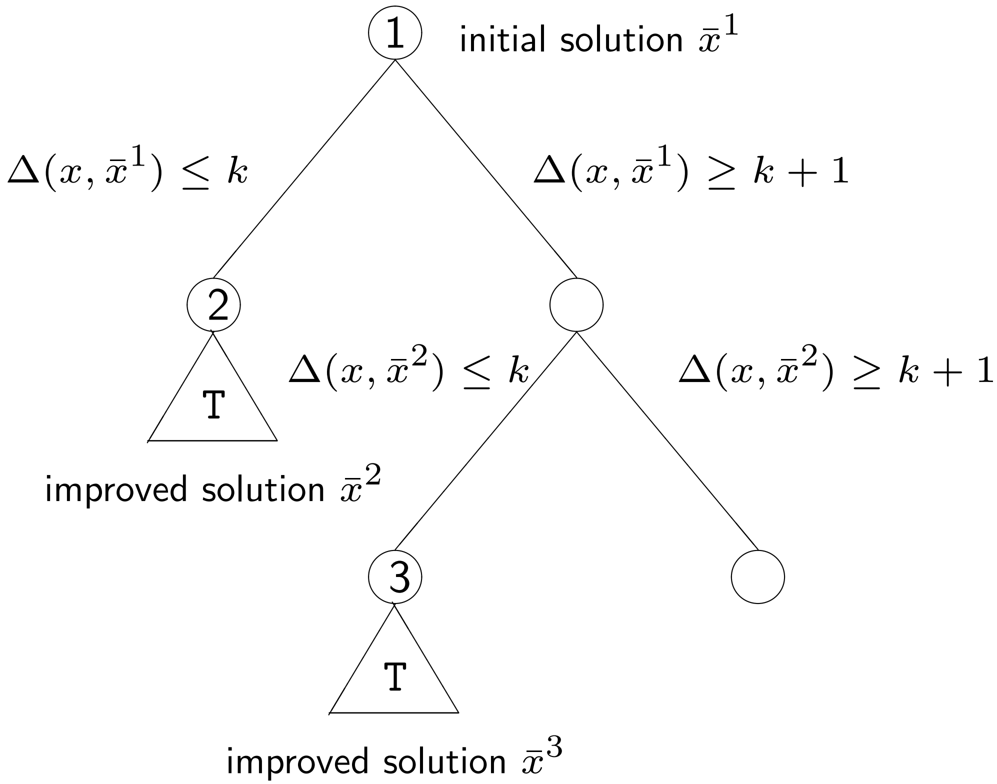
The basic local branching scheme



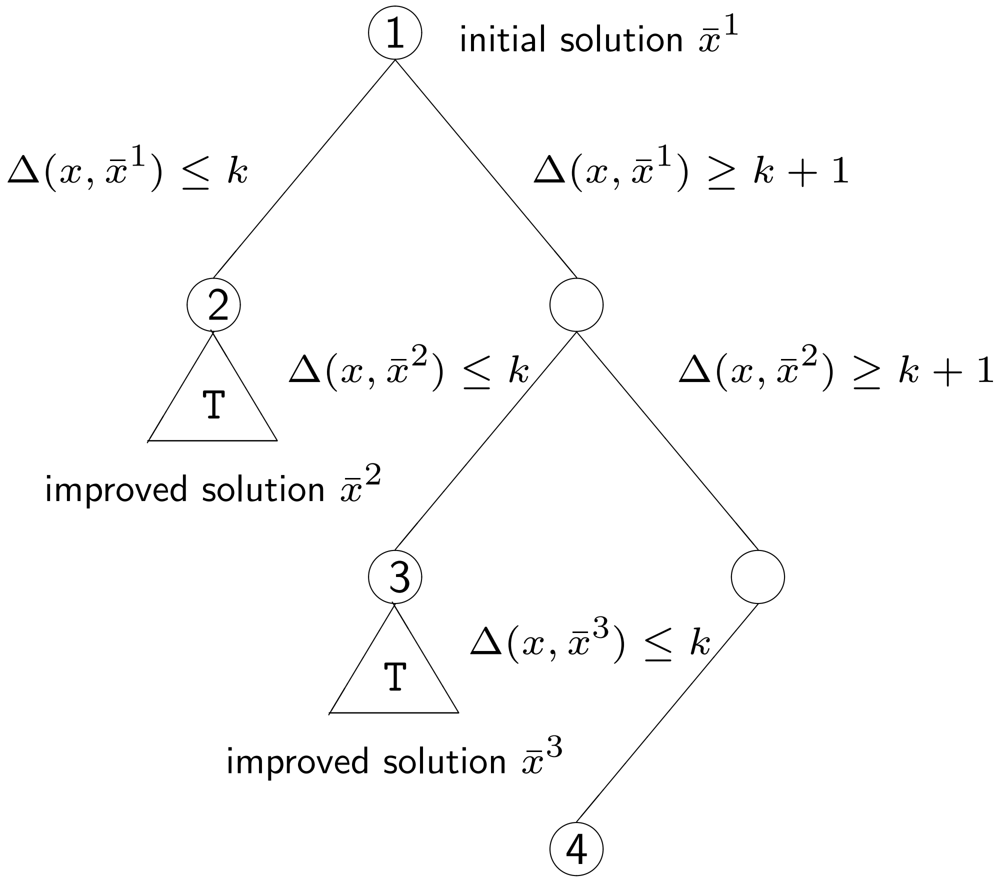
The basic local branching scheme



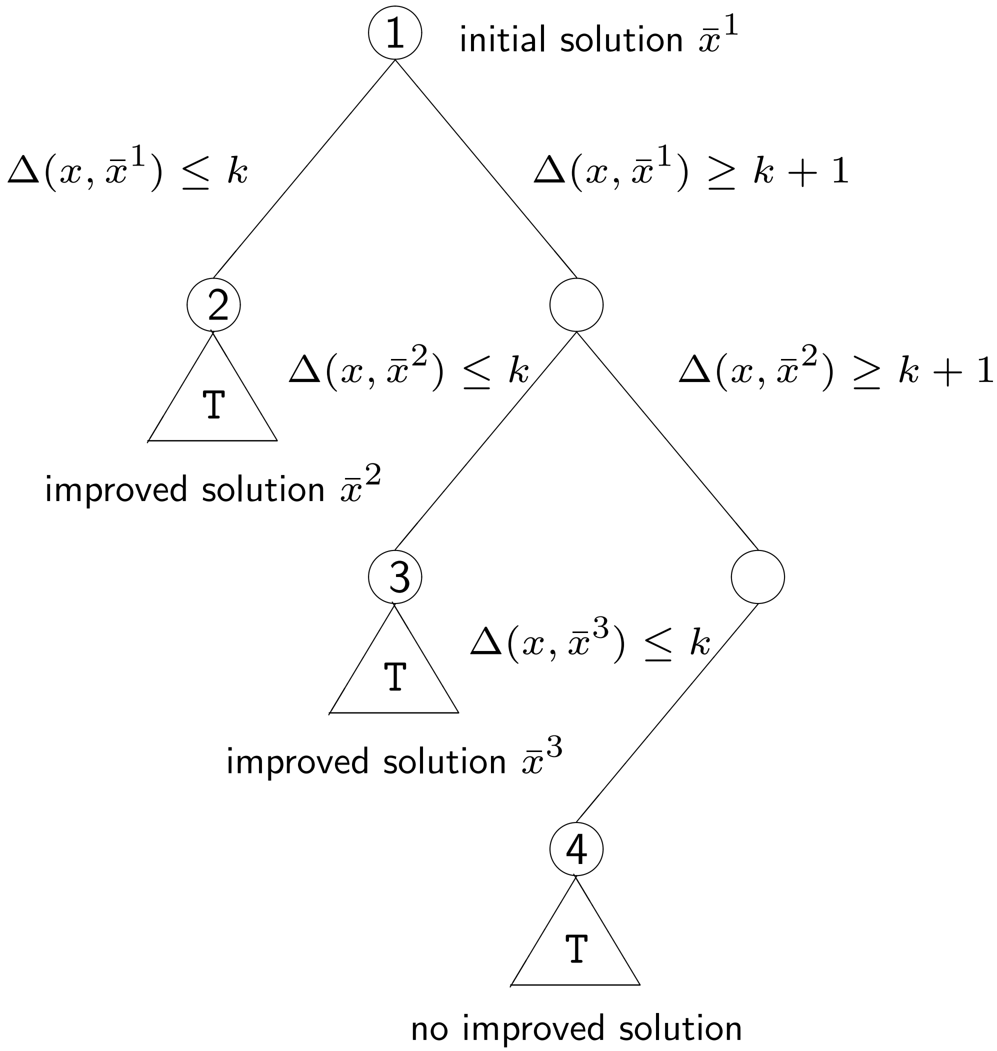
The basic local branching scheme



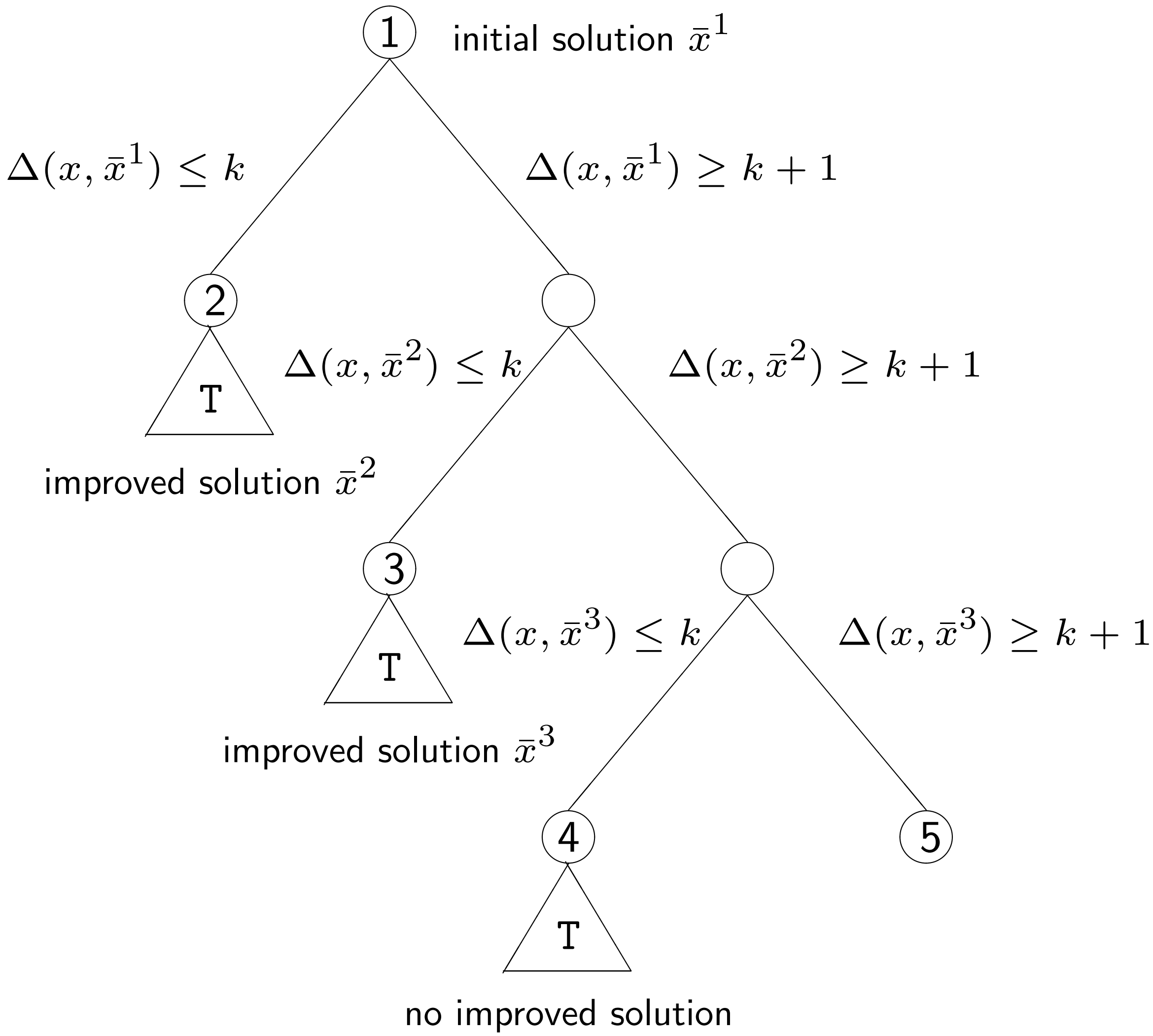
The basic local branching scheme



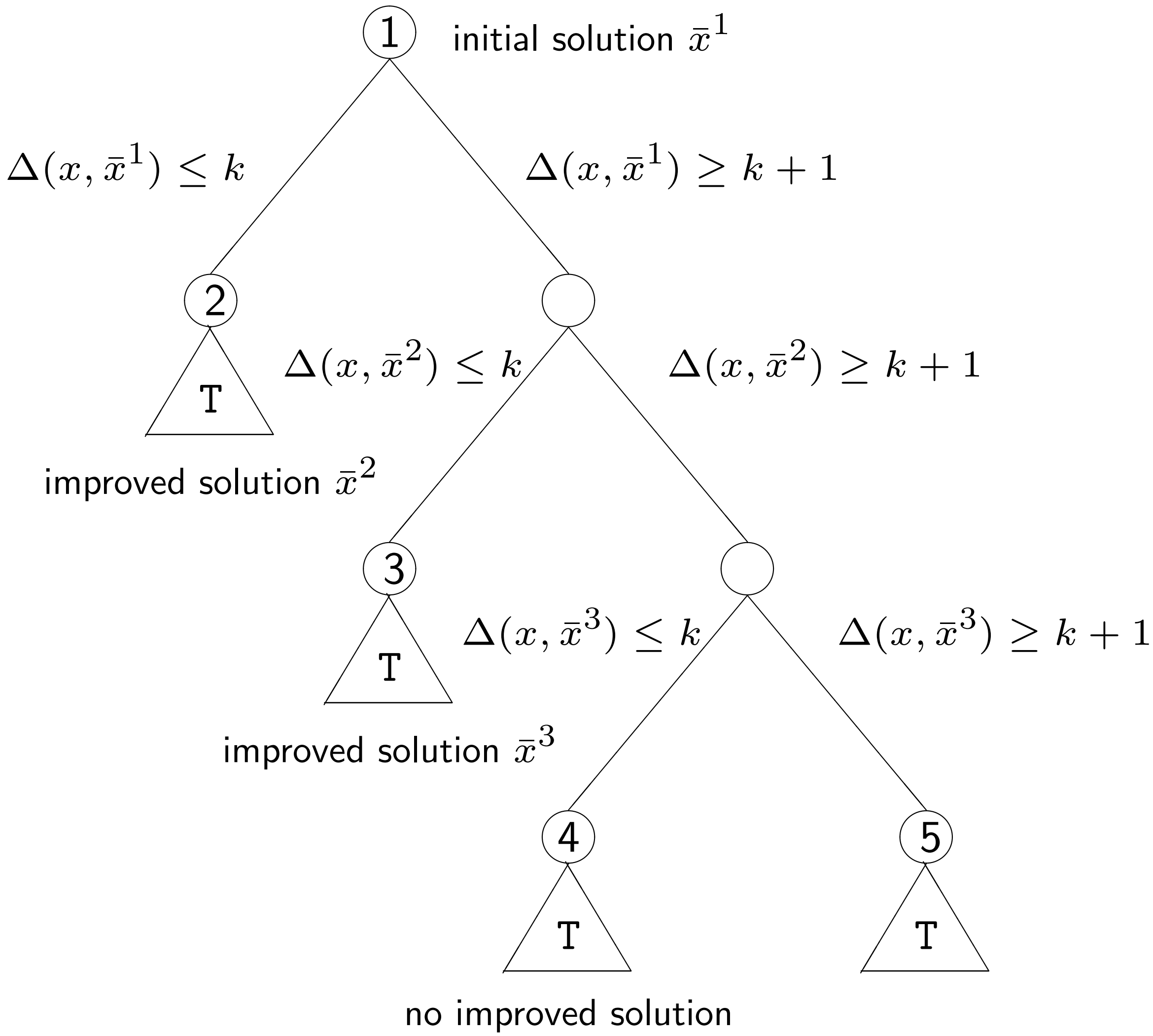
The basic local branching scheme



The basic local branching scheme



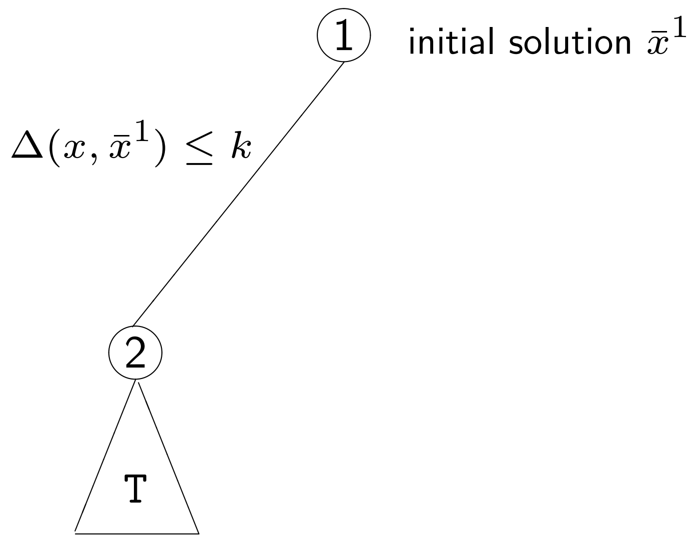
The basic local branching scheme



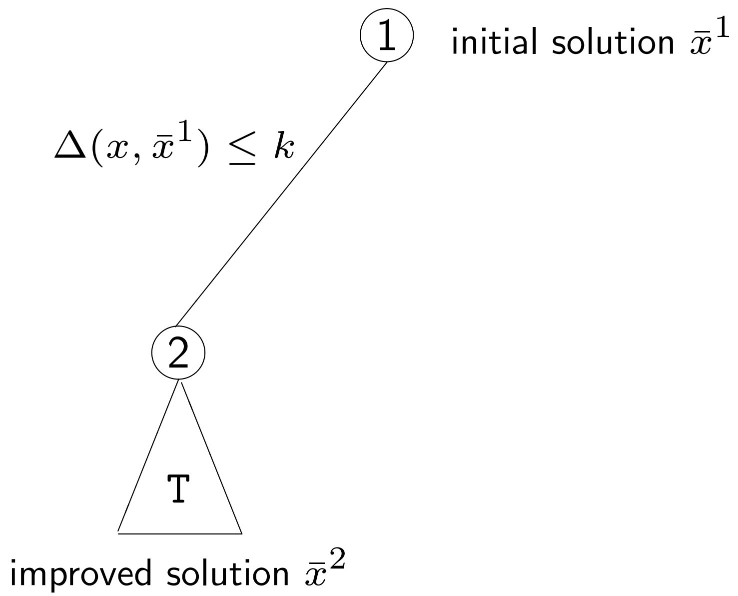
Working with a node time limit: case (a)

