Heuristics for Feasibility and Optimality in Mixed Integer Programming

John Chinneck

Carleton University, Canada
chinneck@sce.carleton.ca

Andrea Lodi

University of Bologna, Italy
andrea.lodi@unibo.it

CIRRELT Spring School on Logistics, Montréal, May 17th, 2010

Outline

1. Introduction and Orientation (Lodi)

Part I: Achieving Integer-Feasibility Quickly

2. Classic Feasibility-Seeking Algorithms (Chinneck)
3. Active Constraint Variable Selection (Chinneck)
4. Branching to Force Change (Chinneck)
5. The Feasibility Pump (Lodi)

Part II: Reaching (quasi-)Optimality Quickly

6. New Node Selection Rules (Chinneck)
7. Local Branching and RINS (Lodi)

Part III: Analyzing Infeasible MIPs

8. Isolating Infeasible Subsystems (Chinneck)
9. Repairing MIP Infeasibility via Local Branching (Lodi)
10. Conclusions (Chinneck)

The role of heuristics in MIP

- We consider a general Mixed Integer Program in the form:

$$\min\{c^T x : Ax \geq b, x \geq 0,\ x_j \text{ integer},\ j \in \mathcal{I}\}$$

  where matrix A does not have a special structure.

The role of heuristics in MIP

- We consider a general Mixed Integer Program in the form:

$$\min\{c^T x : Ax \geq b, x \geq 0,\ x_j \text{ integer},\ j \in \mathcal{I}\}$$

  where matrix A does not have a special structure.

- Thus, the problem is solved through branch-and-bound and the bounds are computed by iteratively solving the LP relaxations through a general-purpose LP solver.

The role of heuristics in MIP

- We consider a general Mixed Integer Program in the form:

$$\min\{c^T x : Ax \geq b, x \geq 0,\ x_j \text{ integer},\ j \in \mathcal{I}\}$$

  where matrix A does not have a special structure.

- Thus, the problem is solved through branch-and-bound and the bounds are computed by iteratively solving the LP relaxations through a general-purpose LP solver.

- MIPs are clearly relevant in logistics because MIP technology is (currently) the most sophisticated and powerful way of modeling complex real-world processes.

The role of heuristics in MIP

- We consider a general Mixed Integer Program in the form:

$$\min\{c^T x : Ax \geq b, x \geq 0,\ x_j \text{ integer},\ j \in \mathcal{I}\}$$

  where matrix A does not have a special structure.

- Thus, the problem is solved through branch-and-bound and the bounds are computed by iteratively solving the LP relaxations through a general-purpose LP solver.

- MIPs are clearly relevant in logistics because MIP technology is (currently) the most sophisticated and powerful way of modeling complex real-world processes.

- Common sense says a MIP solver is a pure and exact algorithm in contrast to (a hybrid and) heuristic one.

The role of heuristics in MIP (cont.d)

- The goal of this introduction is to show that:

The role of heuristics in MIP (cont.d)

- The goal of this introduction is to show that:

  1. MIP solvers are used for a large portion as heuristics
  2. MIP solvers are heuristic in nature
  3. Computation for $\mathcal{NP}$-hard problems is inherently heuristic
  4. Benchmarking is by design heuristic
  5. All kind of heuristic decisions/techniques are hidden everywhere in MIP solvers

The role of heuristics in MIP (cont.d)

- The goal of this introduction is to show that:

  1. MIP solvers are used for a large portion as heuristics
  2. MIP solvers are heuristic in nature
  3. Computation for $\mathcal{NP}$-hard problems is inherently heuristic
  4. Benchmarking is by design heuristic
  5. All kind of heuristic decisions/techniques are hidden everywhere in MIP solvers

  S: MIP solvers are open to different "worlds" and nowadays more and more hybrid algorithms, a perfect framework for the implementation of sophisticated algorithms for finding approximate solutions.

Trivial Facts (1. and 2.)

Some of the points anticipated above are trivial:

1. The classical user of a MIP solver enforces a time/node limit.

Trivial Facts (1. and 2.)

Some of the points anticipated above are trivial:

1. The classical user of a MIP solver enforces a time/node limit.
   The computational resources are limited and sometimes solving MIPs does not require/allow looking for optimality because:

Trivial Facts (1. and 2.)

Some of the points anticipated above are trivial:

1. The classical user of a MIP solver enforces a time/node limit.

   The computational resources are limited and sometimes solving MIPs does not require/allow looking for optimality because:
   - it is just a portion of the iterative decision process
   - the overall problem is highly complex and it is split *a priori* into pieces
   - it must be done online

Trivial Facts (1. and 2.)

Some of the points anticipated above are trivial:

1. The classical user of a MIP solver enforces a time/node limit.
   The computational resources are limited and sometimes solving MIPs does not require/allow looking for optimality because:
   - it is just a portion of the iterative decision process
   - the overall problem is highly complex and it is split *a priori* into pieces
   - it must be done online

2. Any MIP solver works with tolerances.

Trivial Facts (1. and 2.)

Some of the points anticipated above are trivial:

1. The classical user of a MIP solver enforces a time/node limit.
   The computational resources are limited and sometimes solving MIPs does not require/allow looking for optimality because:
   - it is just a portion of the iterative decision process
   - the overall problem is highly complex and it is split *a priori* into pieces
   - it must be done online

2. Any MIP solver works with tolerances.
   The tolerance is both in the solution of the LPs and on the branching side.

Trivial Facts (1. and 2.)

Some of the points anticipated above are trivial:

1. The classical user of a MIP solver enforces a time/node limit.
   The computational resources are limited and sometimes solving MIPs does not require/allow looking for optimality because:
   - it is just a portion of the iterative decision process
   - the overall problem is highly complex and it is split *a priori* into pieces
   - it must be done online

2. Any MIP solver works with tolerances.
   The tolerance is both in the solution of the LPs and on the branching side.
   It is pretty tight for feasibility, thus good solvers certify their solutions as "really feasible".

Trivial Facts (1. and 2.)

Some of the points anticipated above are trivial:

1. The classical user of a MIP solver enforces a time/node limit.
   The computational resources are limited and sometimes solving MIPs does not require/allow looking for optimality because:
   - it is just a portion of the iterative decision process
   - the overall problem is highly complex and it is split *a priori* into pieces
   - it must be done online

2. Any MIP solver works with tolerances.
   The tolerance is both in the solution of the LPs and on the branching side.
   It is pretty tight for feasibility, thus good solvers certify their solutions as "really feasible".
   It is less strict for optimality. A popular default value for that is 0.01% which for special applications is far from acceptable.
   (Ever noticed the number of nodes left to explore at the end of a run?)

Slightly-less Trivial Facts: 3.

3. MIP $\mathcal{NP}$-hardness.

Slightly-less Trivial Facts: 3.

3. MIP $\mathcal{N}P$-hardness.

   By definition there is always in the tree a polynomial path to the optimal solution. However, unless $\mathcal{P} = \mathcal{NP}$, in the worst-case, the path followed will be exponentially long.

Slightly-less Trivial Facts: 3.

3. MIP $\mathcal{N}P$-hardness.

   By definition there is always in the tree a polynomial path to the optimal solution. However, unless $\mathcal{P} = \mathcal{NP}$, in the worst-case, the path followed will be exponentially long.

   In other words, branch-and-bound algorithms HEURISTICALLY explore the tree and such an exploration can be unlucky because of:

Slightly-less Trivial Facts: 3.

3. MIP $\mathcal{NP}$-hardness.

   By definition there is always in the tree a polynomial path to the optimal solution. However, unless $\mathcal{P} = \mathcal{NP}$, in the worst-case, the path followed will be exponentially long.

   In other words, branch-and-bound algorithms HEURISTICALLY explore the tree and such an exploration can be unlucky because of:

   – bad algorithmic decisions (discussed later)

Slightly-less Trivial Facts: 3.

3. MIP $\mathcal{NP}$-hardness.

   By definition there is always in the tree a polynomial path to the optimal solution. However, unless $\mathcal{P} = \mathcal{NP}$, in the worst-case, the path followed will be exponentially long.

   In other words, branch-and-bound algorithms HEURISTICALLY explore the tree and such an exploration can be unlucky because of:

   - bad algorithmic decisions (discussed later)

   - *performance variability*:

     * a change in performance
     * for the same problem (or problems in the same family)
     * created by a change in the solver or in the environment
     * that seems "performance neutral".

Slightly-less Trivial Facts: 3.

3. MIP $\mathcal{NP}$-hardness.

   By definition there is always in the tree a polynomial path to the optimal solution. However, unless $\mathcal{P} = \mathcal{NP}$, in the worst-case, the path followed will be exponentially long.

   In other words, branch-and-bound algorithms HEURISTICALLY explore the tree and such an exploration can be unlucky because of:

   - bad algorithmic decisions (discussed later)

   - *performance variability*:

     * a change in performance
     * for the same problem (or problems in the same family)
     * created by a change in the solver or in the environment
     * that seems "performance neutral".

   - ...

Performance Variability, Emilie Danna #1

Example: 10 teams, CPLEX 11, Linux


```
Tried aggregator 1 time.
MIP Presolve eliminated 20 rows and 425 columns.
Reduced MIP has 210 rows, 1600 columns, and 9600 nonzeros.
Presolve time =    0.01 sec.
Clique table members: 170.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: none, using 1 thread.
Root relaxation solution time =    0.05 sec.

      Nodes                                         Cuts/
 Node  Left     Objective  IInf  Best Integer    Best Node    ItCnt     Gap

    0     0      917.0000   140                  917.0000     1100
    0     0      924.0000   165                  Cuts: 50     1969
    0     0      924.0000   167                  Cuts: 17     2348
    0     0      924.0000   175                 Cliques: 14     2731
*   0+    0                        924.0000     924.0000     2731    0.00%

Clique cuts applied:  16
Zero-half cuts applied:  3
Gomory fractional cuts applied:  1

Solution pool: 1 solution saved.

MIP - Integer optimal solution:  Objective =  9.2400000000e+02
Solution time =    0.41 sec.  Iterations = 2731  Nodes = 0
```

Performance Variability, Emilie Danna #2

Example: 10 teams, CPLEX 11, AIX


```
Tried aggregator 1 time.
MIP Presolve eliminated 20 rows and 425 columns.
Reduced MIP has 210 rows, 1600 columns, and 9600 nonzeros.
Presolve time =    0.00 sec.
Clique table members: 170.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: none, using 1 thread.
Root relaxation solution time =    0.18 sec.

      Nodes                                         Cuts/
 Node  Left     Objective  IInf  Best Integer    Best Node    ItCnt     Gap

    0     0      917.0000   151                  917.0000     1053
    0     0      924.0000   152                  Cuts: 53     1801
    0     0      924.0000   161                 Cliques: 14     2336
    0     0      924.0000   163                 Cliques: 12     2609
    0     2      924.0000   163                  924.0000     2609
*  100+   96                          952.0000     924.0000    12316    2.94%
   1000   520     926.7273    85      952.0000     924.0000    97832    2.94%
*  1425     0     integral     0      924.0000     924.0000   122948    0.00%

Clique cuts applied:  12
Zero-half cuts applied:  4
Gomory fractional cuts applied:  2

Solution pool: 2 solutions saved.

MIP - Integer optimal solution:  Objective =  9.2400000000e+02
Solution time =   41.39 sec.  Iterations = 122948  Nodes = 1426
```

Slightly-less Trivial Facts: 4.