1 initial solution $\bar{x}^1$

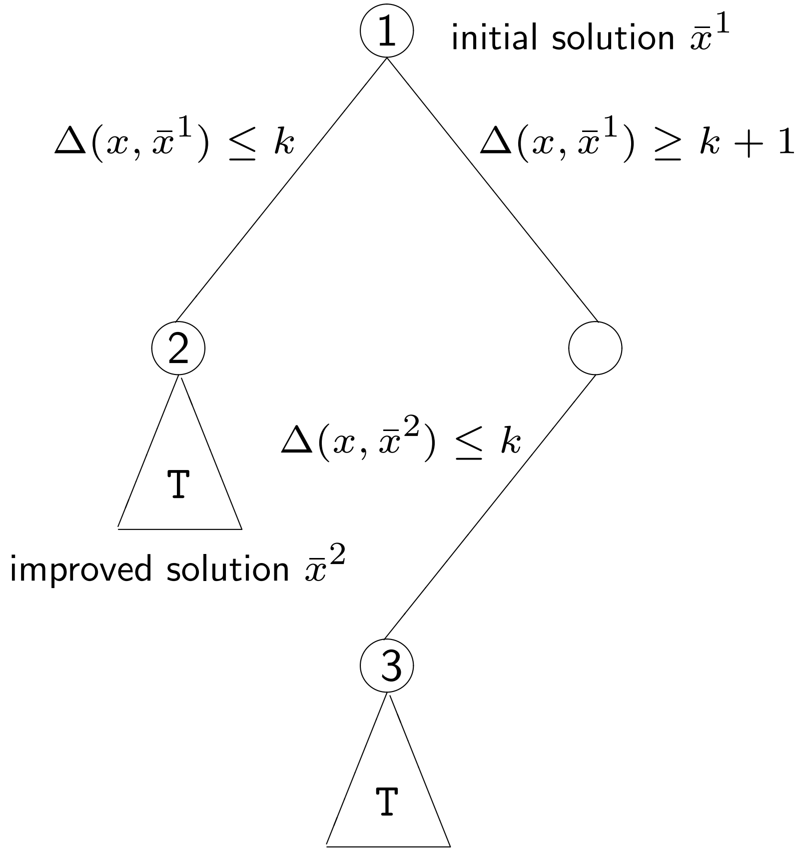
Working with a node time limit: case (a)



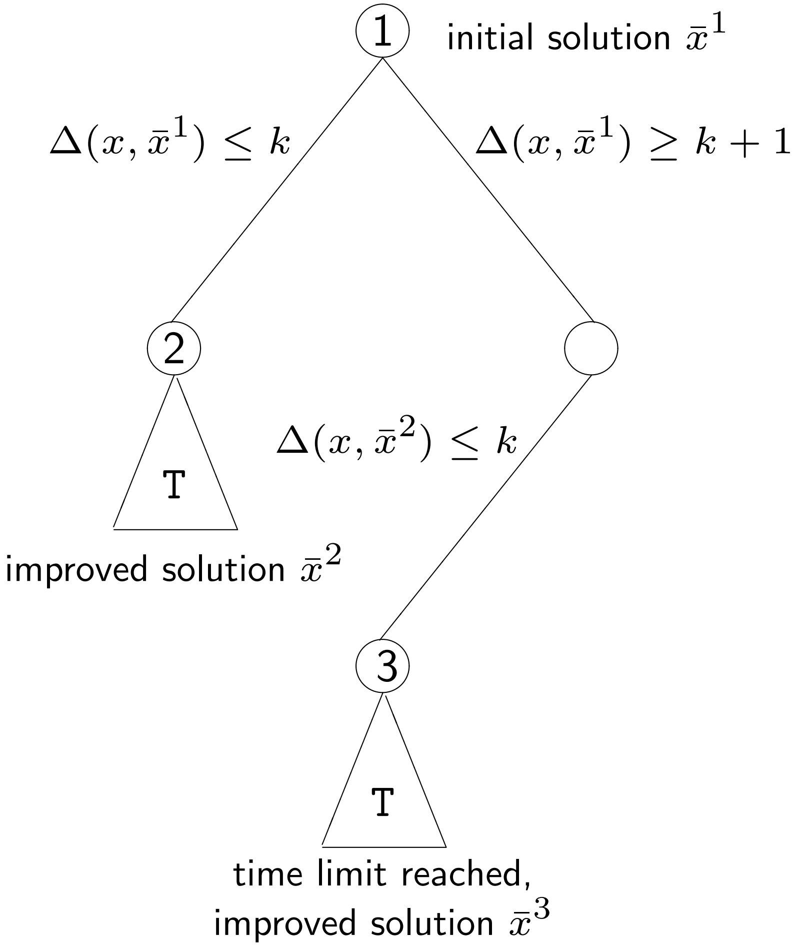
Working with a node time limit: case (a)



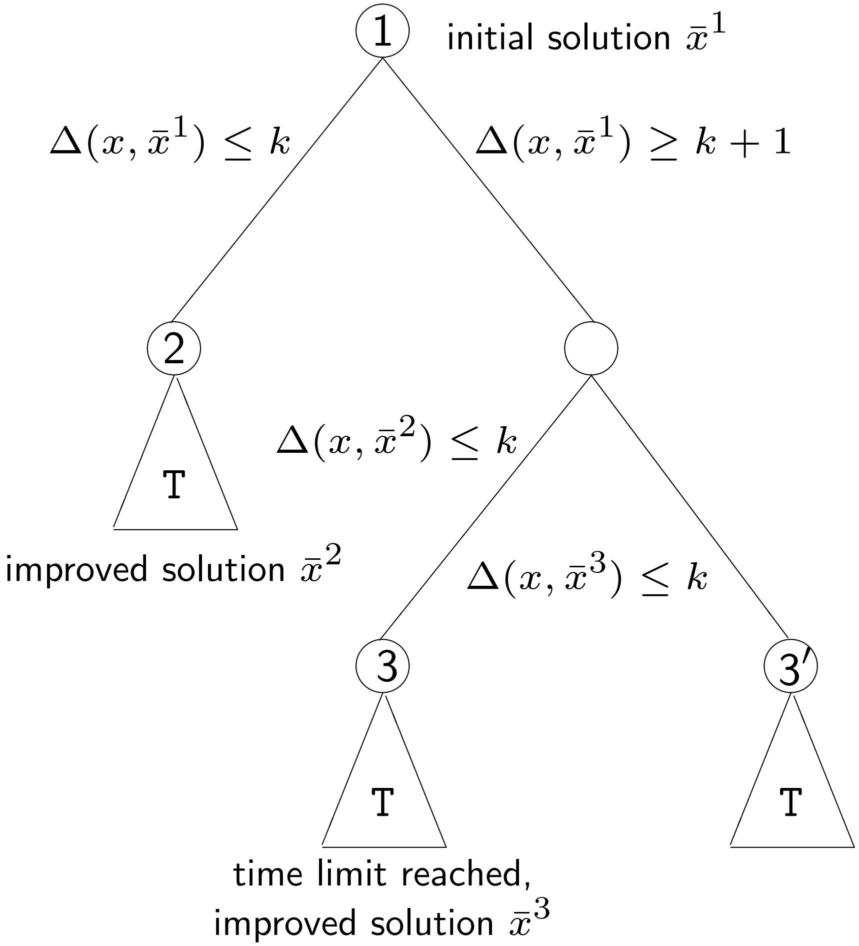
Working with a node time limit: case (a)



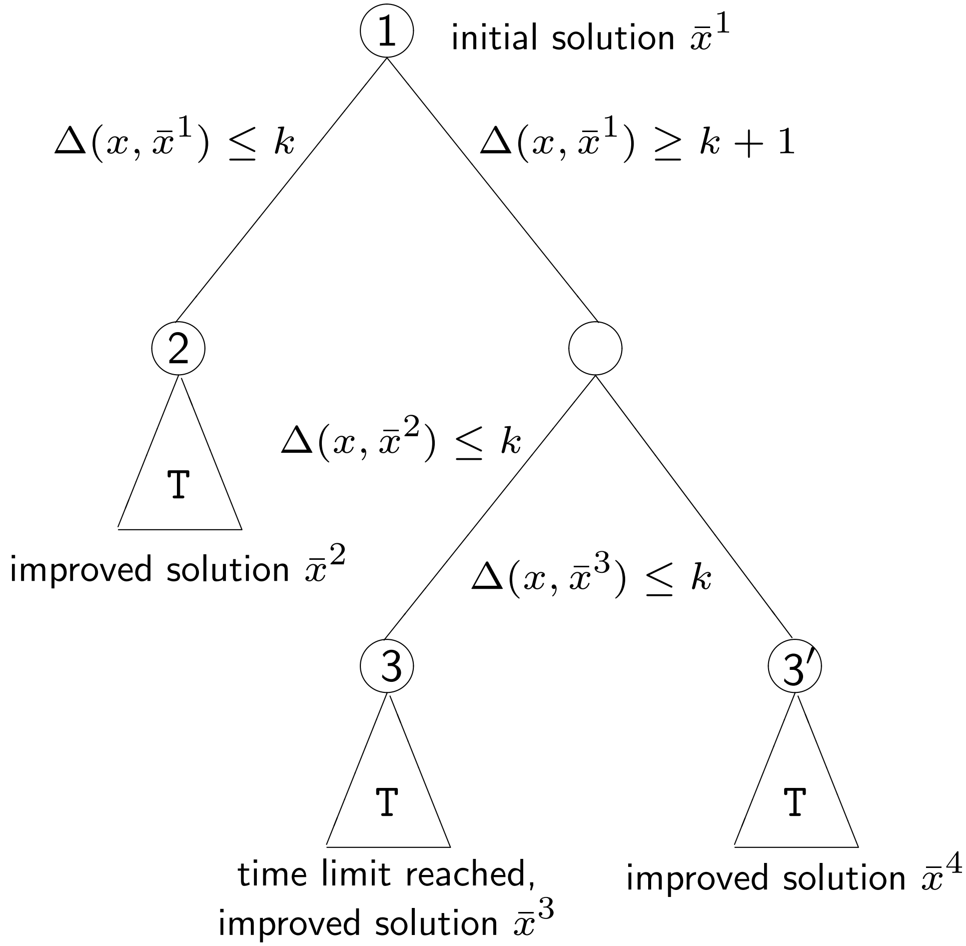
Working with a node time limit: case (a)



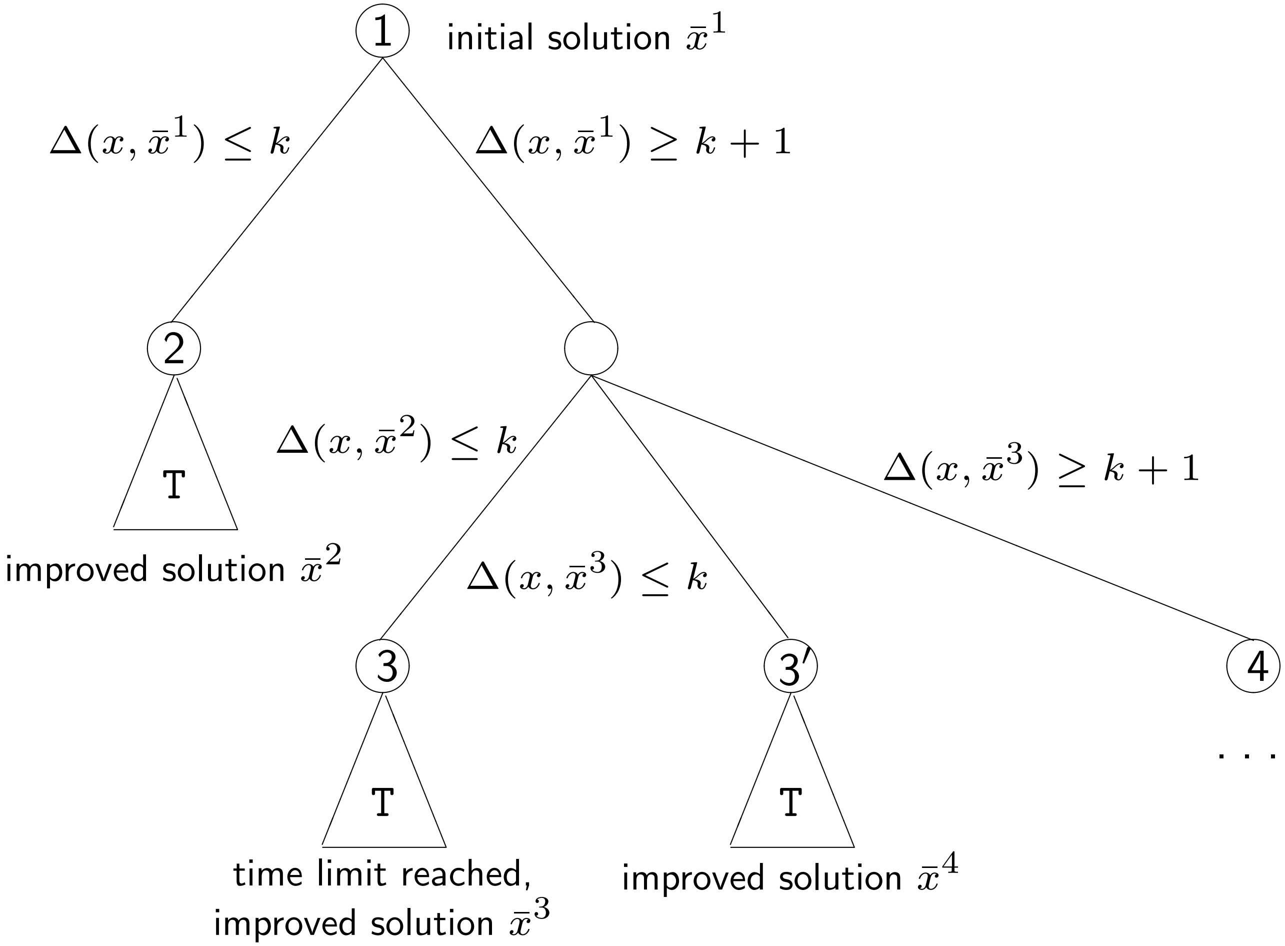
Working with a node time limit: case (a)



Working with a node time limit: case (a)



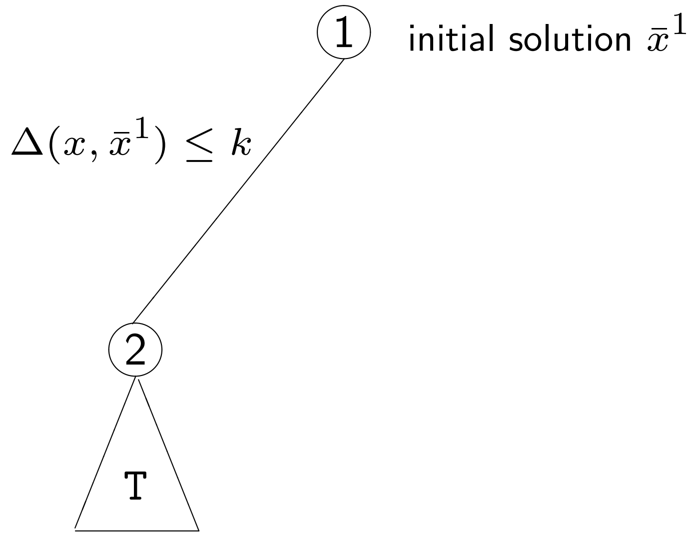
Working with a node time limit: case (a)