4. Benchmarking of (commercial) MIP solvers.

Slightly-less Trivial Facts: 4.

4. Benchmarking of (commercial) MIP solvers.

   The testsets of MIP solvers, in particular the commercial ones, are made of thousands of instances. Instances are classified into categories from very easy to very difficult.

Slightly-less Trivial Facts: 4.

4. Benchmarking of (commercial) MIP solvers.

   The testsets of MIP solvers, in particular the commercial ones, are made of thousands of instances. Instances are classified into categories from very easy to very difficult.

   Any new idea is tested on the entire testset and in order to "make it into the solver" it must:

Slightly-less Trivial Facts: 4.

4. Benchmarking of (commercial) MIP solvers.

   The testsets of MIP solvers, in particular the commercial ones, are made of thousands of instances. Instances are classified into categories from very easy to very difficult.

   Any new idea is tested on the entire testset and in order to "make it into the solver" it must:

   - improve on the subset of problems to which the idea applies
   - not deteriorate (too much) on the rest of the problems, generally the easy ones

Slightly-less Trivial Facts: 4.

4. Benchmarking of (commercial) MIP solvers.

   The testsets of MIP solvers, in particular the commercial ones, are made of thousands of instances. Instances are classified into categories from very easy to very difficult.

   Any new idea is tested on the entire testset and in order to "make it into the solver" it must:

   - improve on the subset of problems to which the idea applies
   - not deteriorate (too much) on the rest of the problems, generally the easy ones

   Because of the MIP $\mathcal{NP}$-hardness, it is hard to recognize problems as good or bad for an idea, then such an idea must be HEURISTICALLY "weakened" to accomplish simultaneously the two given goals.

5. MIP Solvers, nowadays key features

- The current generation of MIP solvers incorporates key ideas developed continuously during the first 50 years of Integer Programming:

5. MIP Solvers, nowadays key features

- The current generation of MIP solvers incorporates key ideas developed continuously during the first 50 years of Integer Programming:

  A Cutting plane generation:
  Gomory Mixed Integer cuts, Mixed Integer Rounding, cover cuts, flow covers, 0/1 cuts, . . .

5. MIP Solvers, nowadays key features

- The current generation of MIP solvers incorporates key ideas developed continuously during the first 50 years of Integer Programming:

  A Cutting plane generation:
  Gomory Mixed Integer cuts, Mixed Integer Rounding, cover cuts, flow covers, 0/1 cuts, . . .

  B Sophisticated branching strategies:
  strong branching, pseudo-cost branching, diving and hybrids

5. MIP Solvers, nowadays key features

- The current generation of MIP solvers incorporates key ideas developed continuously during the first 50 years of Integer Programming:

  A Cutting plane generation:
  Gomory Mixed Integer cuts, Mixed Integer Rounding, cover cuts, flow covers, 0/1 cuts, . . .

  B Sophisticated branching strategies:
  strong branching, pseudo-cost branching, diving and hybrids

  C Primal heuristics:
  rounding heuristics (from easy to complex), local search, metaheuristics, . . .

5. MIP Solvers, nowadays key features

- The current generation of MIP solvers incorporates key ideas developed continuously during the first 50 years of Integer Programming:

  A Cutting plane generation:
  Gomory Mixed Integer cuts, Mixed Integer Rounding, cover cuts, flow covers, 0/1 cuts, . . .

  B Sophisticated branching strategies:
  strong branching, pseudo-cost branching, diving and hybrids

  C Primal heuristics:
  rounding heuristics (from easy to complex), local search, metaheuristics, . . .

  D Preprocessing:
  probing, bound strengthening, propagation

5. MIP Solvers, nowadays key features

- The current generation of MIP solvers incorporates key ideas developed continuously during the first 50 years of Integer Programming:

  A Cutting plane generation:
  Gomory Mixed Integer cuts, Mixed Integer Rounding, cover cuts, flow covers, 0/1 cuts, . . .

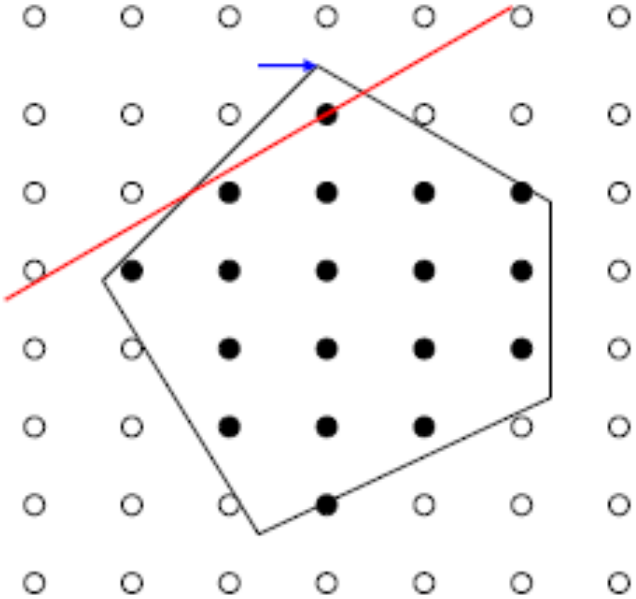
  B Sophisticated branching strategies:
  strong branching, pseudo-cost branching, diving and hybrids

  C Primal heuristics:
  rounding heuristics (from easy to complex), local search, metaheuristics, . . .

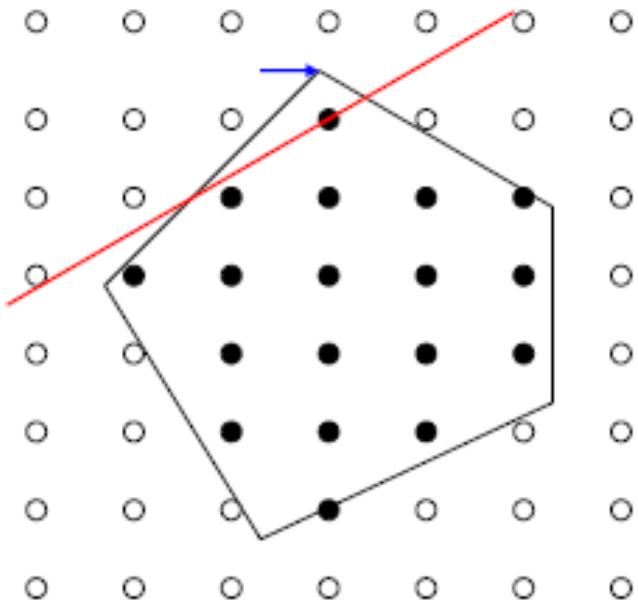
  D Preprocessing:
  probing, bound strengthening, propagation

- Moreover, the MIP computation has reached such an effective and stable quality to allow the solution of sub-MIPs in the algorithmic process, the MIPping approach. [Fischetti & L., *MPB* 2002]
  These sub-MIPs are solved both for cutting plane generation and in the primal heuristic context.

5.A: Cutting Planes

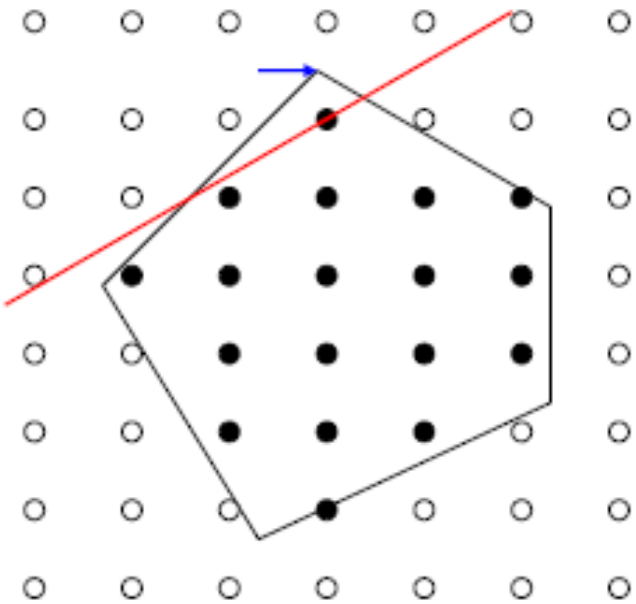


5.A: Cutting Planes



- Current MIP cutting plane technology is based on the following almost trivial two-step heuristic:

5.A: Cutting Planes



- Current MIP cutting plane technology is based on the following almost trivial two-step heuristic:
  - HEURISTICALLY aggregate the entire MIP into a mixed-integer set of one single (!) row
  - apply the cut separation procedure to such a mixed-integer set
    (Gomory fractional, Gomory Mixed-Integer, MIR, $\{0, \frac{1}{2}\}$, . . .)

5.B: Branching

- The branching phase is by construction (probably) the most delicate part of the branch-and-cut framework currently implemented with commercial and non-commercial MIP solvers.

5.B: Branching

- The branching phase is by construction (probably) the most delicate part of the branch-and-cut framework currently implemented with commercial and non-commercial MIP solvers.

- "Bad" decisions at early stages of the search, i.e., high levels in the tree, result in the exponential increase of the tree itself.

5.B: Branching

- The branching phase is by construction (probably) the most delicate part of the branch-and-cut framework currently implemented with commercial and non-commercial MIP solvers.

- "Bad" decisions at early stages of the search, i.e., high levels in the tree, result in the exponential increase of the tree itself.

- Almost all sophisticated techniques used to avoid as much as possible these "bad" decisions belong to the family called *strong branching* in which several LPs are solved to measure the impact on the bound of a variable-value assignment.

5.B: Branching

- The branching phase is by construction (probably) the most delicate part of the branch-and-cut framework currently implemented with commercial and non-commercial MIP solvers.

- "Bad" decisions at early stages of the search, i.e., high levels in the tree, result in the exponential increase of the tree itself.

- Almost all sophisticated techniques used to avoid as much as possible these "bad" decisions belong to the family called *strong branching* in which several LPs are solved to measure the impact on the bound of a variable-value assignment.

- However, the computational effort required by using "fully" such a technique would be too high (at every decision point, all variable-value assignments would need to be evaluated).

5.B: Branching

- The branching phase is by construction (probably) the most delicate part of the branch-and-cut framework currently implemented with commercial and non-commercial MIP solvers.

- "Bad" decisions at early stages of the search, i.e., high levels in the tree, result in the exponential increase of the tree itself.

- Almost all sophisticated techniques used to avoid as much as possible these "bad" decisions belong to the family called *strong branching* in which several LPs are solved to measure the impact on the bound of a variable-value assignment.

- However, the computational effort required by using "fully" such a technique would be too high (at every decision point, all variable-value assignments would need to be evaluated). Thus, two heuristic criteria are applied:
  - only a subset of these assignments are evaluated, and
  - each LP is not solved to optimality but within a given iteration limit.

5.B: Branching

- The branching phase is by construction (probably) the most delicate part of the branch-and-cut framework currently implemented with commercial and non-commercial MIP solvers.

- "Bad" decisions at early stages of the search, i.e., high levels in the tree, result in the exponential increase of the tree itself.

- Almost all sophisticated techniques used to avoid as much as possible these "bad" decisions belong to the family called *strong branching* in which several LPs are solved to measure the impact on the bound of a variable-value assignment.

- However, the computational effort required by using "fully" such a technique would be too high (at every decision point, all variable-value assignments would need to be evaluated). Thus, two heuristic criteria are applied:
  - only a subset of these assignments are evaluated, and
  - each LP is not solved to optimality but within a given iteration limit.

- The heuristic side of branching is not limited to the above criteria and has an impact in almost all branching components, as for example in the decision on how to break ties.

5.C: Primal Heuristics

- This is of course the most trivial context in which "real" heuristics play a central role in the MIP solvers.

5.C: Primal Heuristics

- This is of course the most trivial context in which "real" heuristics play a central role in the MIP solvers.

- From the one side, the last 5 to 10 years have seen a tremendous improvement in the capability of primal heuristics to find very good (almost optimal) solution early in the tree.

5.C: Primal Heuristics

- This is of course the most trivial context in which "real" heuristics play a central role in the MIP solvers.

- From the one side, the last 5 to 10 years have seen a tremendous improvement in the capability of primal heuristics to find very good (almost optimal) solution early in the tree.

- On the other hand, a very meaningful experiment by Tobias Achterberg has shown that even the knowledge of the optimal solution from the beginning of the search only improves the running time of a MIP solver on average by a factor of 2.

5.C: Primal Heuristics

- This is of course the most trivial context in which "real" heuristics play a central role in the MIP solvers.

- From the one side, the last 5 to 10 years have seen a tremendous improvement in the capability of primal heuristics to find very good (almost optimal) solution early in the tree.

- On the other hand, a very meaningful experiment by Tobias Achterberg has shown that even the knowledge of the optimal solution from the beginning of the search only improves the running time of a MIP solver on average by a factor of 2.
  In other words, heuristics largely impact on the user perception of the quality of a solver.

5.C: Primal Heuristics

- This is of course the most trivial context in which "real" heuristics play a central role in the MIP solvers.

- From the one side, the last 5 to 10 years have seen a tremendous improvement in the capability of primal heuristics to find very good (almost optimal) solution early in the tree.

- On the other hand, a very meaningful experiment by Tobias Achterberg has shown that even the knowledge of the optimal solution from the beginning of the search only improves the running time of a MIP solver on average by a factor of 2.
  In other words, heuristics largely impact on the user perception of the quality of a solver.

- However, the most surprising impact of primal heuristics in the solver is on MIPping.

5.C: Primal Heuristics

- This is of course the most trivial context in which "real" heuristics play a central role in the MIP solvers.

- From the one side, the last 5 to 10 years have seen a tremendous improvement in the capability of primal heuristics to find very good (almost optimal) solution early in the tree.

- On the other hand, a very meaningful experiment by Tobias Achterberg has shown that even the knowledge of the optimal solution from the beginning of the search only improves the running time of a MIP solver on average by a factor of 2.
  In other words, heuristics largely impact on the user perception of the quality of a solver.

- However, the most surprising impact of primal heuristics in the solver is on MIPping.
  - Within a MIP one can easily define all components of local search: (i) the neighborhood, (ii) the search within the neighborhood and (iii) the escape from local optima, i.e., the diversification.

5.C: Primal Heuristics

- This is of course the most trivial context in which "real" heuristics play a central role in the MIP solvers.

- From the one side, the last 5 to 10 years have seen a tremendous improvement in the capability of primal heuristics to find very good (almost optimal) solution early in the tree.

- On the other hand, a very meaningful experiment by Tobias Achterberg has shown that even the knowledge of the optimal solution from the beginning of the search only improves the running time of a MIP solver on average by a factor of 2.
  In other words, heuristics largely impact on the user perception of the quality of a solver.

- However, the most surprising impact of primal heuristics in the solver is on MIPping.
  - Within a MIP one can easily define all components of local search: (i) the neighborhood, (ii) the search within the neighborhood and (iii) the escape from local optima, i.e., the diversification.
  - The sub-MIPs used to find better solutions and/or to generate cuts are NEVER solved to optimality: the full integration of sophisticated heuristics in the solvers allows to count on the fact that nested calls of the same solvers could produce heuristic solutions fast!

5.D: Preprocessing

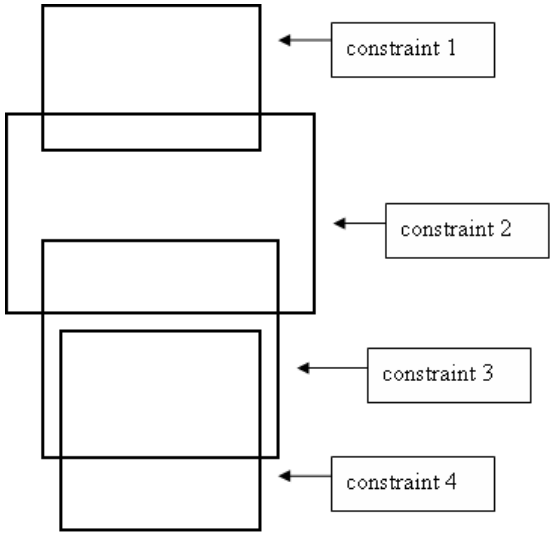
- In the preprocessing phase a MIP solver tries to detect certain changes in the input that will probably lead to a better performance of the solution process. Two examples are canceling redundant constraints and strengthening variable bounds.

5.D: Preprocessing

- In the preprocessing phase a MIP solver tries to detect certain changes in the input that will probably lead to a better performance of the solution process. Two examples are canceling redundant constraints and strengthening variable bounds.
- Many of the used techniques are called in the *Constraint Programming* context propagation algorithms:

5.D: Preprocessing

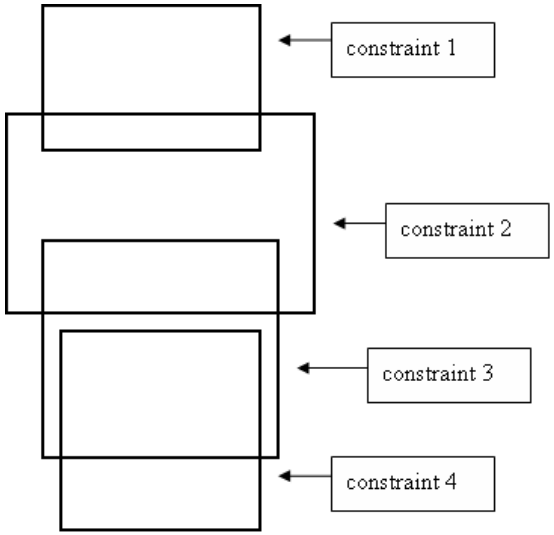
- In the preprocessing phase a MIP solver tries to detect certain changes in the input that will probably lead to a better performance of the solution process. Two examples are canceling redundant constraints and strengthening variable bounds.
- Many of the used techniques are called in the *Constraint Programming* context propagation algorithms:



  - A global constraint defines combinatorially a portion of the feasible region, i.e., it is able to check feasibility of an assignment of values to variables.

5.D: Preprocessing

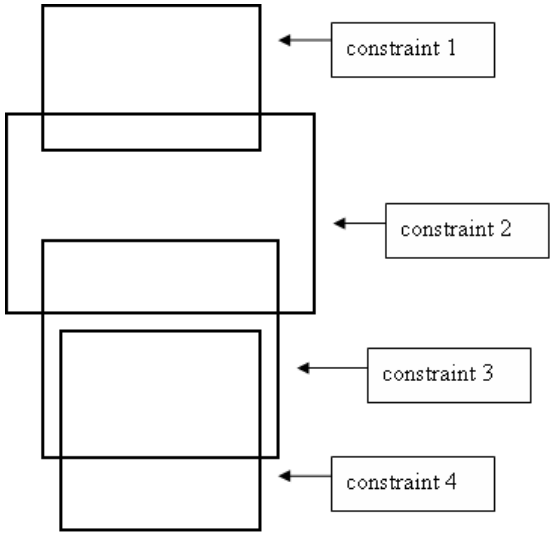
- In the preprocessing phase a MIP solver tries to detect certain changes in the input that will probably lead to a better performance of the solution process. Two examples are canceling redundant constraints and strengthening variable bounds.
- Many of the used techniques are called in the *Constraint Programming* context propagation algorithms:



  - A global constraint defines combinatorially a portion of the feasible region, i.e., it is able to check feasibility of an assignment of values to variables.
  - Moreover, a global constraint contains an algorithm which prunes (filters) values from the variable domains so as reduce as much as possible the search space.