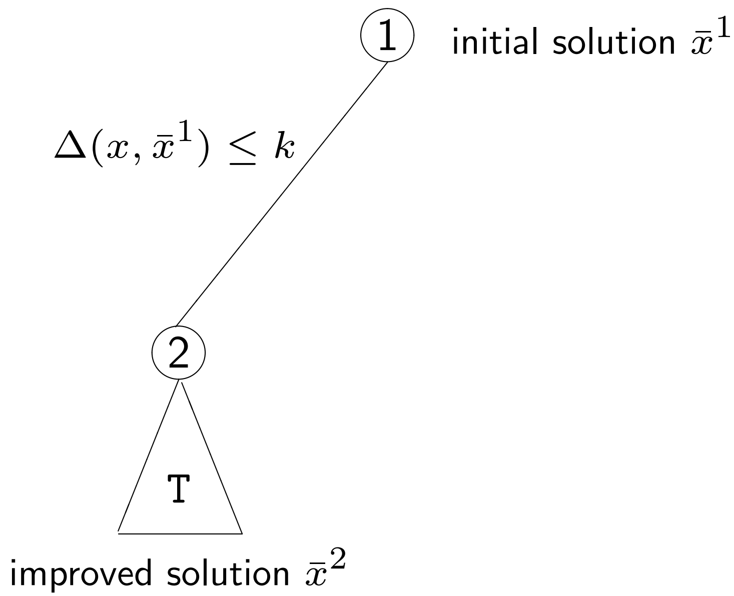
Working with a node time limit: case (b)

1 initial solution $\bar{x}^1$

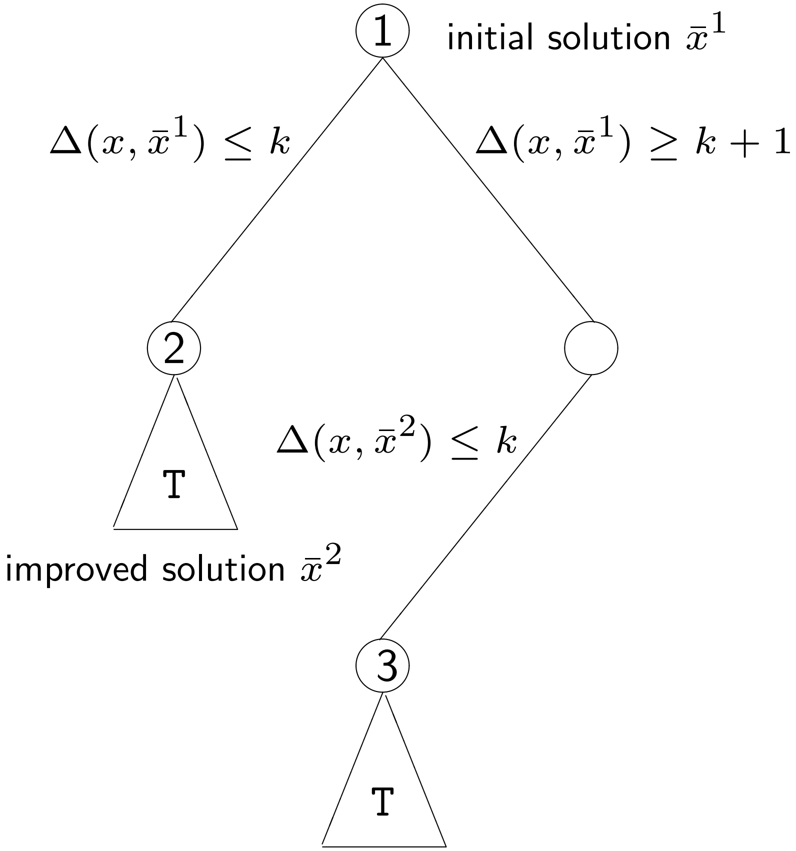
Working with a node time limit: case (b)



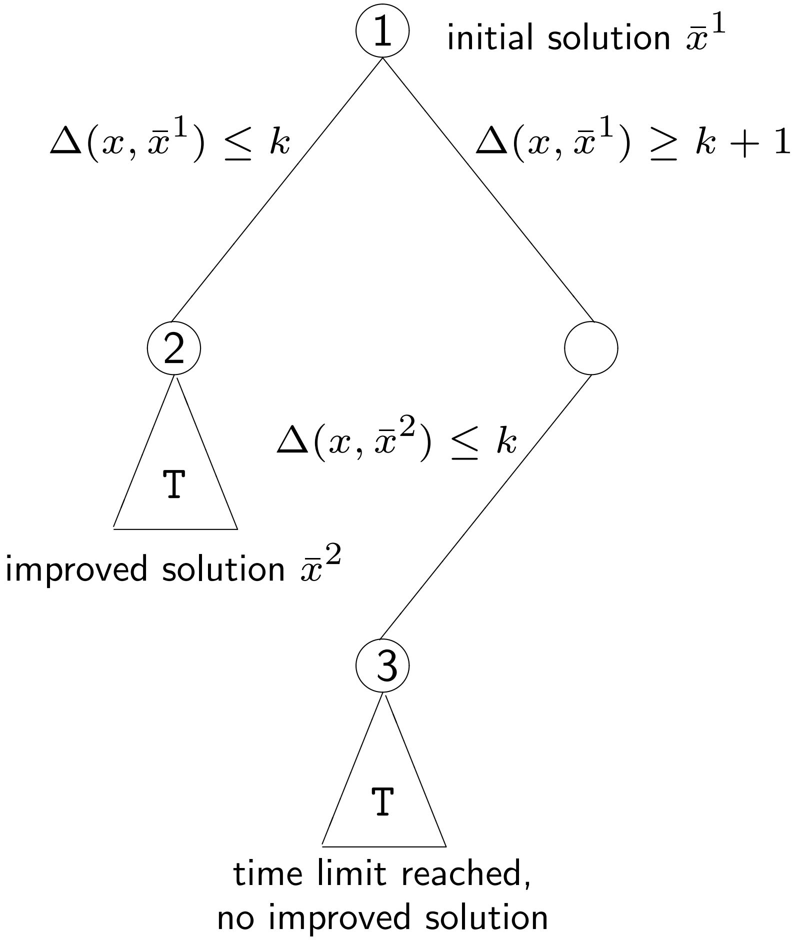
Working with a node time limit: case (b)



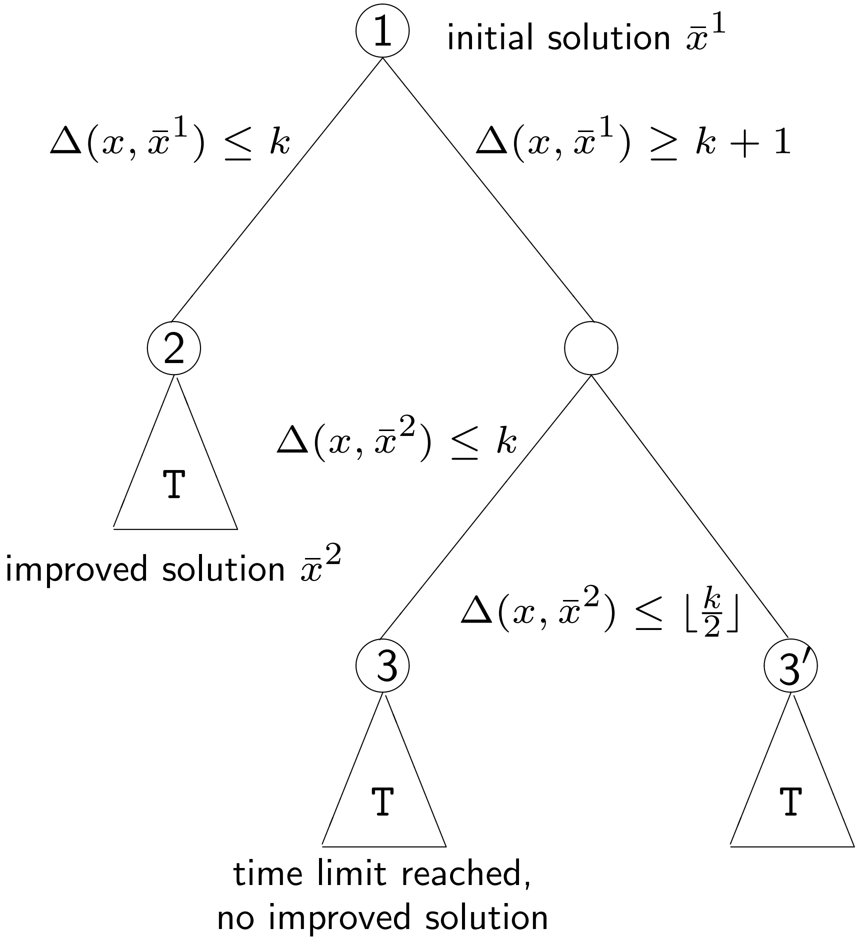
Working with a node time limit: case (b)



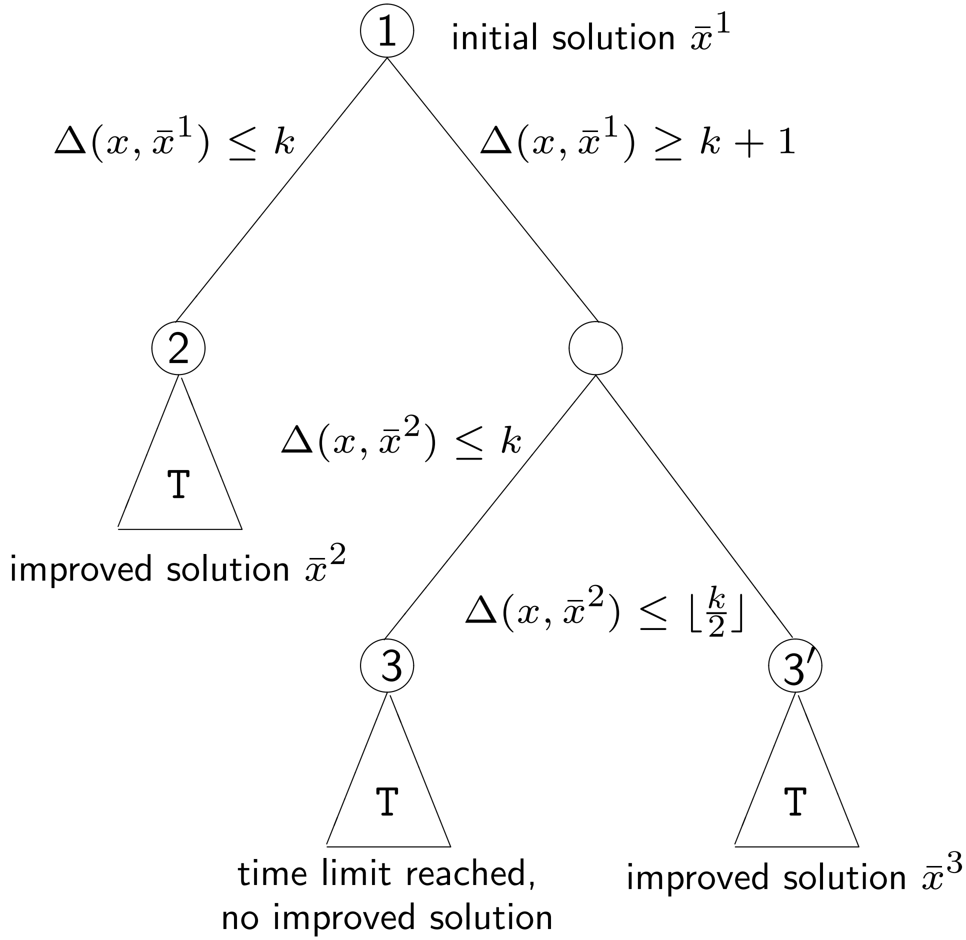
Working with a node time limit: case (b)



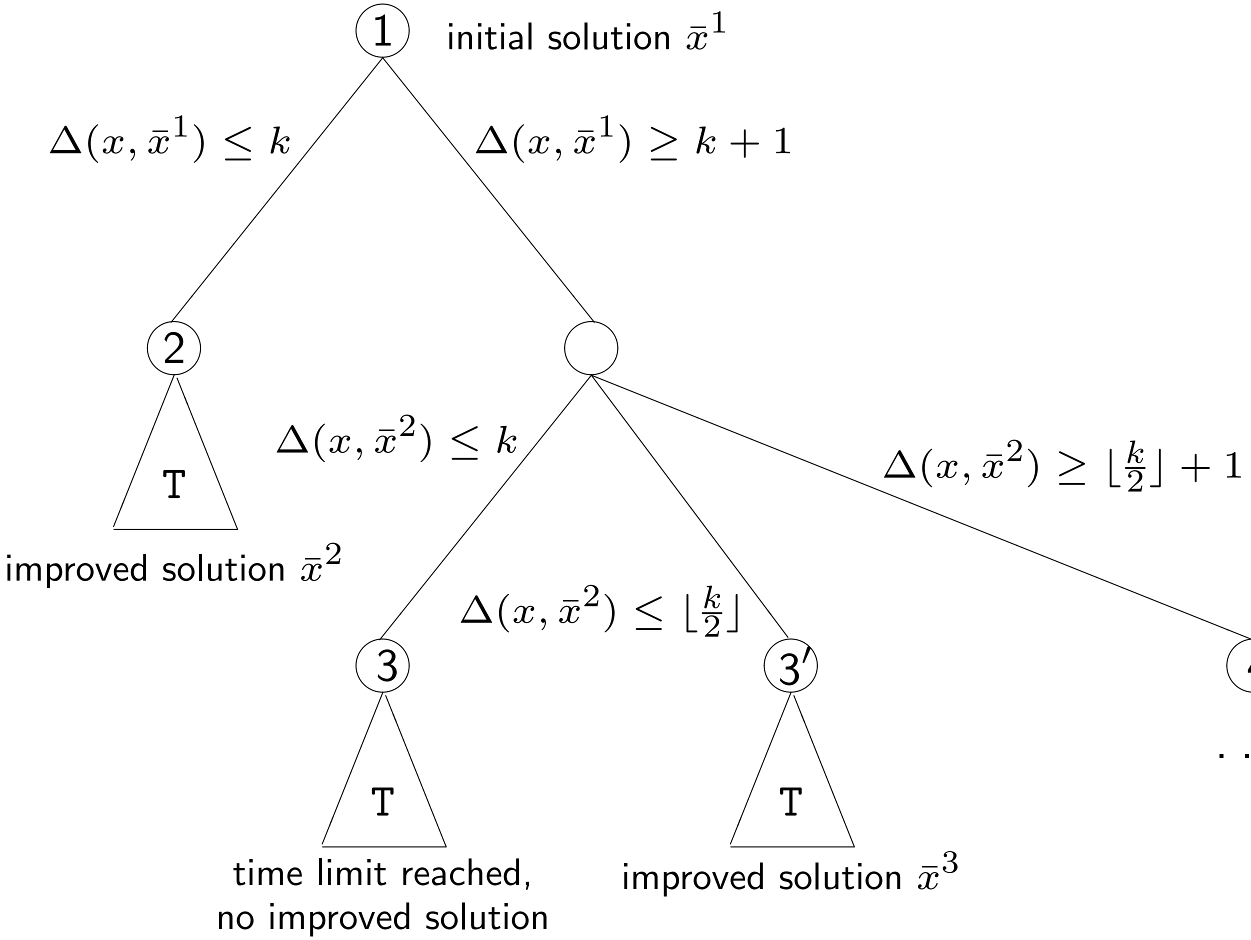
Working with a node time limit: case (b)



Working with a node time limit: case (b)



Working with a node time limit: case (b)



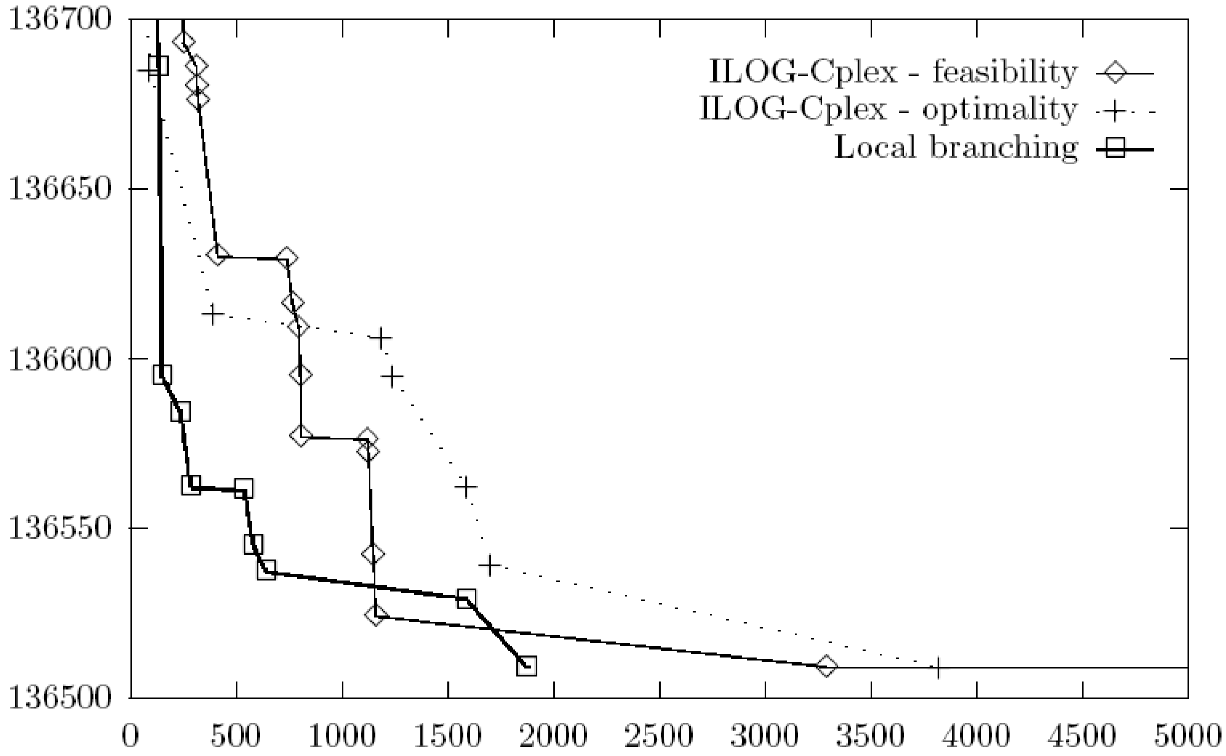


Figure 2: Solving MIP instance `tr24-15` (solution value vs. CPU seconds).