5.D: Preprocessing

- In the preprocessing phase a MIP solver tries to detect certain changes in the input that will probably lead to a better performance of the solution process. Two examples are canceling redundant constraints and strengthening variable bounds.
- Many of the used techniques are called in the *Constraint Programming* context propagation algorithms:

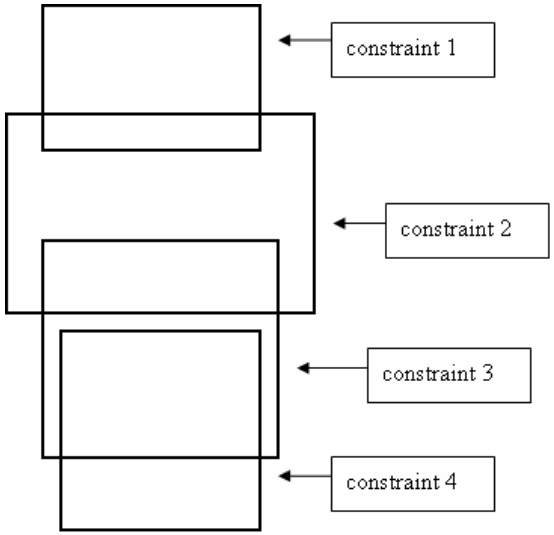


  - A global constraint defines combinatorially a portion of the feasible region, i.e., it is able to check feasibility of an assignment of values to variables.
  - Moreover, a global constraint contains an algorithm which prunes (filters) values from the variable domains so as reduce as much as possible the search space.

- MIPs do not explicitly contain global constraints, thus, the propagation is applied by LOCALLY comparing constraints/variables, a structurally heuristic process.

5.D: Preprocessing

- In the preprocessing phase a MIP solver tries to detect certain changes in the input that will probably lead to a better performance of the solution process. Two examples are canceling redundant constraints and strengthening variable bounds.
- Many of the used techniques are called in the *Constraint Programming* context propagation algorithms:



  - A global constraint defines combinatorially a portion of the feasible region, i.e., it is able to check feasibility of an assignment of values to variables.
  - Moreover, a global constraint contains an algorithm which prunes (filters) values from the variable domains so as reduce as much as possible the search space.

- MIPs do not explicitly contain global constraints, thus, the propagation is applied by LOCALLY comparing constraints/variables, a structurally heuristic process.
- Indeed, random permutations of rows/columns of the MIP generally lead to worse performance of the solvers because of reduced preprocessing effectiveness.

Summary: MIP solvers as (Hybrid) Heuristic Algorithms

- Not only MIP solvers:

Summary: MIP solvers as (Hybrid) Heuristic Algorithms

- Not only MIP solvers:
  - are used as heuristics (they are in several important cases the best heuristic approaches even in the case of structured problems, e.g., set covering type problems)

Summary: MIP solvers as (Hybrid) Heuristic Algorithms

- Not only MIP solvers:
  - are used as heuristics (they are in several important cases the best heuristic approaches even in the case of structured problems, e.g., set covering type problems)
  - are developed and tested using heuristic criteria

Summary: MIP solvers as (Hybrid) Heuristic Algorithms

- Not only MIP solvers:
  - are used as heuristics (they are in several important cases the best heuristic approaches even in the case of structured problems, e.g., set covering type problems)
  - are developed and tested using heuristic criteria
  - use heuristic decisions in each of their basic components

Summary: MIP solvers as (Hybrid) Heuristic Algorithms

- Not only MIP solvers:
  - are used as heuristics (they are in several important cases the best heuristic approaches even in the case of structured problems, e.g., set covering type problems)
  - are developed and tested using heuristic criteria
  - use heuristic decisions in each of their basic components
- but also incorporate:

Summary: MIP solvers as (Hybrid) Heuristic Algorithms

- Not only MIP solvers:
  - are used as heuristics (they are in several important cases the best heuristic approaches even in the case of structured problems, e.g., set covering type problems)
  - are developed and tested using heuristic criteria
  - use heuristic decisions in each of their basic components
- but also incorporate:
  - heuristic algorithms which are not only used to find good solutions but also as building blocks of sophisticated algorithmic strategies like MIPping

Summary: MIP solvers as (Hybrid) Heuristic Algorithms

- Not only MIP solvers:
  - are used as heuristics (they are in several important cases the best heuristic approaches even in the case of structured problems, e.g., set covering type problems)
  - are developed and tested using heuristic criteria
  - use heuristic decisions in each of their basic components
- but also incorporate:
  - heuristic algorithms which are not only used to find good solutions but also as building blocks of sophisticated algorithmic strategies like MIPping
  - propagation algorithms from Constraint Programming

Summary: MIP solvers as (Hybrid) Heuristic Algorithms

- Not only MIP solvers:
  - are used as heuristics (they are in several important cases the best heuristic approaches even in the case of structured problems, e.g., set covering type problems)
  - are developed and tested using heuristic criteria
  - use heuristic decisions in each of their basic components
- but also incorporate:
  - heuristic algorithms which are not only used to find good solutions but also as building blocks of sophisticated algorithmic strategies like MIPping
  - propagation algorithms from Constraint Programming
- In other words, MIP solvers are open frameworks which already are, and will more and more be, sophisticated hybrids.

Heuristics for Feasibility and Optimality in Mixed Integer Programming

John Chinneck

Carleton University, Canada
chinneck@sce.carleton.ca

Andrea Lodi

University of Bologna, Italy
andrea.lodi@unibo.it

CIRRELT Spring School on Logistics, Montréal, May 17th, 2010

Outline

1. Introduction and Orientation (Lodi)

Part I: Achieving Integer-Feasibility Quickly

2. Classic Feasibility-Seeking Algorithms (Chinneck)
3. Active Constraint Variable Selection (Chinneck)
4. Branching to Force Change (Chinneck)
5. The Feasibility Pump (Lodi)

Part II: Reaching (quasi-)Optimality Quickly

6. New Node Selection Rules (Chinneck)
7. Local Branching and RINS (Lodi)

Part III: Analyzing Infeasible MIPs

8. Isolating Infeasible Subsystems (Chinneck)
9. Repairing MIP Infeasibility via Local Branching (Lodi)
10. Conclusions (Chinneck)

Finding an initial feasible solution by Feasibility Pump

- We consider a generic 0-1 MIP of the form:

$$
\begin{aligned}
(P) \qquad & \min c^T x && (1)\\
& Ax \geq b && (2)\\
& x_j \in \{0,1\} \quad \forall j \in \mathcal{B} && (3)\\
& x_j \geq 0 \qquad \forall j \in \mathcal{C} && (4)
\end{aligned}
$$

Finding an initial feasible solution by Feasibility Pump

- We consider a generic 0-1 MIP of the form:

$$(P) \qquad \min c^T x \tag{1}$$

$$Ax \geq b \tag{2}$$

$$x_j \in \{0,1\} \quad \forall j \in \mathcal{B} \tag{3}$$

$$x_j \geq 0 \qquad \forall j \in \mathcal{C} \tag{4}$$

and we look for an initial feasible solution through an algorithm called `Feasibility Pump` (`FP`)

[Fischetti, Glover & L., *MPA* 2005]

`Feasibility Pump`: the basic scheme

- We start from any $x^* \in P$, and define its rounding $\widetilde{x}$.

`Feasibility Pump`: the basic scheme

- 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, it is an easily solvable LP problem.

`Feasibility Pump`: the basic scheme

- 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, it is an easily solvable LP problem.
- If x_j^* is integral $\forall j \in \mathcal{B}$, then x^* is a feasible MIP solution and we are done.

Feasibility Pump: the basic scheme

- 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, it is an easily solvable LP problem.
- If x_j^* is integral $\forall j \in \mathcal{B}$, then x^* is a feasible MIP solution and we are done.
- Otherwise, we replace $\widetilde{x}$ by the rounding of x^*, and repeat.

Feasibility Pump: the basic scheme

- 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, it is an easily solvable LP problem.
- If x_j^* is integral $\forall j \in \mathcal{B}$, 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:

Feasibility Pump: the basic scheme

- 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, it is an easily solvable LP problem.
- If x_j^* is integral $\forall j \in \mathcal{B}$, 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,

Feasibility Pump: the basic scheme

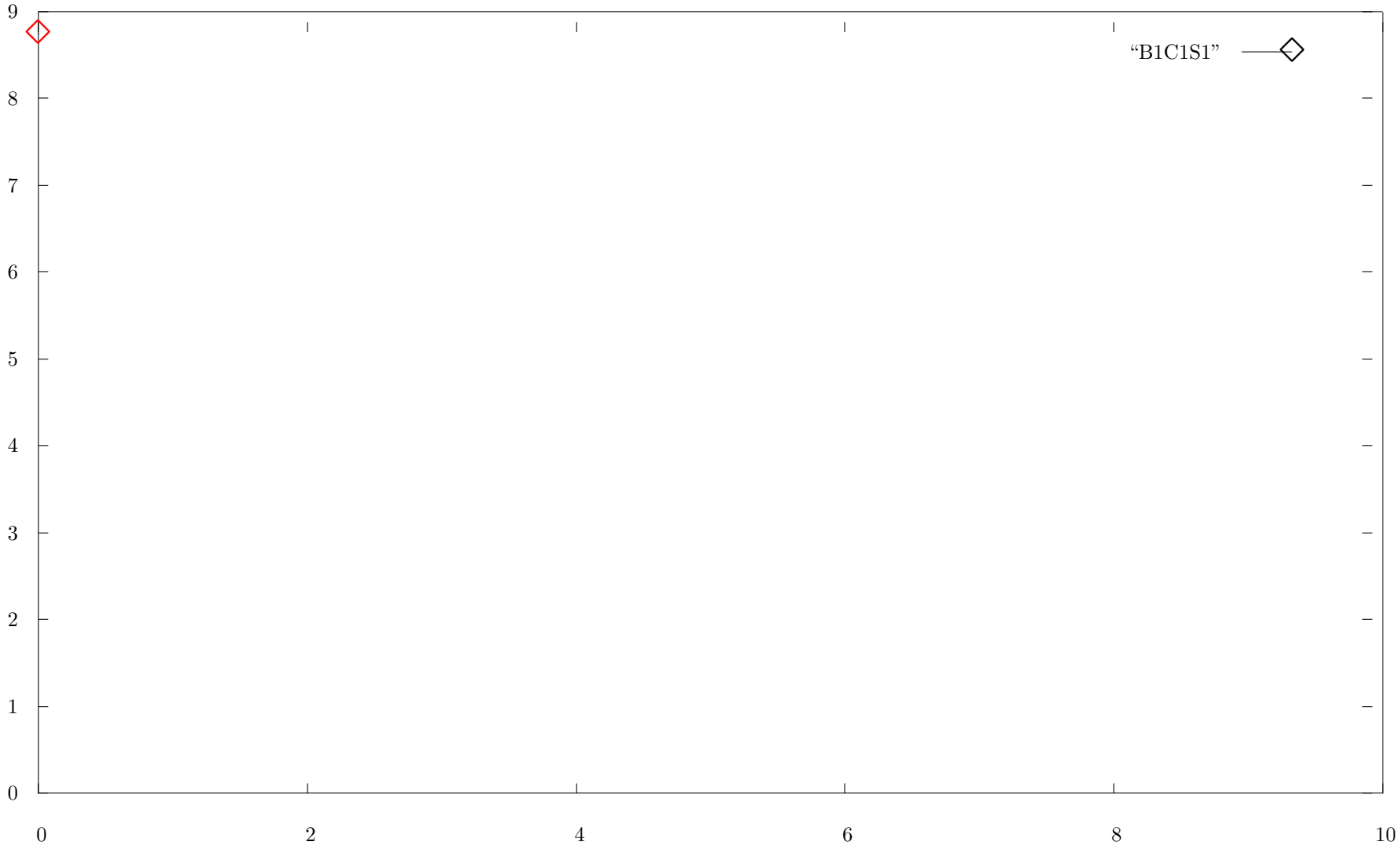
- We start from any $x^* \in P$, and define its rounding $\widetilde{x}$.
- At each iteration we look for a point $x^* \in P$ which is as close as possible to the current $\widetilde{x}$ by solving the problem:

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

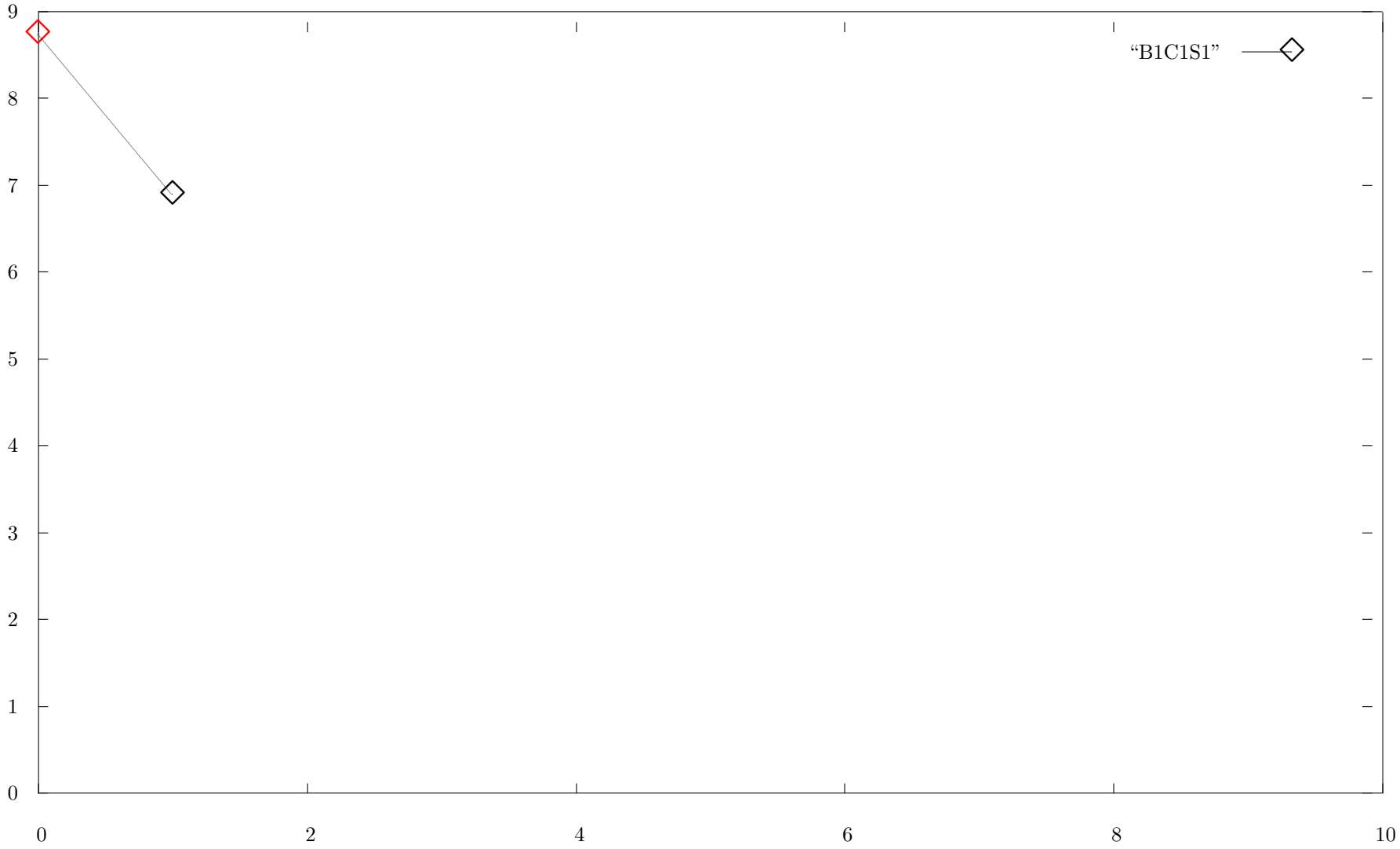
  Assuming $\Delta(x, \widetilde{x})$ is chosen appropriately, it is an easily solvable LP problem.
- If x_j^* is integral $\forall j \in \mathcal{B}$, then x^* is a feasible MIP solution and we are done.
- Otherwise, we replace $\widetilde{x}$ by the rounding of x^*, and repeat.

- 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.

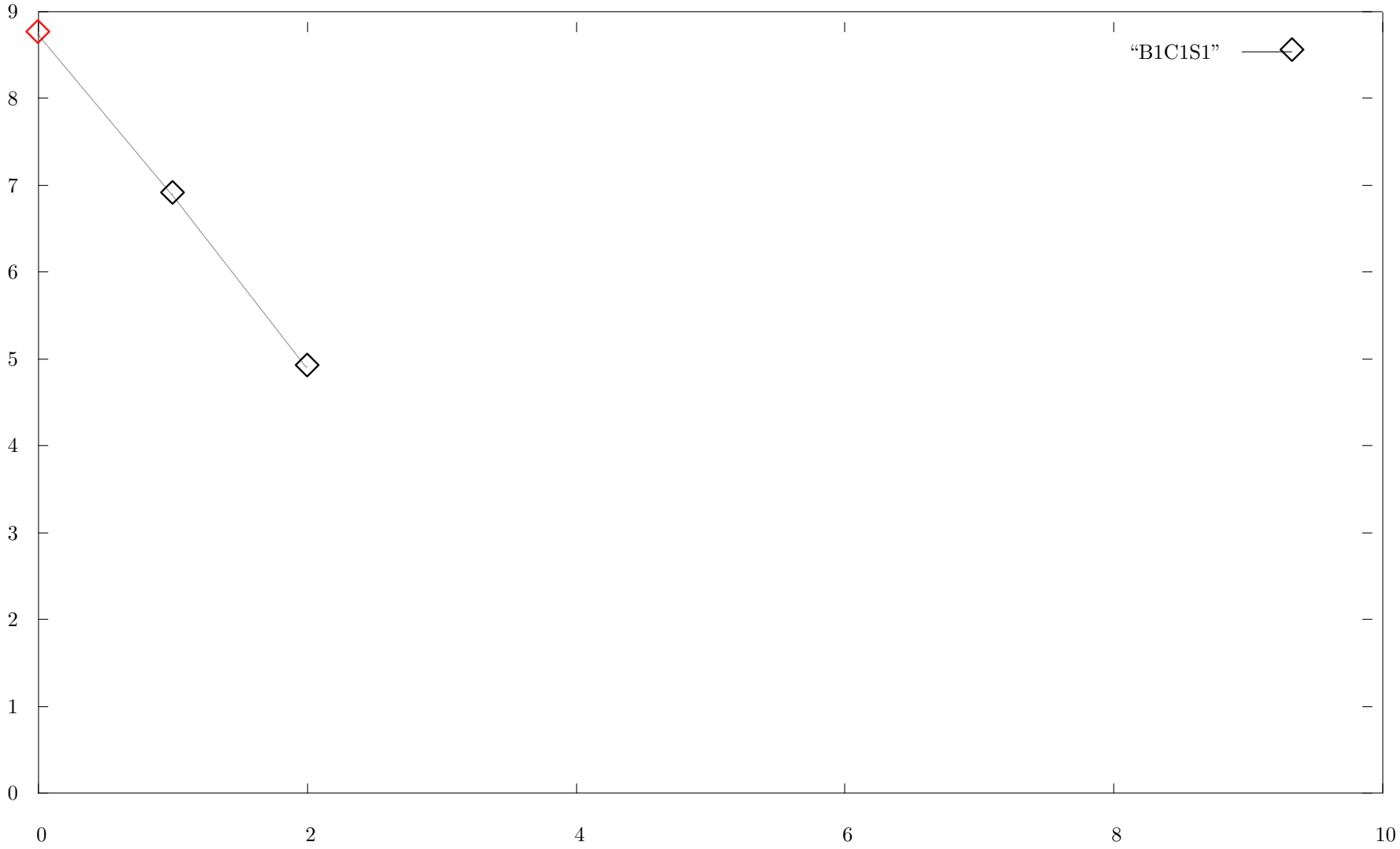
FP: Plot of the infeasibility measure $\Delta(x^*, \widetilde{x})$ at each iteration