Local branching in a heuristic solution framework

Easy adaptation of the previous framework: in case of stalling, use a **diversification** mechanism to find a (worse) solution x^{h+1} to replace the current-best solution x^h, and continue

- Diversification by **Variable Neighbourhood Search** (Hansen & Mladenovic, 1998):

  Find a solution x^{h+1} close enough to x^h, but outside the current k-OPT neighbourhood

- **Implementation:** run the black-box solver (initial upper bound $= +\infty$) to find the first feasible solution x^{h+1} of the current problem amended by the **diversification constraint**

$$k+1 \leq \Delta(x, x^h) \leq k + k/2$$

"Akin to a random 3-OPT move after several 2-OPT moves for TSP"

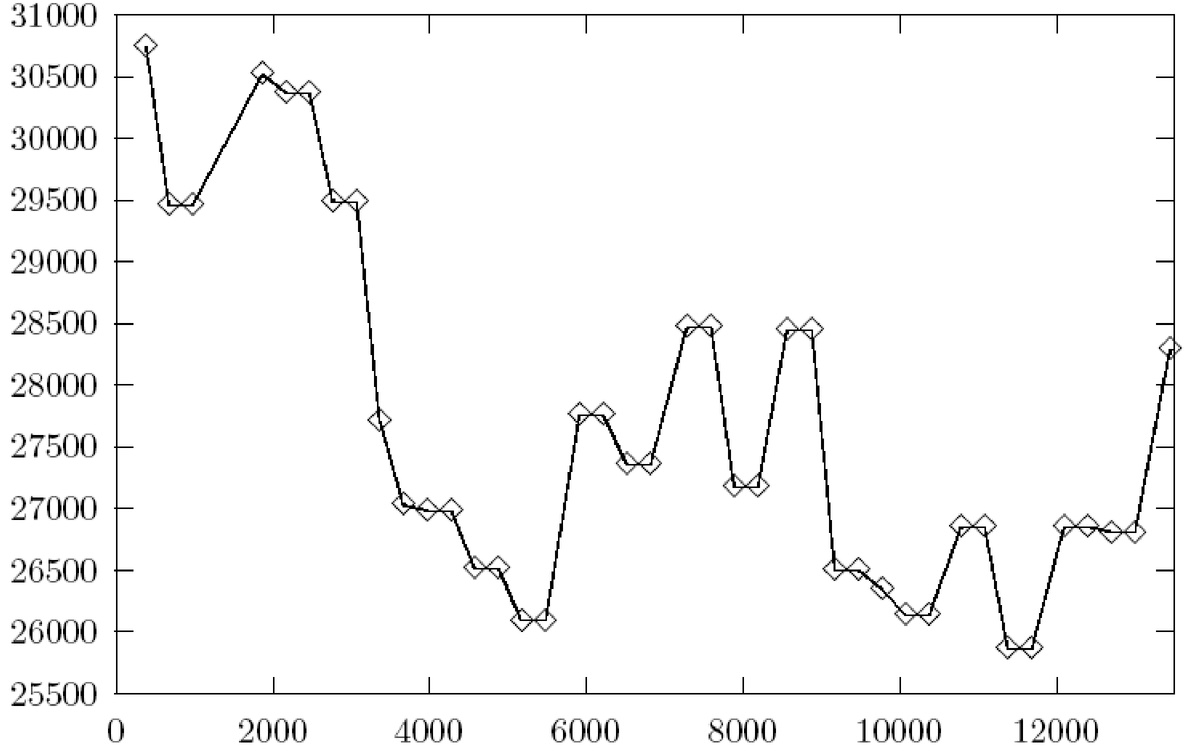


Figure 6: LocBra acting as a heuristic for instance B1C1S1 (solution value vs. CPU seconds).

Computational performance in a heuristic context

Very good computational performance reported in the recent literature

- M. Fischetti, A. Lodi, "Local Branching", Mathematical Programming A, 98, 23-47, 2003
- M. Fischetti, C. Polo, M. Scantamburlo, "A Local Branching Heuristic for Mixed-Integer Programs with 2-Level Variables", Networks 44 (2), 61-72, 2004
- P. Hansen, N. Mladenovìc, D. Urosevic, "Variable Neighborhood Search and Local Branching", Les Cahiers du GERAD, June 2004.

Related methodologies inspired by the local branching paradigm:

- E. Danna, E. Rothberg, C. Le Paper, "Exploring relaxation induced neighborhoods to improve MIP solutions", Mathematical Programming A, 102, 71–90, 2005

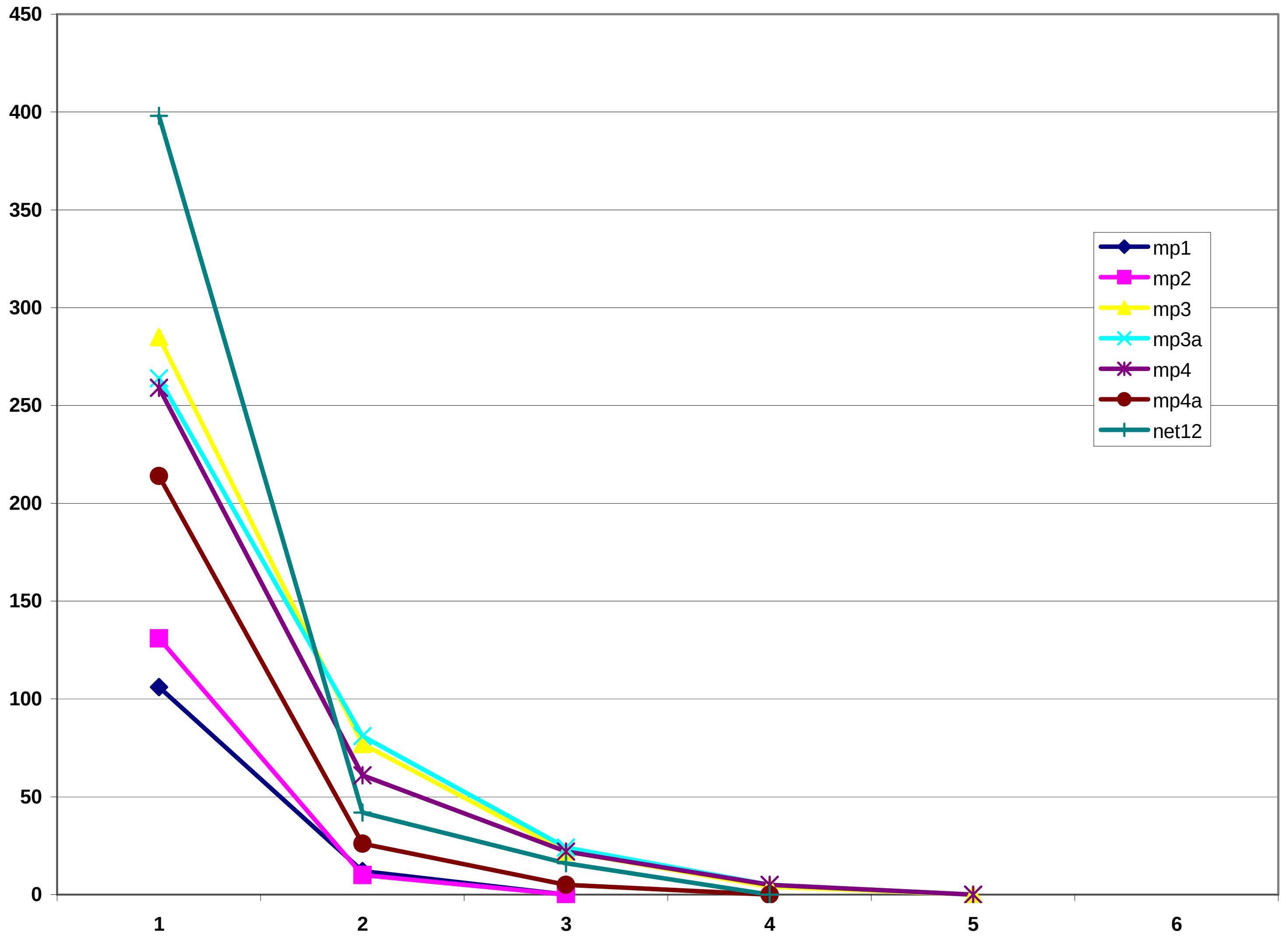
Towards feasibility… and beyond

- Instances for which even finding a first **feasible** solution is extremely hard in practice, hence the local branching framework (as stated) cannot be initialized in a proper way…

  [Relaxed model]: **relax** the MIP model by introducing **artificial variables** with big-M coefficients in the objective function

- The **"to feasibility and beyond"** solution approach:

  1. define an **infeasible** solution x^H, e.g., by rounding the optimal LP sol.
  2. **relax** the MIP model by introducing an **artificial variable** (with big-M coefficient in the objective function) for each constraint violated by x^H
  3. apply the standard **local branching** framework starting from x^H



Infeasibility reduction starting from the rounded LP solution

The Feasibility Pump

joint work with F. Glover (Univ. Colorado at Boulder, USA) and A. Lodi (DEIS, Univ. Bologna)

Motivation

- In some important practical cases, state-of-the-art MIP solvers may spend a very large computational effort before initializing their incumbent solution.
- We concentrate on heuristic methods to find a feasible solution for hard MIPs.
- This issue became even more important in the recent years, due to the success of local-search approaches for general MIPs such as *local branching* [Fischetti & Lodi, 2002]

  and *RINS* and *guided dives* [Danna, Rothberg, Le Pape, 2003]
- Indeed, these methods can only be applied if an initial feasible solution is known.

$\boxed{\text{Hence: the earlier a feasible solution is found, the better!}}$

The basic scheme

- How do you define feasibility for a MIP problem of the form:

$$\min\{c^T x : Ax \geq b, x_j \text{ integer } \forall j \in \mathcal{I}\} \quad ?$$

- We propose the following definition:

  a feasible solution is a point $x^* \in P := \{x : Ax \geq b\}$ that is coincident with its rounding $\widetilde{x}$

  where:

  1. $[\cdot]$ represents scalar rounding to the nearest integer;
  2. $\widetilde{x}_j := [x^*_j]$ if $j \in \mathcal{I}$; and
  3. $\widetilde{x}_j := x^*_j$ otherwise.

- Replacing coincident with as close as possible relatively to a suitable distance function $\Delta(x^*, \widetilde{x})$ suggests an iterative heuristic for finding a feasible solution of a given MIP.

The basic scheme (cont.d)

- We start from any $x^* \in P$, and define its rounding $\widetilde{x}$.
- At each iteration we look for a point $x^* \in P$ which is as close as possible to the current $\widetilde{x}$ by solving the problem:

$$\min\{\Delta(x, \widetilde{x}) : x \in P\}$$

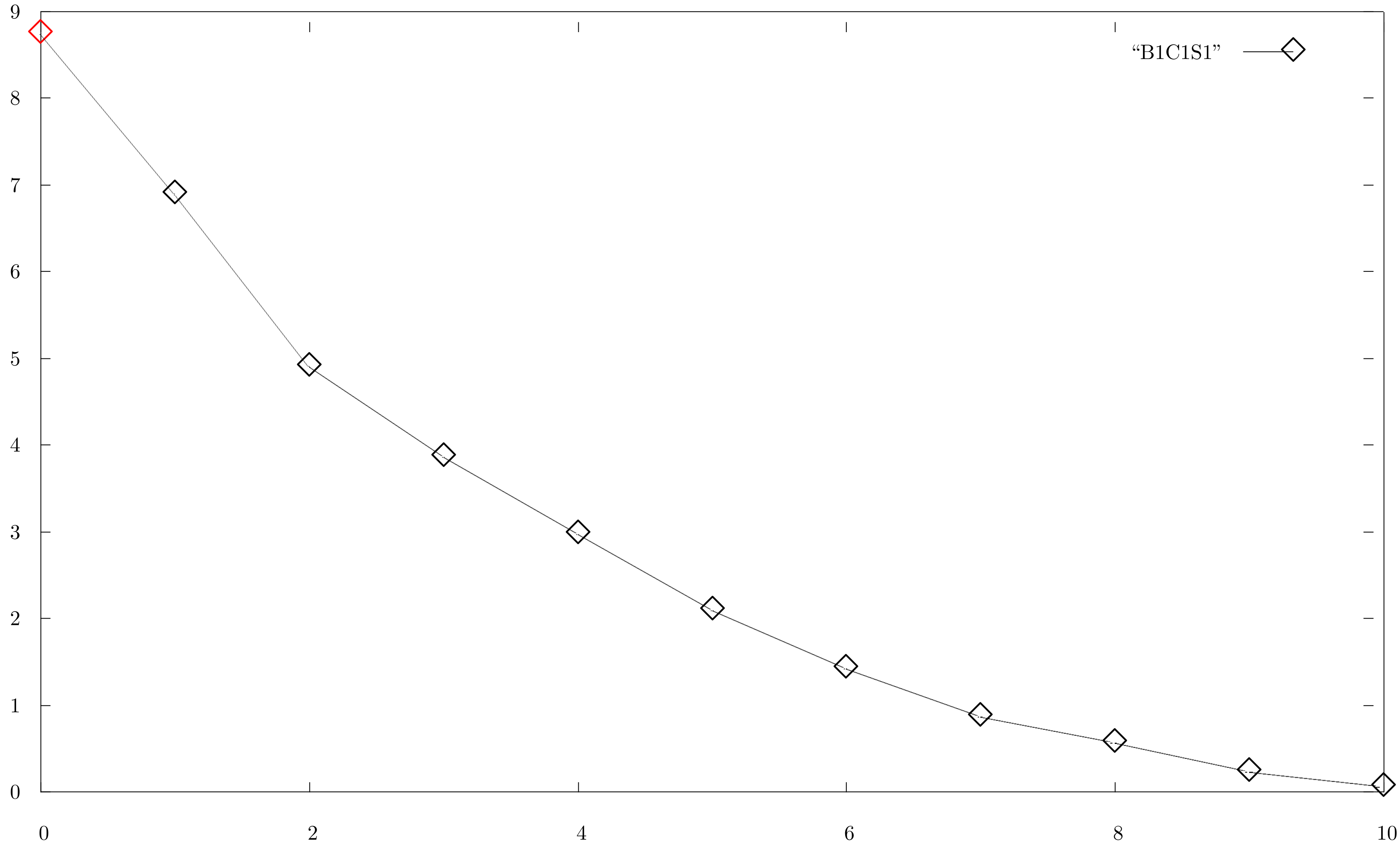
  Assuming $\Delta(x, \widetilde{x})$ is chosen appropriately, is an easily solvable LP problem.

- If $\Delta(x^*, \widetilde{x}) = 0$, then x^* is a feasible MIP solution and we are done.
- Otherwise, we replace $\widetilde{x}$ by the rounding of x^*, and repeat.

- From a geometric point of view, this simple heuristic generates two hopefully convergent trajectories of points x^* and $\widetilde{x}$ which satisfy feasibility in a complementary but partial way:

  1. one, x^*, satisfies the linear constraints,
  2. the other, $\widetilde{x}$, the integer requirement.

Plot of the infeasibility measure $\Delta(x^*, \widetilde{x})$ at each iteration



Definition of $\Delta(x^*, \widetilde{x})$

- We consider the L_1-norm distance between a generic point $x \in P$ and a given integer $\widetilde{x}$, defined as:

$$\Delta(x, \widetilde{x}) = \sum_{j \in \mathcal{I}} |x_j - \widetilde{x}_j|$$

- Assuming (for the sake of notation) that all integer-constrained variables are binary, $\Delta(x^*, \widetilde{x})$ attains the simple form:

$$\Delta(x, \widetilde{x}) := \sum_{j \in \mathcal{I} : \widetilde{x}_j = 0} x_j + \sum_{j \in \mathcal{I} : \widetilde{x}_j = 1} (1 - x_j) \qquad (1)$$

- Given an integer $\widetilde{x}$, the closest point $x^* \in P$ can therefore be determined by solving the LP:

$$\min\{\Delta(x, \widetilde{x}) : Ax \geq b\} \qquad (2)$$

A basic FP implementation

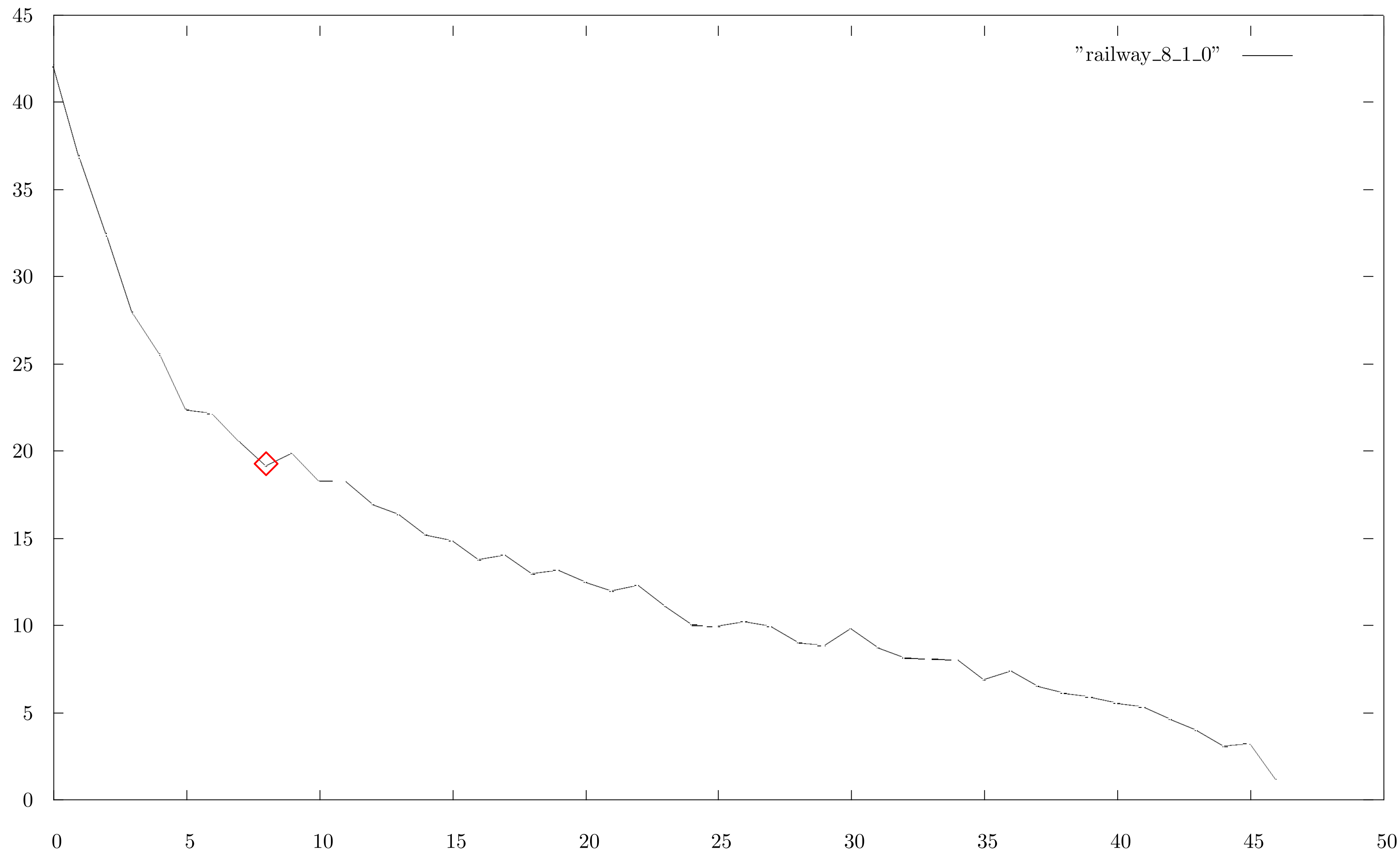
1. initialize nIT := 0 and $x^* := \text{argmin}\{c^T x : Ax \geq b\}$;
2. if x^* is integer, return(x^*);
3. let $\widetilde{x} := [x^*]$ (= rounding of x^*);
4. while (time < TL) do
5.   let nIT := nIT $+ 1$ and compute $x^* := \text{argmin}\{\Delta(x, \widetilde{x}) : Ax \geq b\}$;
6.   if x^* is integer, return(x^*);
7.   if $[x^*] \neq \widetilde{x}$ then
8. $\widetilde{x} := [x^*]$

    else

9.   flip the TT = rand(T/2,3T/2) entries $\widetilde{x}_j$ $(j \in \mathcal{I})$ with highest $|x_j^* - \widetilde{x}_j|$
10.   endif
11. enddo

- Step 9 (stalling): we modify $\widetilde{x}$, even if this increases its distance from x^*.

Plot of the infeasibility measure $\Delta(x^*, \widetilde{x})$ at each pumping cycle



Summary of the computational results

- `ILOG-Cplex` 8.1 is run on default version but avoiding preprocessing (following the suggestion of Ed Rothberg).
- `FP` solves LPs by leaving `ILOG-Cplex` decide which is the best algorithm (`CPXoptimize`).
- Over 83 hard 0-1 MIP instances in the MIPLIB test-bed:

  `FP` failed in finding a feasible solution only in 3 case, while

  `ILOG-Cplex` 8.1 failed 19 times.
- The quality of the solutions obtained is generally comparable, as well as the computing times.

Mipping Chvàtal-Gomory cuts

Matteo Fischetti
University of Padova, Italy

Andrea Lodi
University of Bologna, Italy
IBM, T.J. Watson Research Center

Notation and definitions

- We consider an Integer Linear Program (ILP) of the form:

$$\min\{c^T x : Ax \le b, x \ge 0 \text{ integer}\}$$

and two associated polyhedra:

$$P := \{x \in \mathbb{R}^n_+ : Ax \le b\}$$
$$P_I := conv\{x \in Z^n_+ : Ax \le b\} = conv(P \cap Z^n)$$

- A *Chvàtal-Gomory* (*CG*) *cut* is a valid inequality for P_I of the form:

$$\lfloor u^T A \rfloor x \le \lfloor u^T b \rfloor$$

where $u \in R^m_+$ is called the CG multiplier vector, and $\lfloor \cdot \rfloor$ denotes lower integer part.

- The first Chvàtal closure of P is defined as:

$$P_1 := \{x \ge 0 : Ax \le b, \lfloor u^T A \rfloor x \le \lfloor u^T b \rfloor \text{ for all } u \in \mathbb{R}^m_+\}$$

- P_1 is indeed a polyhedron, i.e., a finite number of CG cuts suffice to define it. [Chvàtal 1973]

Notation and definitions (cont.d)

- Clearly, $P_I \subseteq P_1 \subseteq P$.

- Every fractional vertex x^* of P associated with a certain basis B (say) of (A, I) can be cut off by the CG cut in which u is chosen as the i-th row of B^{-1}, where i is the row associated with any fractional component of x^*. [Gomory 1958,1963]

- In some cases, one has that $P_I = P_1$ as, e.g., for matching problems where undominated CG cuts correspond to the famous Edmonds' blossom inequalities. [Edmonds 1965]

- By the well-known equivalence between optimization and separation, we will address the Chvàtal-Gomory separation problem (CG-SEP) of the form:

  Given any point $x^* \in P$ find (if any) a CG cut that is violated by x^*, i.e., find $u \in \mathbb{R}^m_+$ such that $\lfloor u^T A \rfloor x^* > \lfloor u^T b \rfloor$, or prove that no such u exists.

- However, CG-SEP is NP-hard, so optimizing over P_1 also is. [Eisembrand 1999]

Some practical questions

- How difficult is, **in practice**, to optimize *exactly* over the first Chvàtal closure of a generic ILP?
- Which fraction of the integrality gap can be closed this way, e.g., for some hard problems in the MIPLIB library?
- **Before affording the effort of designing and implementing sophisticated separation tools, we want to be sure the overall approach has some potentials...**

MIPping CG separation

- Given the input point $x^* \geq 0$ to be separated, CG-SEP calls for a CG cut $\alpha^T x \leq \alpha_0$ which is (maximally) violated by x^*, where $\alpha = \lfloor u^T A \rfloor$ and $\alpha_0 = \lfloor u^T b \rfloor$ for a certain $u \in I\!R_+$.

- Some properties:
  1. Any variable x_j such that $x^*_j = 0$ can be omitted.
     Indeed, it does not contribute to the violation and its coefficient can be recomputed a posteriori as $\alpha_j := u^T A_j$ (no time-consuming lifting operations being needed).
  2. The same holds for variables at their upper bound in x^*, which can be complemented.
  3. It is known that one can assume $u_i < 1$ in case the i-th row of (A, b) is integer.

- Avoiding weak cuts:
  - Several equivalent solutions of the separation problem (in its optimization version) typically exist, some of which produce very weak cuts.
  - In practice, finding stronger cuts corresponds to producing "minimal" CG multiplier vectors with as few nonzero entries as possible.

- **Our approach is to model the rank-1 Chvàtal-Gomory separation problem, which is known to be NP-hard, through a MIP model, which is then solved (exactly/heuristically) through a general-purpose MIP solver.**

Mipping CG separation (cont.d)

- We then propose the following MIP model for CG-SEP:

$$\max \quad (\sum_{j \in J(x^*)} \alpha_j x_j^* - \alpha_0) - \sum_{i=1}^{m} w_i u_i \tag{1}$$

$$f_j = u^T A_j - \alpha_j, \quad \text{for } j \in J(x^*) \tag{2}$$

$$f_0 = u^T b - \alpha_0 \tag{3}$$

$$0 \le f_j \le 1 - \delta, \quad \text{for } j \in J(x^*) \cup \{0\} \tag{4}$$

$$0 \le u_i \le 1 - \delta, \quad \text{for } i = 1, \cdots, m \tag{5}$$

$$\alpha_j \text{ integer}, \quad \text{for } j \in J(x^*) \cup \{0\} \tag{6}$$

where $J(x^*) := \{j \in \{1, \cdots, n\} : x_j^* > 0\}$ is the support of x^* (possibly after having complemented some variables and updated b accordingly).

- We chose the $\delta = 0.01$ so as to improve numerical stability.

- We also introduced the penalty term $-\sum_i w_i u_i$ in the objective function (1), where $w_i = 10^{-4}$ for all i, which is aimed at favoring the "minimality" of the CG multiplier vector u.

Solving the CG-separation MIP

- Preliminary experiments where the CG-separation MIP is solved through a commercial **general-purpose** MIP solver (`ILOG-Cplex` 9.0.2)

  1. When the LP relaxation of the original ILP model is solved, we take all the violated Gomory fractional cuts that can be read from the tableau, and skip CG separation.

  2. The MIP solver for CG separation is invoked with an initial lower bound of 0.01, meaning that we are only interested in CG cuts violated by more than 0.01.

  3. At each update of the MIP incumbent solution x^*, the corresponding CG cut is stored in a pool, and added at the end of the separation phase (among the cuts with the same violation, only the one with the sparsest support is added).

  4. The MIP execution for CG separation is stopped if:
     - either the optimal solution has been found,
     - or τ branching nodes has been explored after the last x^*update.

     $\tau = 1000$ if the violation of the incumbent is less than 0.2, and $\tau = 100$ otherwise.

- We keep generating violated CG cuts of rank 1 until either no such violated cut exists (in which case we have optimized over the first closure), or because a time-limit condition is met.

Can we solve matching problems?

- We started in a "friendly" setting by addressing 2-matching problems

ID	initial LB	Optimum	# iter.s	# cuts	CPU time
eil101	619.0	623.0	26	43	9.01
gr120	6,662.5	6,694.0	33	45	10.47
pr124	50,164.0	51,477.0	124	320	555.54
gr137	66,643.5	67,009.0	11	31	1.68
pr144	32,776.0	33,652.0	39	78	9.57
ch150	6,281.0	6,337.0	59	141	71.19
rat195	2,272.5	2,297.0	85	237	202.87
kroA200	27,053.0	27,426.0	26	84	10.93
kroB200	27,347.0	27,768.0	189	558	2,249.55
ts225	115,605.0	121,261.0	323	857	4,906.48
pr226	55,247.5	57,177.0	401	901	4,077.66
gr229	127,411.0	128,353.0	78	224	219.00
gil262	2,222.5	2,248.0	105	266	372.10
a280	2,534.0	2,550.0	52	104	40.21
lin318	38,963.5	39,266.0	292	768	6,103.32

Can we solve matching problems? (cont.d)

- Some of these instances can be solved in a much shorter computing time by just applying ILOG-Cplex 9.0.2 MIP solver (by heavy branching), and obviously by considering the use of the special purpose separation of 2-matching inequalities. [Letchford, Reinelt and Theis 2004]

- However, for some hard instances a cut-and-branch approach in which we separate 100 rounds of rank-1 cuts, and we then switch to a commercial MIP solver for concluding the optimization gives promising results.

	ILOG-Cplex			cut-and-branch				
ID	% gap closed	nodes	time	# cuts	% gap closed	separation time	nodes	total time
pr124	100.0	43,125	104.17	116	62.1	27.96	1,925	37.51
kroB200	100.0	330,913	2,748.24	129	64.1	49.34	4,113	76.30
ts225	47.1	230,115	1h	250	80.7	164.77	13,552	352.35
pr226	55.0	288,901	1h	179	62.9	61.13	19,977	281.89
gr229	100.0	15,005	180.79	126	82.8	9.65	155	60.94
gil262	100.0	117,506	2,094.77	110	84.0	12.24	217	36.78
lin318	53.3	117,100	1h	187	64.9	110.69	25,953	933.97

How tight is the first closure for MIPLIB instances?

- Instances from MIPLIB, time limit of 3 hours

ID	Optimum	# iter.s	# cuts	% gap closed	time
air03	340,160.00	1	35	100.0	1.47
gt2	21,166.00	160	424	100.0	506.25
lseu	1,120.00	73	190	91.3	565.22
mitre	115,155.00	1,509	5,398	100.0	9,394.17
mod008	307.00	26	109	100.0	8.00
mod010	6,548.00	17	62	100.0	13.05
nw04	16,862.00	78	236	100.0	227.13
p0033	3,089.00	40	152	85.4	12.95
p0548	8,691.00	886	3,356	100.0	1,575.83
stein27	18.00	98	295	0.0	490.02

- A cut-and-branch approach on instance harp2 gave very interesting results:
  - 100 rounds of separation (211 rank-1 CG cuts, 53 tight at the end),
  - 1,500 CPU seconds and 400K nodes (including both cut generation and branching).
  - `ILOG-Cplex` alone required more than 15,000 CPU seconds and 7M nodes.

Beyond the first closure?

- We addressed the possibility of using our cutting plane method as a pre-processing tool, to be used to strengthen the user's formulation by exploiting cuts of Chvàtal rank larger than 1.

  This idea was evaluated by comparing two different cut preprocessors, namely:

  – *cpx*: Apply `ILOG-Cplex` 9.0.2 (with mip emphasis "move best bound") on the current ILP model, save the final root-node model (including the generated cuts) in a file, and repeat on the new model until a total time limit is exceeded.

  – *cpx-cg*: Apply `ILOG-Cplex` 9.0.2 (with mip emphasis "move best bound") on the current ILP model, followed by 600 seconds of our CG separation procedure; then save in file the ILP model with all the cuts that are active in the last LP solution, and repeat on the new model until a total time limit is exceeded.

- For the first time we found a provable optimal solution of value 51,200.00 for the very hard instance nsrand-ipx.

- Precisely, *cpx-cg* ran for 4,800 CPU seconds obtaining a tightened formulation that brought the initial LP bound from 49,667.89 to 50,665.71.

Beyond the first closure? (cont.d)

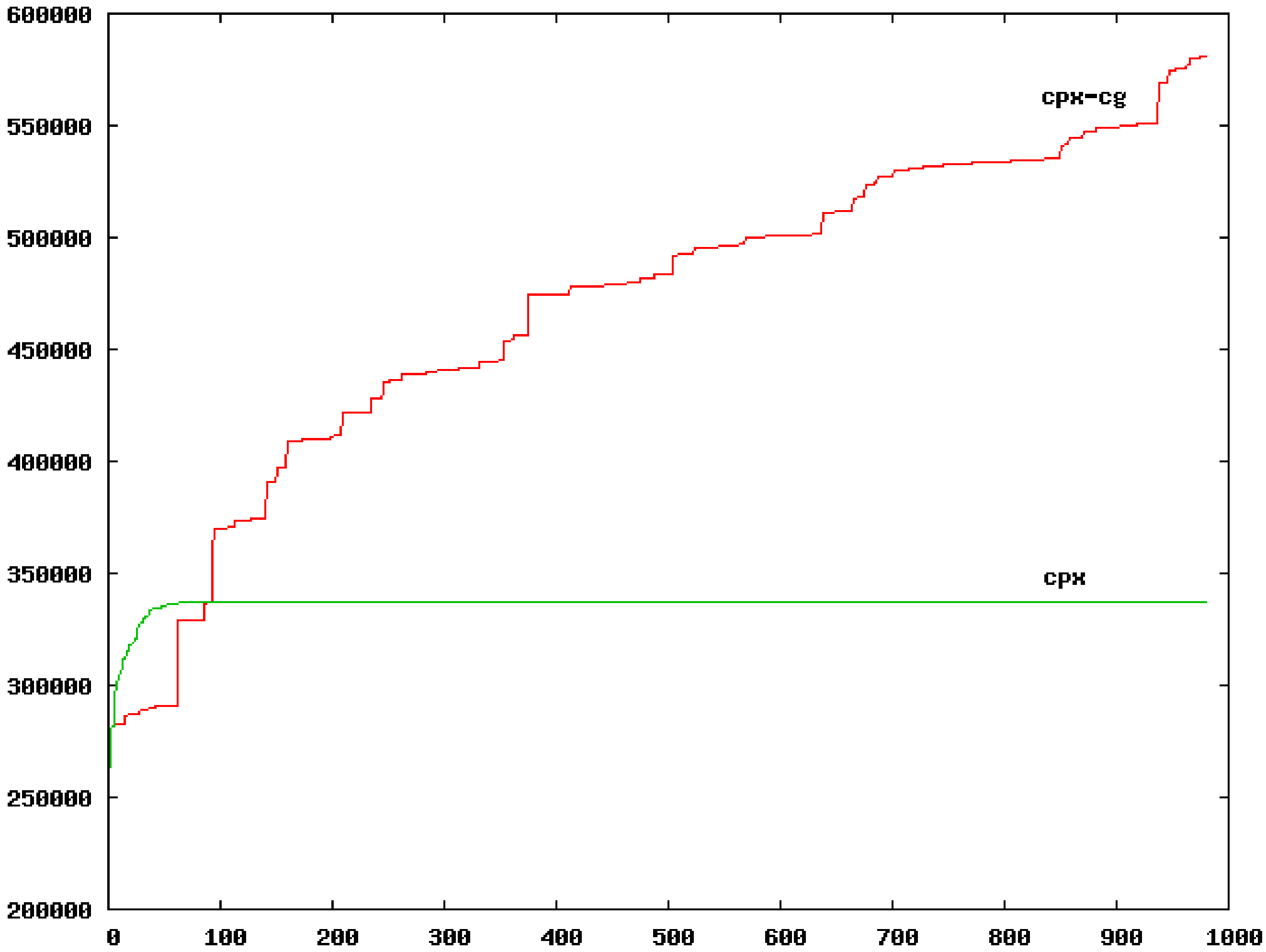


Figure 1: Lower bounds provided by *cpx* and *cpx-cg* after each call of the separation procedures, for the hard MIPLIB instance *timtab1*.