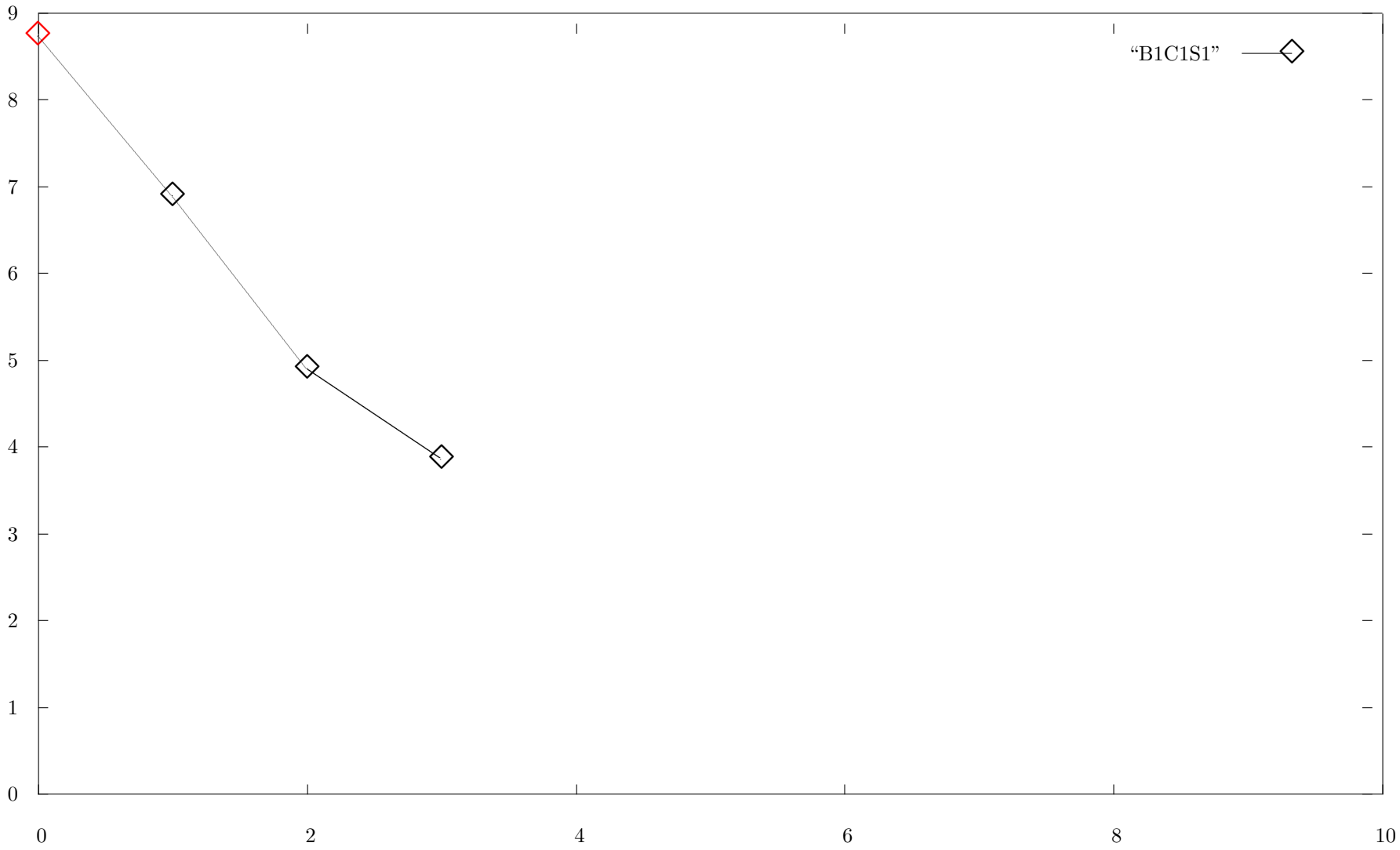
FP: Plot of the infeasibility measure $\Delta(x^*, \widetilde{x})$ at each iteration



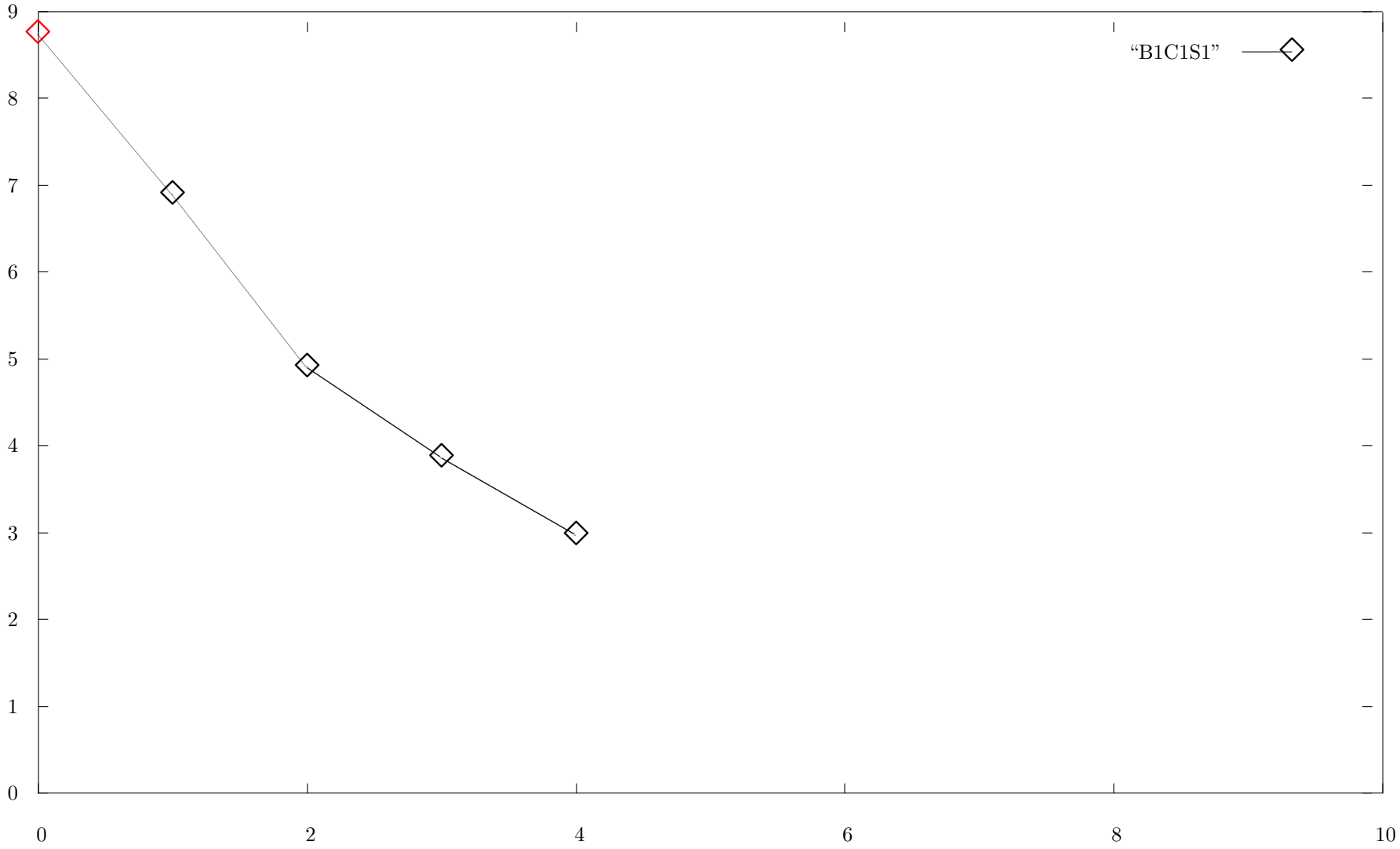
FP: Plot of the infeasibility measure $\Delta(x^*, \widetilde{x})$ at each iteration



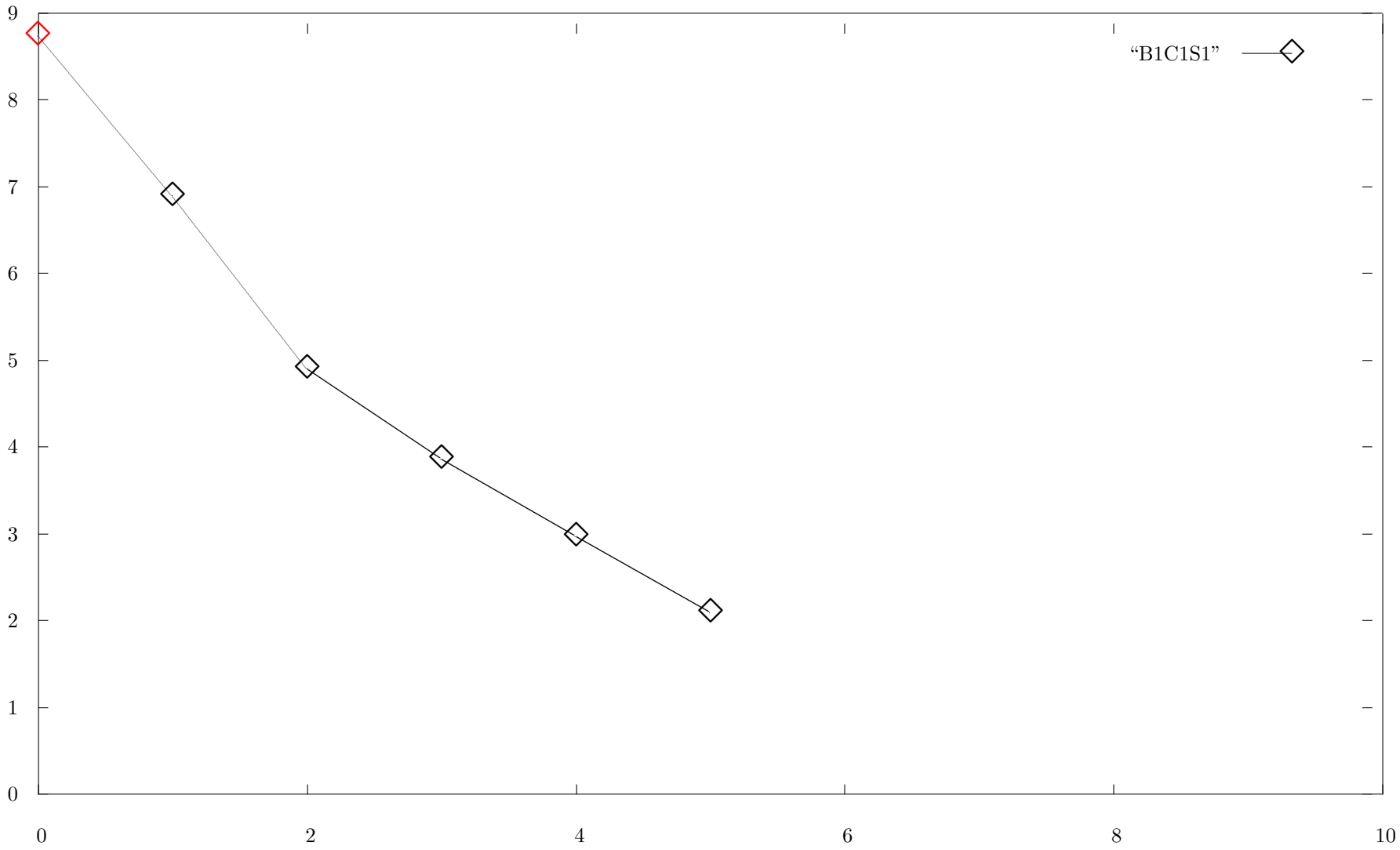
FP: Plot of the infeasibility measure $\Delta(x^*, \widetilde{x})$ at each iteration



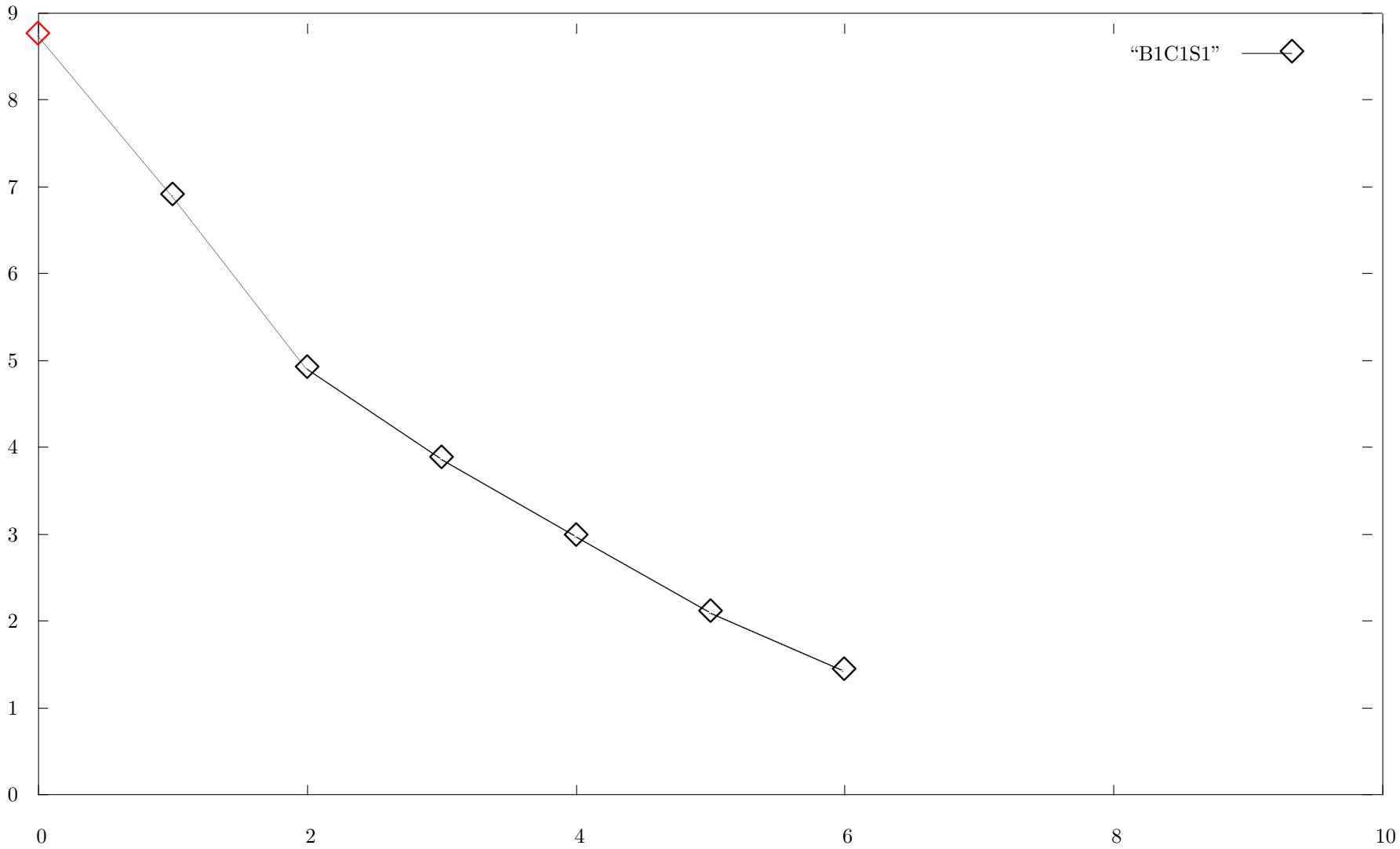
FP: Plot of the infeasibility measure $\Delta(x^*, \widetilde{x})$ at each iteration



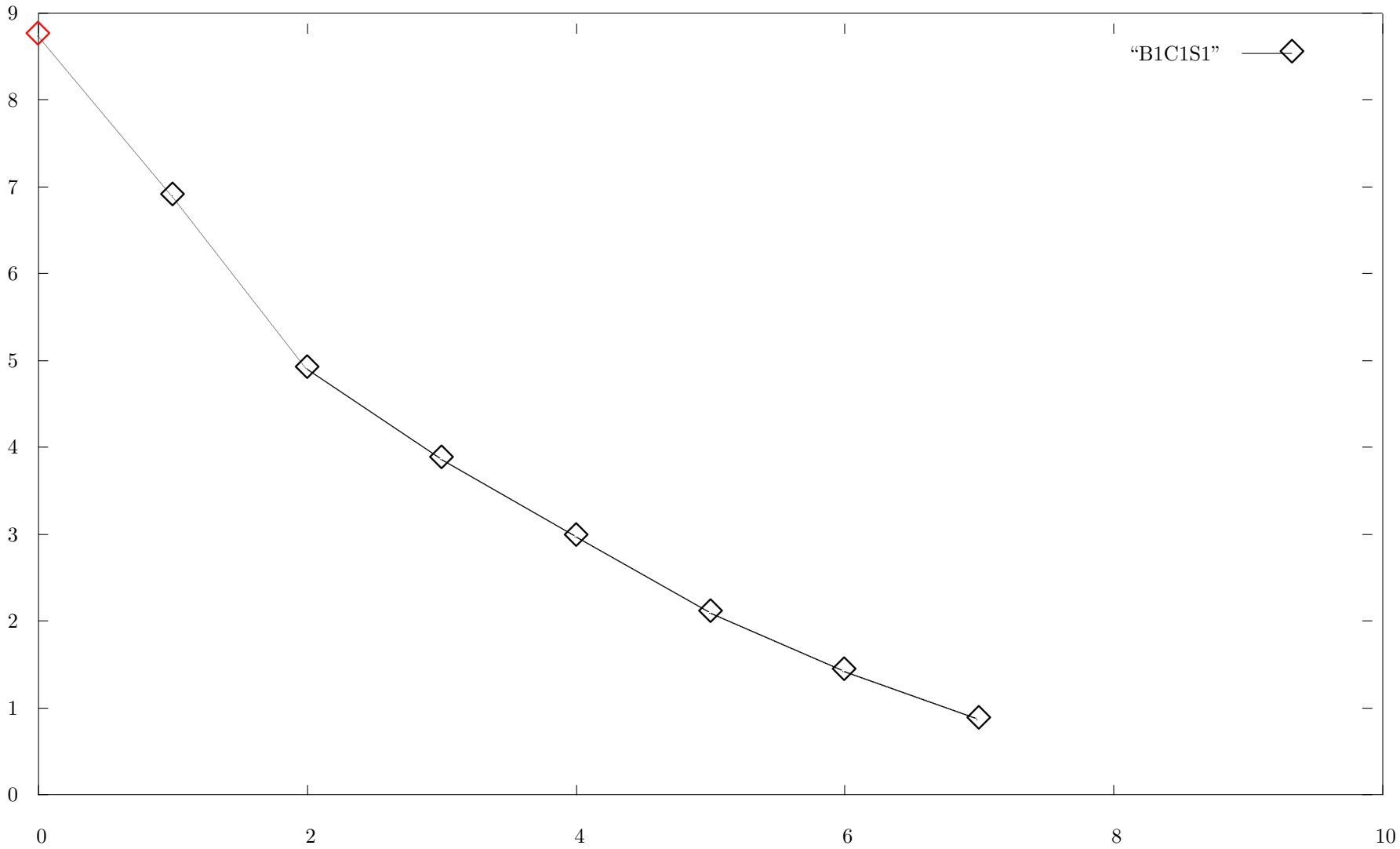
FP: Plot of the infeasibility measure $\Delta(x^*, \widetilde{x})$ at each iteration



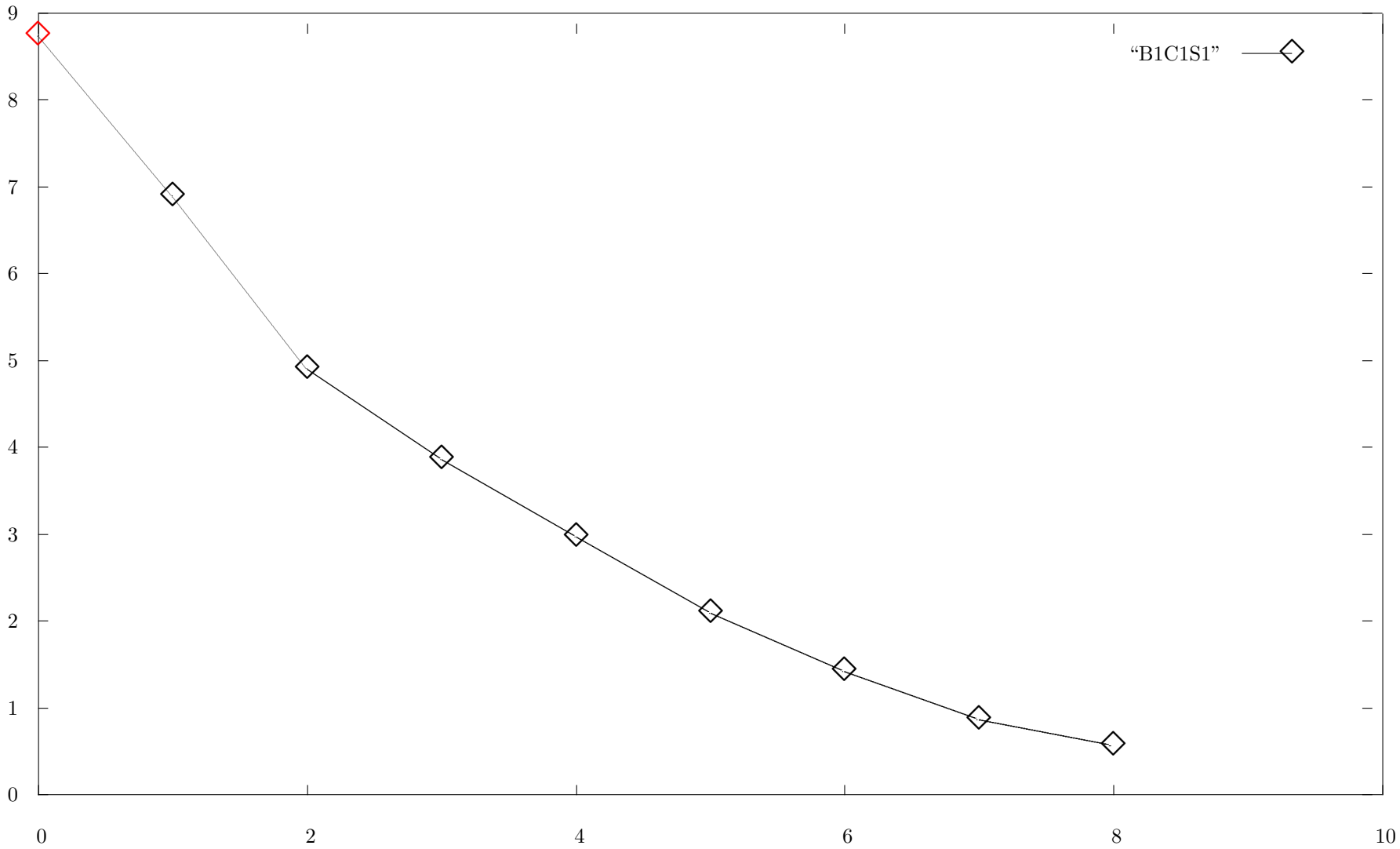
FP: Plot of the infeasibility measure $\Delta(x^*, \widetilde{x})$ at each iteration