Can we discover new classes of strong inequalities?

- As in the spirit of PORTA, we can use the framework for obtaining off-line the facial structure of a specific problem. Advantage: we are not restricted to instances of very small size.

- To illustrate a possible application to the Asymmetric Travelling Salesman Problem (ATSP), we took a partial ATSP formulation including out- and in-degree equations, plus the SECs on 2-node sets, i.e., the NP-hard *Asymmetric Assignment Problem* (AAP) relaxation. [Balas 1989]

- We applied the method to ry48p from ATSPLIB, and we stored the CG cuts along with the associated CG multipliers.

- Through a careful analysis of one returned cut we have that:

$$\alpha = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} ; \alpha_0 = 5 \tag{7}$$

which is (by computational methods) facet-defining for ATSP. Using *clique lifting* we can then obtain a large class of ATSP facets, that to the best of our knowledge is new.

Conclusion and future work

- We have been able to show computationally the quality of the Chvàtal-Gomory cuts and to answer (at least partially) to several natural questions about their **practical** effectiveness.
- Although an NP-hard problem has to be solved to separate inequalities of rank 1, the issue of generating those cuts is affordable in practice and it definitely deserves attention.
- An obvious issue for future research is the design of more specific separation procedures for CG cuts, i.e., **ad-hoc heuristics for the corresponding MIP model**.

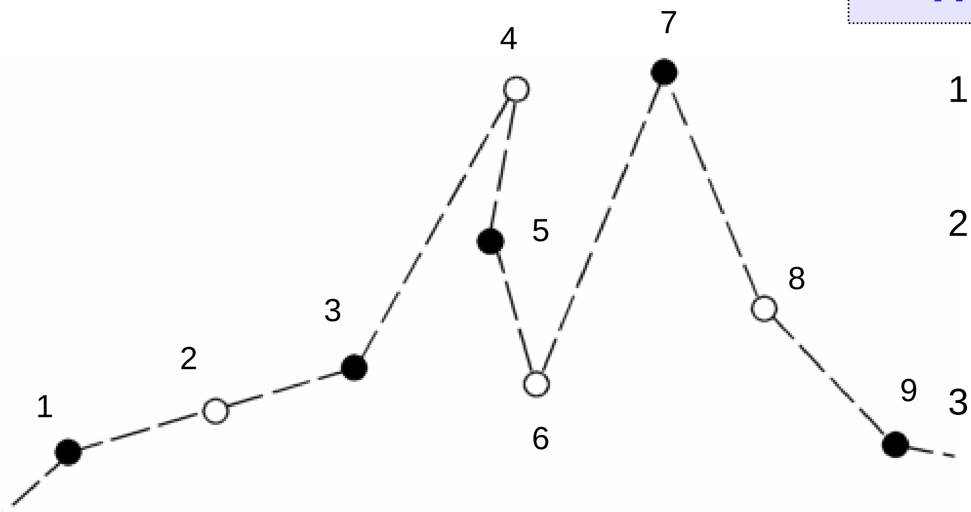
A new heuristic algorithm for the *Vehicle Routing Problem*

Roberto De Franceschi, DEI, University of Padua

Matteo Fischetti, DEI, University of Padua

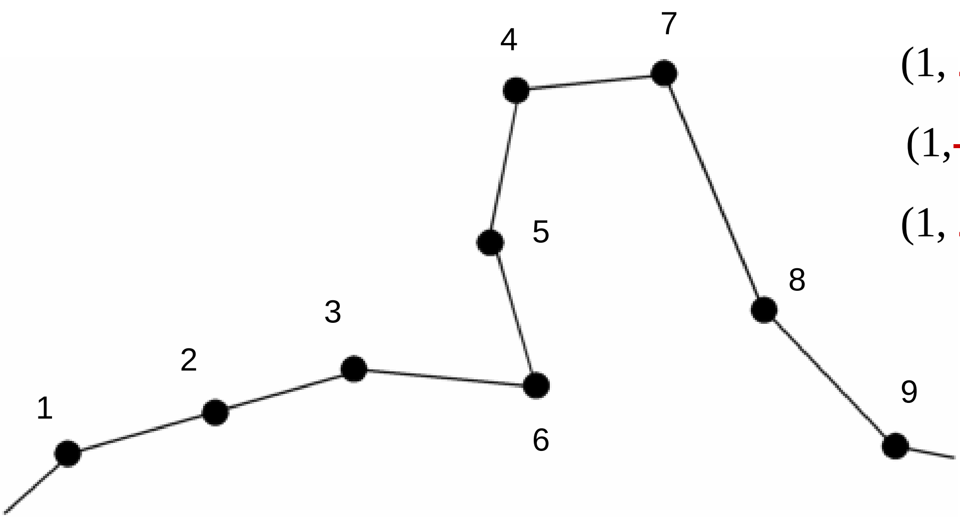
Paolo Toth, DEIS, University of Bologna

A method for the TSP (Sarvanov and Doroshko, 1981)



The ASSIGN *neighborhood*

1. consider a **given** tour as a sequence of nodes
2. fix the nodes in **odd** position, and remove the nodes in **even** position
3. Reassign the removed nodes in optimal way—an easy-solvable min-cost **assignment problem**



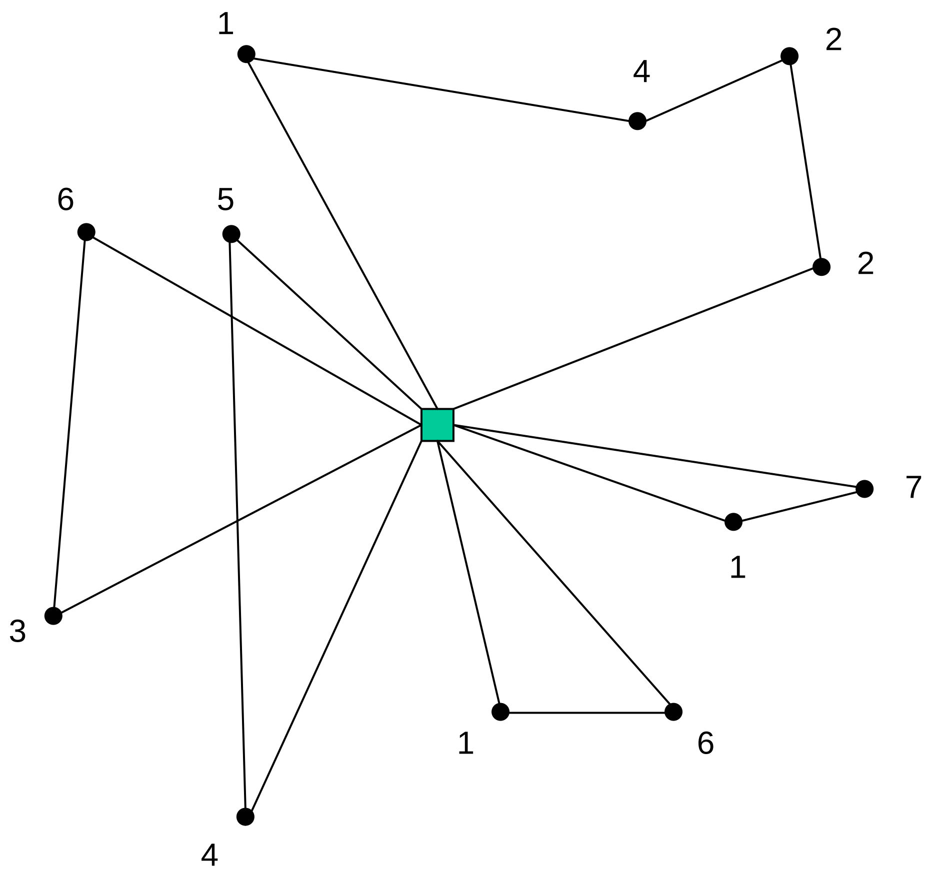
(1, **2**, 3, **4**, 5, **6**, 7, **8**, 9, …)

(1,**--**, 3, **--**,5, **--**,7, **--**, 9, …)

(1, **2**, 3, **6**, 5, **4**, 7, **8**, 9,…)

Neighborhood of **exponential cardinality** searchable in polynomial time, recently studied by:
Deineko and Woeginger (2000)
Firla, Spille and Weismantel (2002)

Capacitated Vehicle Routing Problem



Input

Depot

K vehicles

each with capacity *C*

N customers

with known demand d_i

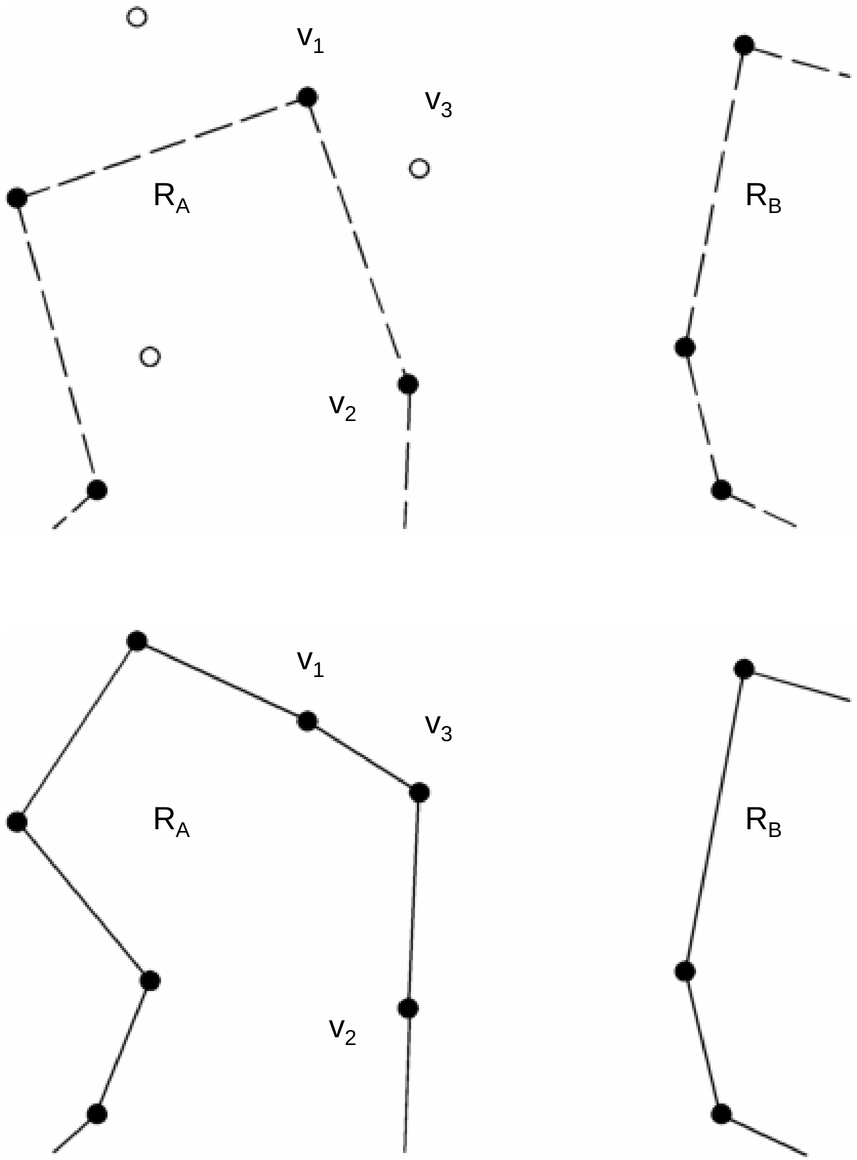
Goal

K routes

not exceeding the given capacity

with minimum total cost

Basic extensions – Part I



Issue …

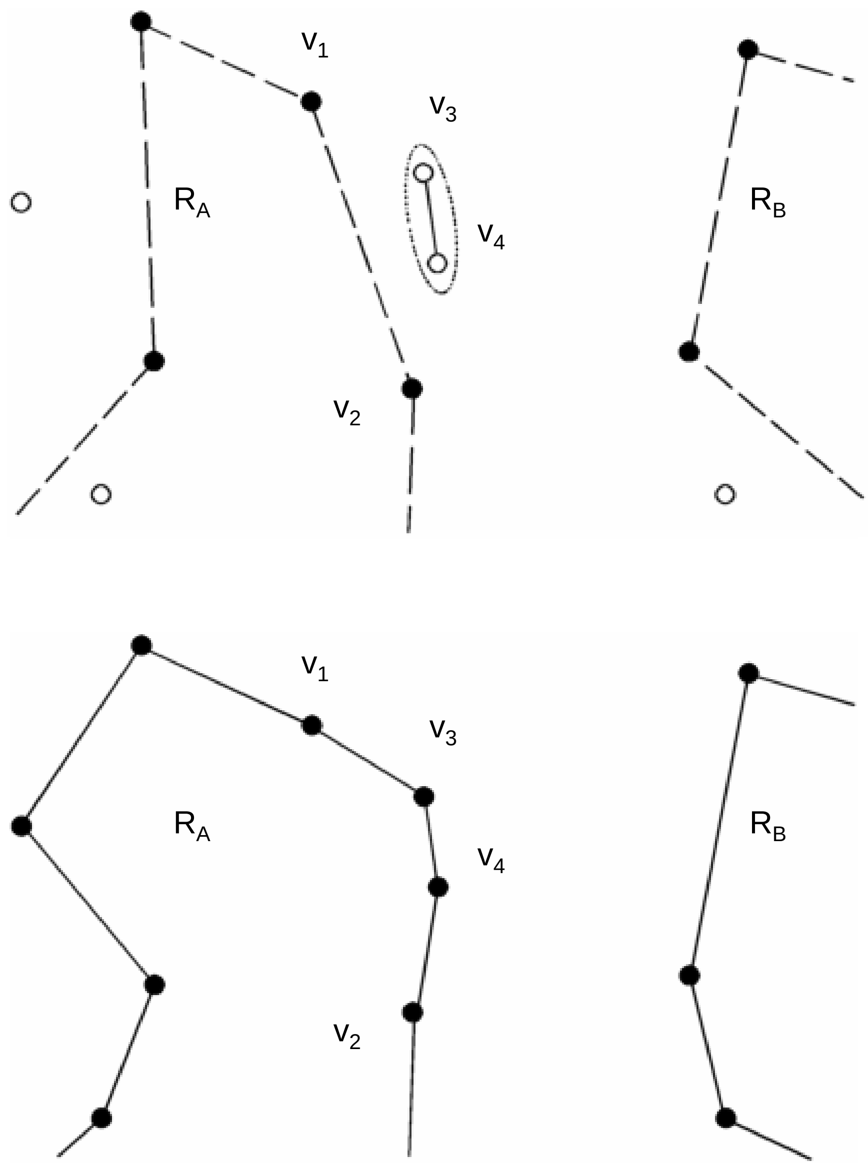
It seems useful to "move" node v_3 to route R_A (assuming this is feasible w.r.t.the capacity constraints)

But … this cannot be done by a simple position-exchange between nodes

… solution

Introduce the concepts of ***restricted solution*** and ***insertion point***

Basic extensions – Part II



Issue …

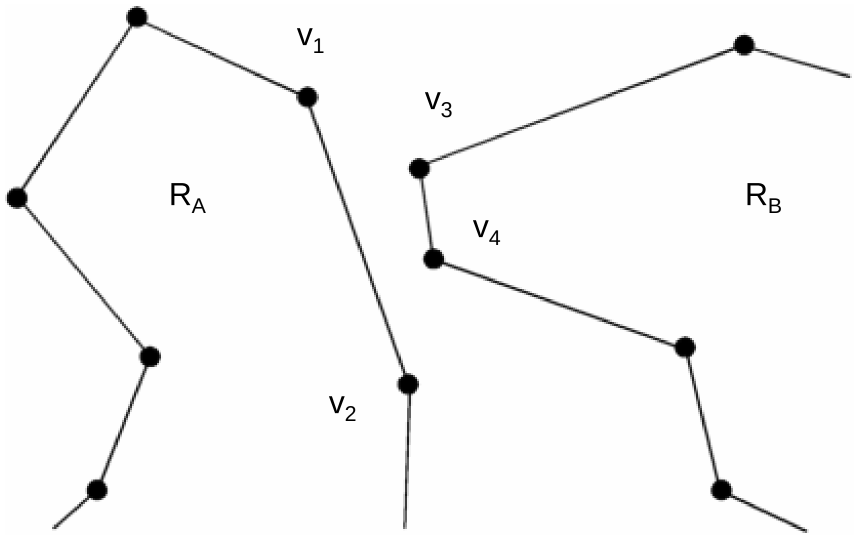
It seems useful to "move" **both** v_3 and v_4 to R_A (if feasible)

But … this cannot be done in one step by only "moving" single nodes

… solution

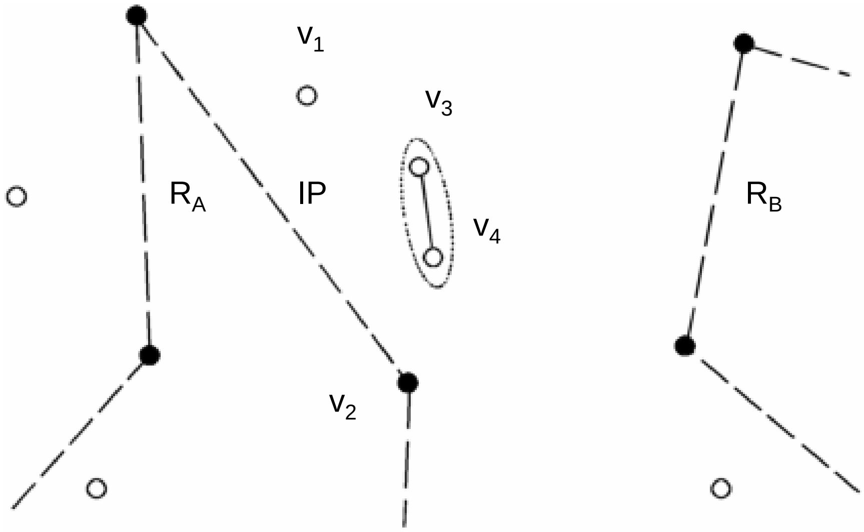
go beyond the basic odd/even scheme and introduce the notion of **extracted node sequences**

Basic extensions – Part III



Issue …

It is not possible to insert ***both*** v_1 and v_3-v_4 into the insertion point IP



… solution

generate a (possibly large) number of ***derived sequences*** through extracted nodes

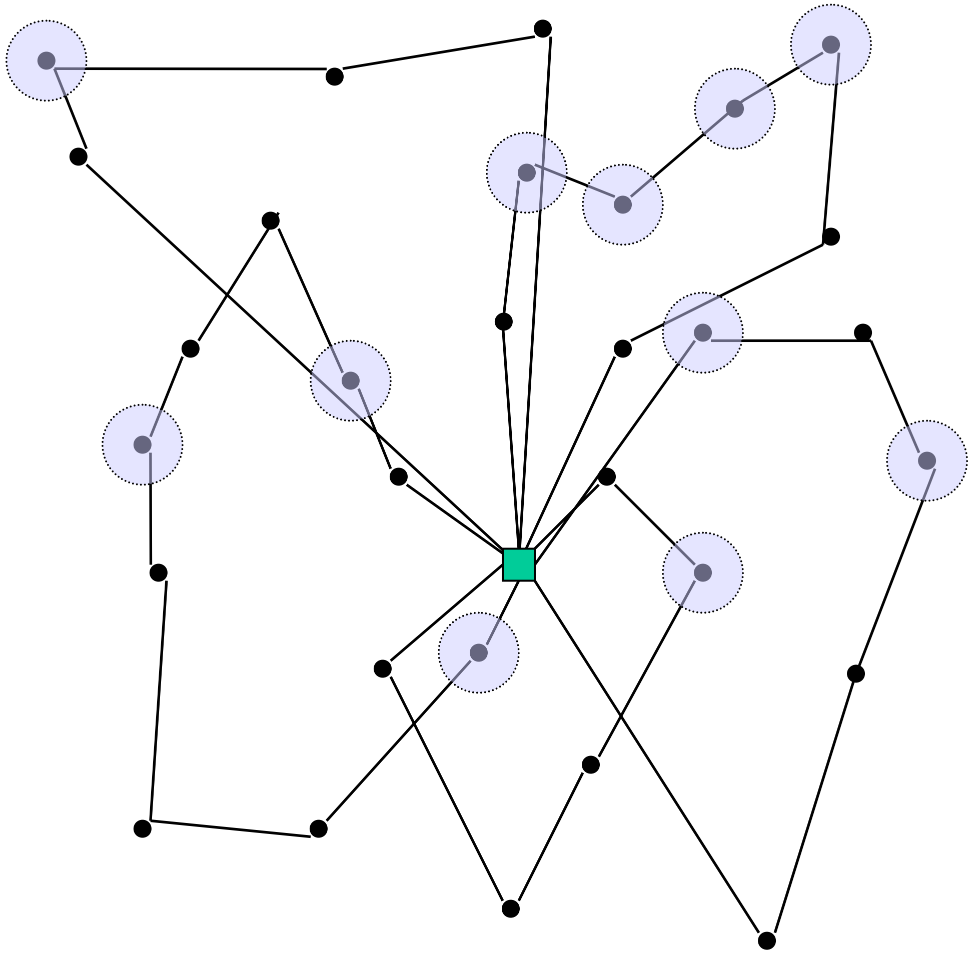
In the example, it is useful to generate the sequence v_1-v_3-v_4 to be placed in the insertion point IP

The SERR algorithm

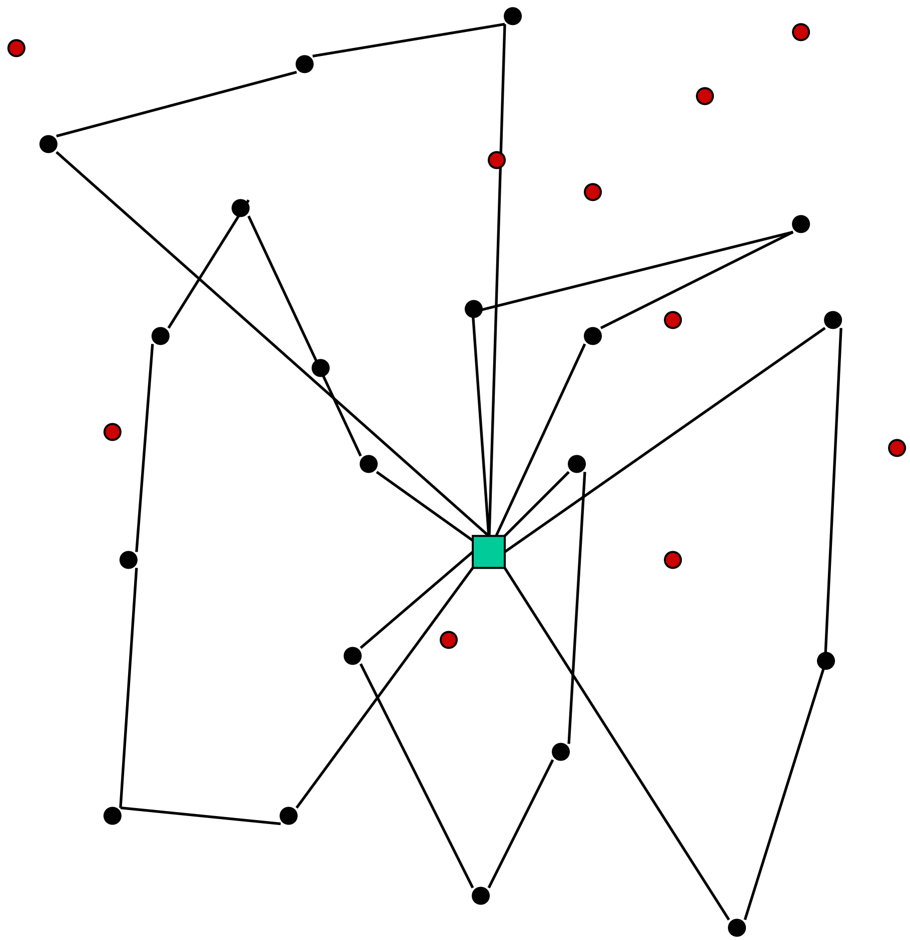
Steps

Initialization	generate, by any heuristic or metaheuristic, an initial solution
	Iteratively:
Selection	select the nodes to be extracted, according to suitable criteria (**schemes**)
Extraction	remove the selected nodes and generate the **restricted solution**
Recombination	starting from extracted nodes, generate a (possibly large) number of **derived sequences**
Re-insertion	re-insert a subset of the derived sequences into the restricted solution, in such a way that all the extracted nodes are covered again
Evaluation	verify a stopping condition and return, if it is the case, to the selection step

An example



An example



SERR Algorithm

Node re-insertion

Node re-insertion is done by solving the following ***set-partitioning*** model:

$$\min \sum_{s \in S} \sum_{i \in I} C_{si} x_{si}$$

$$\sum_{s \ni v} \sum_{i \in I} x_{si} = 1 \quad \forall v \text{ extracted}$$

$$\sum_{s \in S} x_{si} \leq 1 \quad \forall i \in I$$

$$d(r) + \sum_{s \in S} \sum_{i \in r} d(s) x_{si} \leq C \quad \forall r \in R$$

$$0 \leq x_{sj} \leq 1 \text{ integer} \quad \forall s \in S, \forall i \in I$$

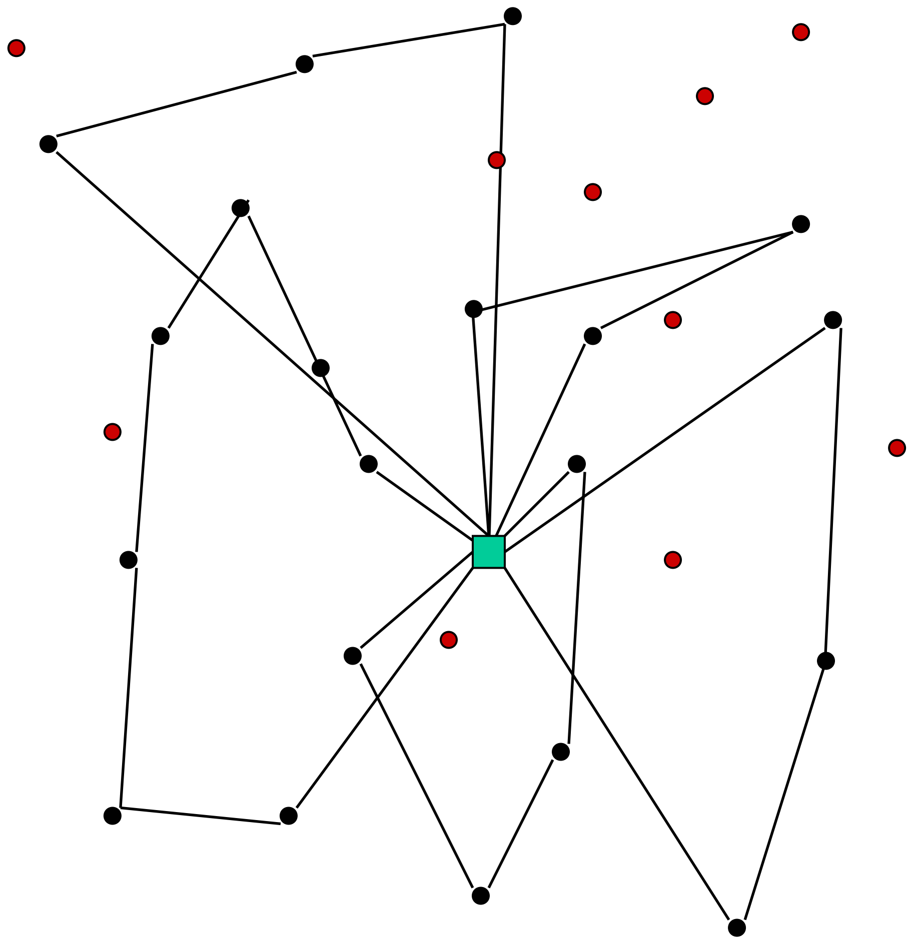
$x_{si} = 1$ if and only if sequence s goes into the insertion point i

C_{si} (best) insertion cost of sequence s into the insertion point i

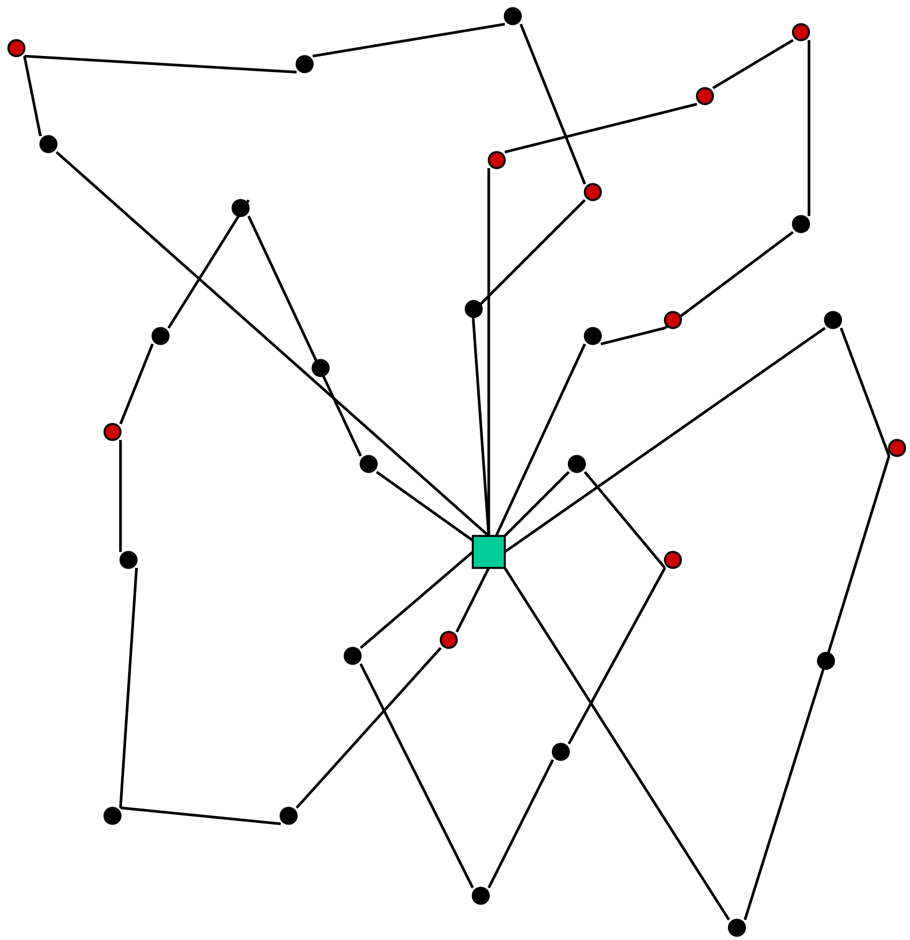
$d(r)$ total demand of the restricted route r

$d(s)$ total demand in the node sequence s

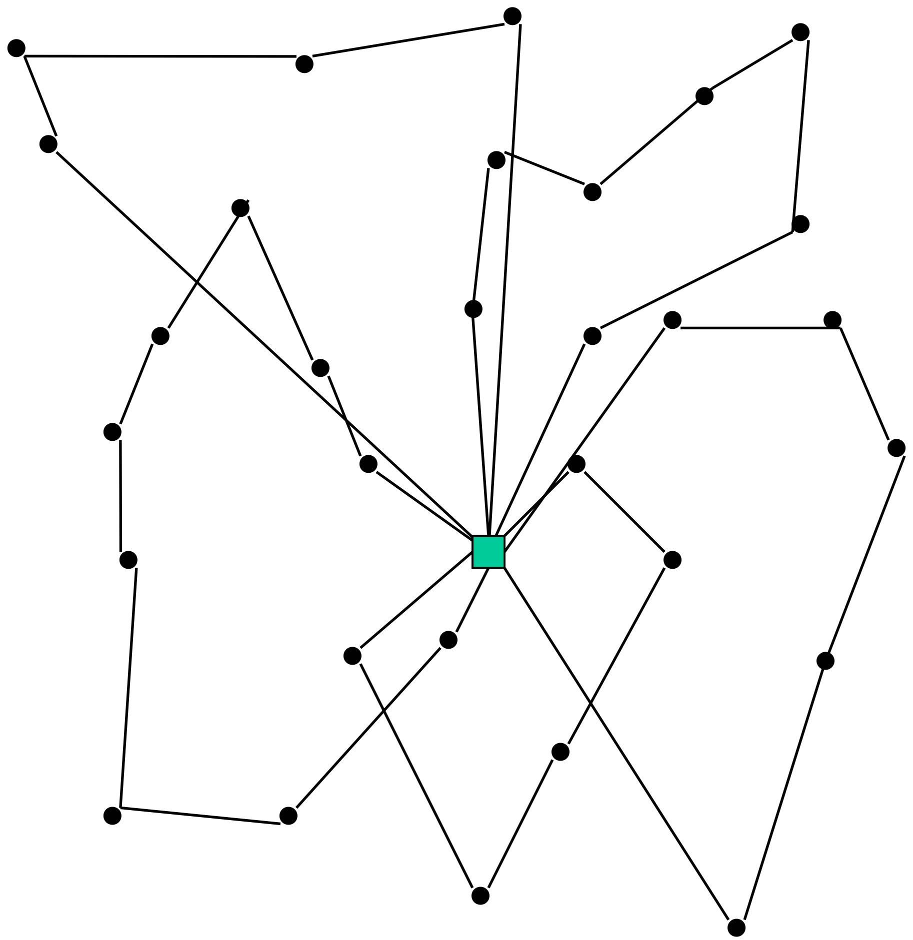
An example (cont.d)



An example (cont.d)

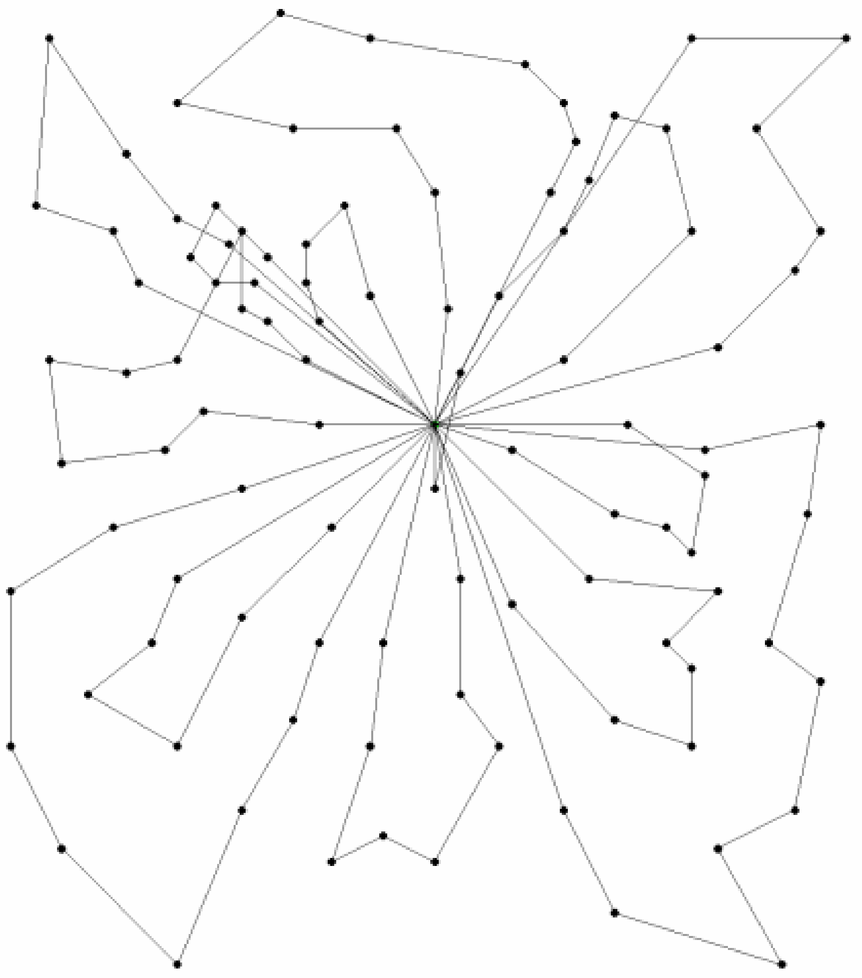


Initial Solution

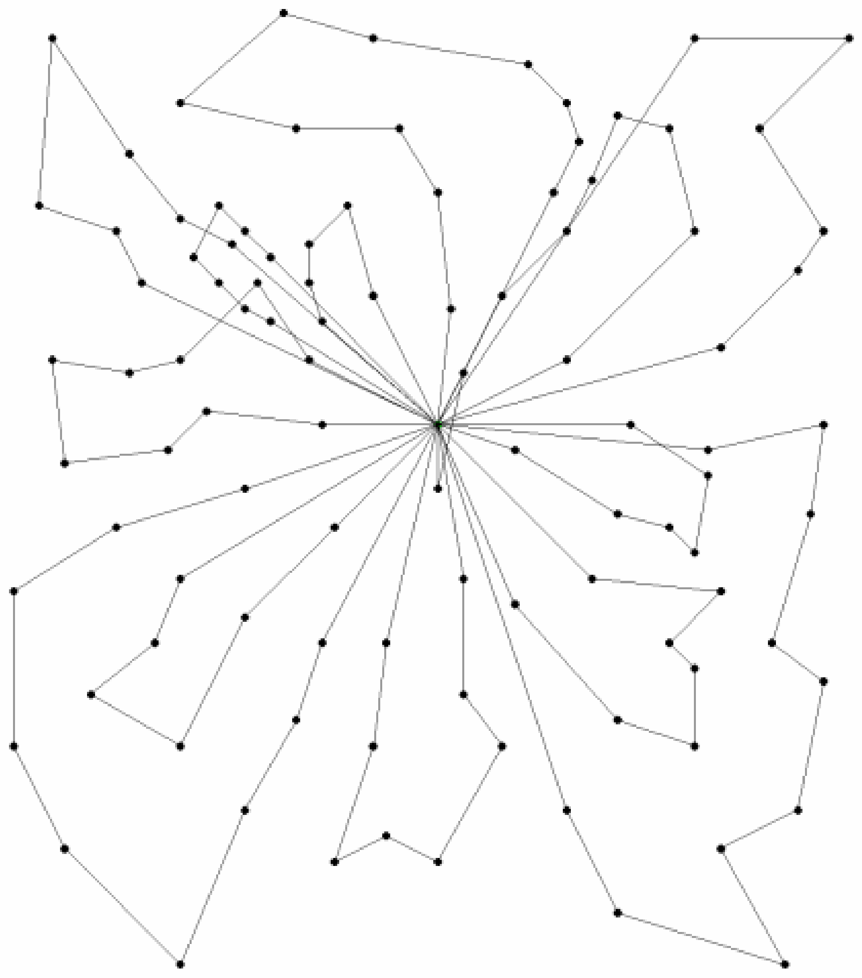


Interesting solutions

Instance E-n101-k14 with rounded costs



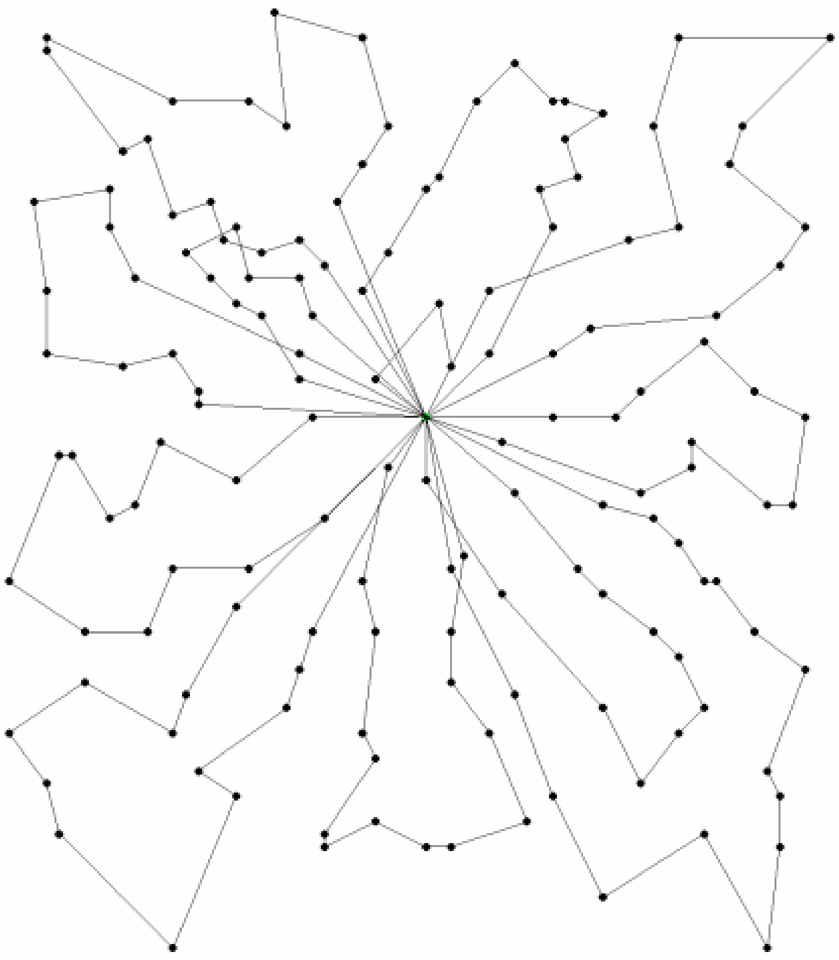
Initial solution: cost 1076
Xu and Kelly, 1996



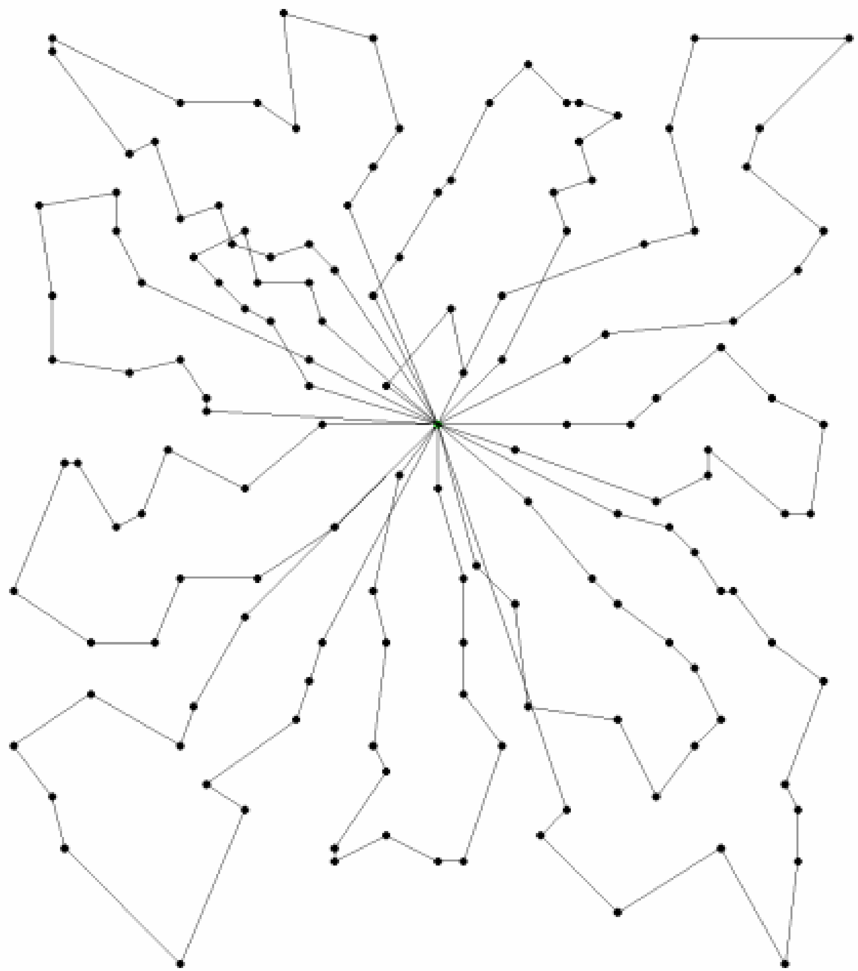
Final solution: cost 1067
New best known solution

Interesting solutions

Instance M-n151-k12 with rounded costs



Initial solution: cost 1023
Gendreau, Hertz and Laporte, 1996



Final solution: cost 1022
New best known solution

Some Computational Results

Instance	Optimal	*SERR* sol.	Gap	Time
P-n50-k8	631	631	0.00%	11:08
P-n55-k10	694	700	0.86%	16:50
P-n60-k10	744	744	0.00%	25:01
P-n60-k15	968	975	0.72%	12:27
P-n65-k10	792	796	0.51%	12:26
P-n70-k10	827	834	0.48%	50:08
B-n68-k9	1272	1275	0.24%	3:02:01
E-n51-k5	521	521	0.00%	4:30
E-n76-k7	682	682	0.00%	27:35
E-n76-k8	735	742	0.95%	30:39
E-n76-k10	830	835	0.60%	1:19:30
E-n76-k14	1021	1032	1.08%	2:45:20
E-n101-k8	815	820	0.61%	2:54:04
E051-05e	524.61	524.61	0.00%	4:51
E076-10e	835.26	835.32	< 0.01%	1:12:05
E101-08e	826.14	831.91	0.70%	2:30:55
E101-10c	819.56	819.56	0.00%	2:35:36
E-n101-k14	-	1076 -> 1067	-	1:36:05
M-n151-k12-a	-	1023 -> 1022	-	7:46:33

New best known solution

Optimal solution(*)

New best heuristic solution known

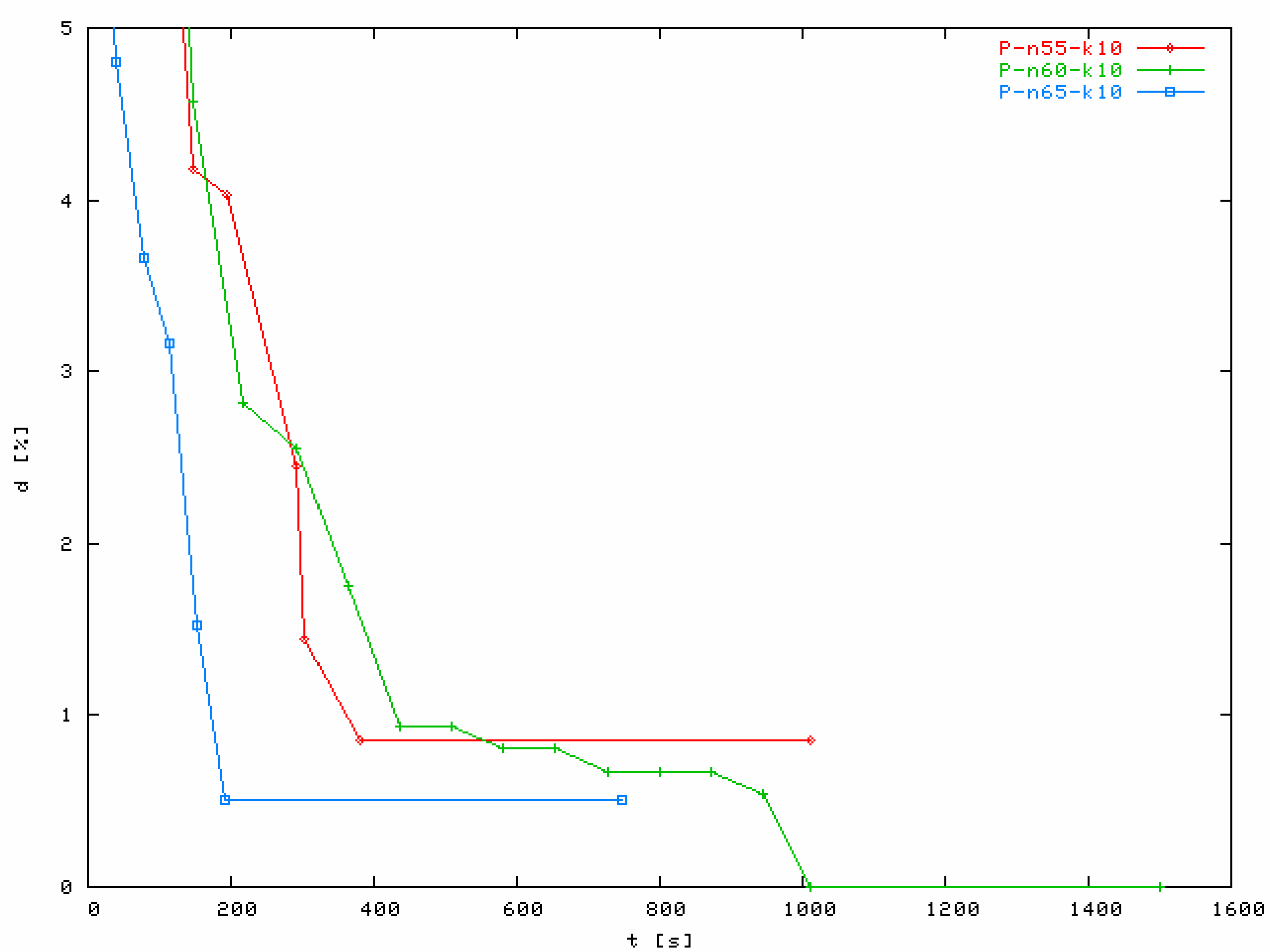
CPU times in the format [hh:]mm:ss

PC: Pentium M 1.6GHz

(*) Most optimal solutions have been found very recently by Fukasawa, Poggi de Aragao, Reis, and Uchoa (September 2003)

Results

Convergence properties of the SERR method



Low-cost solutions available in the first iterations

The best heuristics from the literature are credited for errors of about 2%

Conclusions

Achieved goals

1. **Definition** of a new **neighborhood** with exponential cardinality and of an effective (non-polynomial) **search algorithm**
2. **Simple implementation** based on a general ILP solver
3. **Evaluation** of the algorithm on a widely-used set of instances
4. Determination of the **new best solution** for two of the few instances not yet solved to optimality

Future directions of work

1. **Adaptation** of the method to more constrained versions of VRP, including VRP with **precedence constraints**
2. Use of an external **metaheuristic scheme**

Special contents…