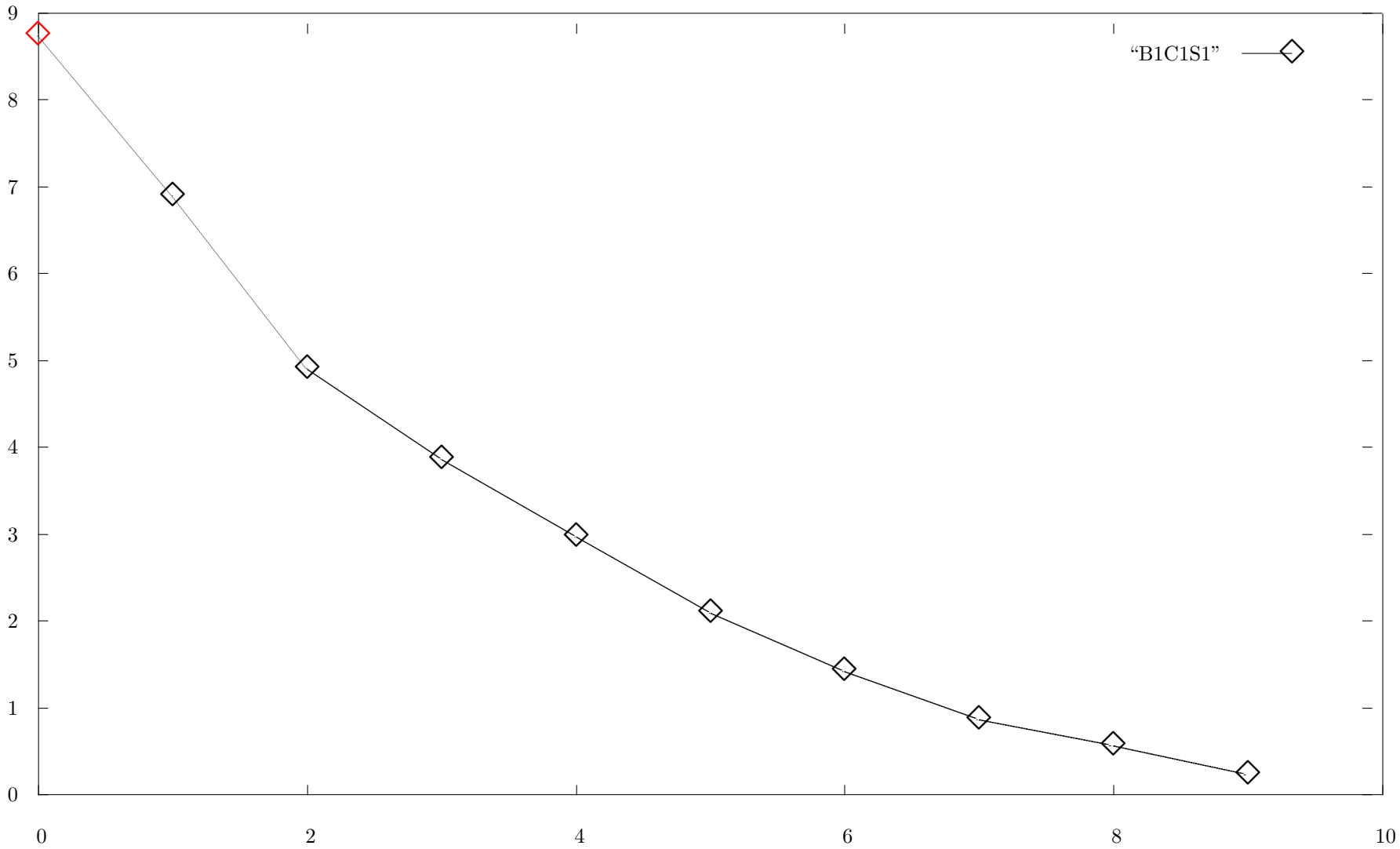
FP: Plot of the infeasibility measure $\Delta(x^*, \widetilde{x})$ at each iteration



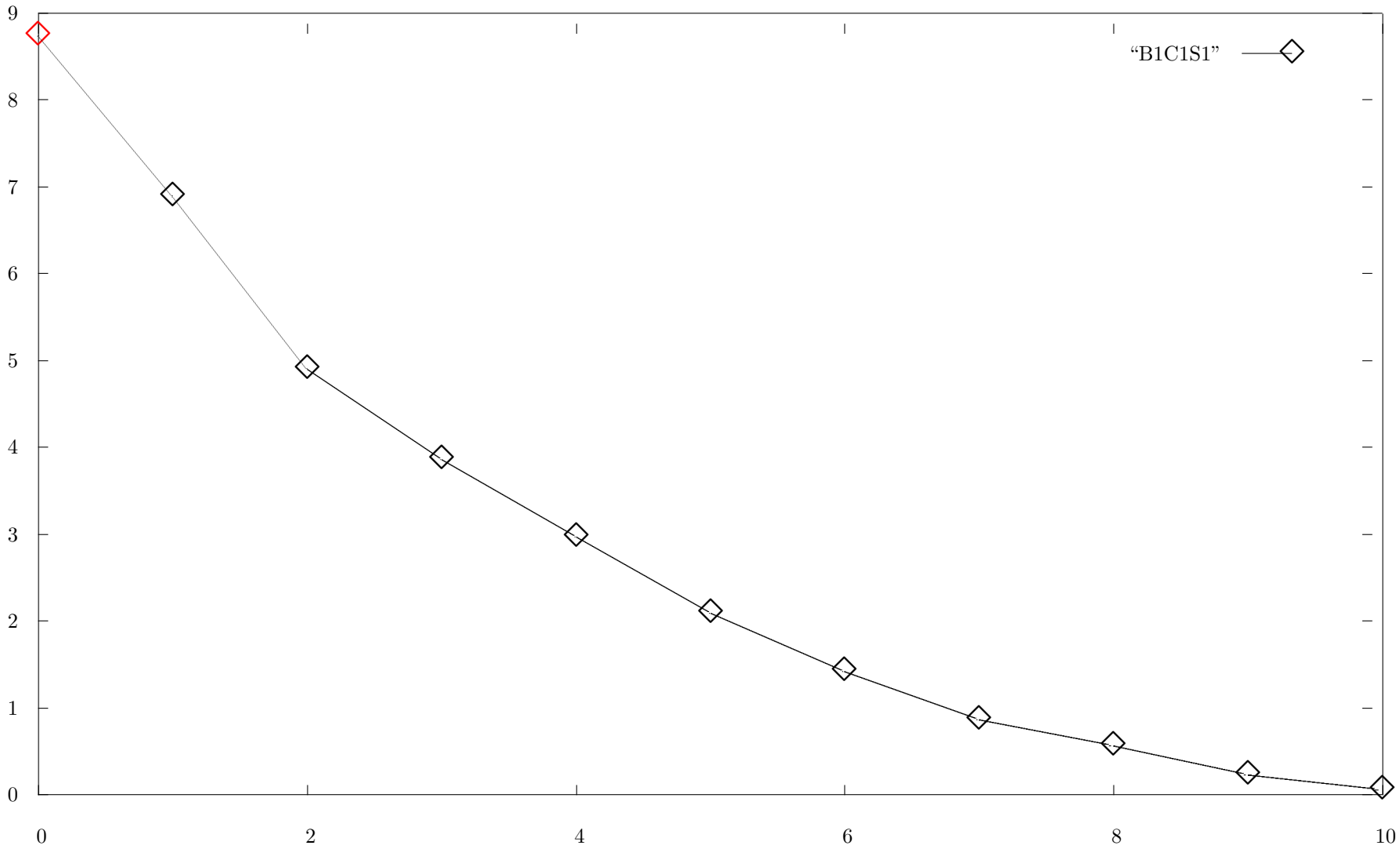
FP: Plot of the infeasibility measure $\Delta(x^*, \widetilde{x})$ at each iteration



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



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



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

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

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

The continuous variables x_j with $j \notin \mathcal{B}$, if any, do not contribute to this function.

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

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

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

  The continuous variables x_j with $j \notin \mathcal{B}$, if any, do not contribute to this function.

- In the case of a binary MIP:

$$\Delta(x, \widetilde{x}) := \sum_{j \in \mathcal{B}: \widetilde{x}_j = 0} x_j + \sum_{j \in \mathcal{B}: \widetilde{x}_j = 1} (1 - x_j) \tag{5}$$

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

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

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

  The continuous variables x_j with $j \notin \mathcal{B}$, if any, do not contribute to this function.

- In the case of a binary MIP:

$$\Delta(x, \widetilde{x}) := \sum_{j \in \mathcal{B}: \widetilde{x}_j = 0} x_j + \sum_{j \in \mathcal{B}: \widetilde{x}_j = 1} (1 - x_j) \tag{5}$$

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

$$\min\{\Delta(x, \widetilde{x}) : Ax \geq b\} \tag{6}$$

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

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

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

  The continuous variables x_j with $j \notin \mathcal{B}$, if any, do not contribute to this function.

- In the case of a binary MIP:

$$\Delta(x, \widetilde{x}) := \sum_{j \in \mathcal{B}: \widetilde{x}_j = 0} x_j + \sum_{j \in \mathcal{B}: \widetilde{x}_j = 1} (1 - x_j) \tag{5}$$

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

$$\min\{\Delta(x, \widetilde{x}) : Ax \geq b\} \tag{6}$$

- We like to see such a distance as a difference of pressure between the two complementary infeasibility of x^* and $\widetilde{x}$, that we try to reduce by pumping the integrality of $\widetilde{x}$ into x^*.

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

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

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

  The continuous variables x_j with $j \notin \mathcal{B}$, if any, do not contribute to this function.

- In the case of a binary MIP:

$$\Delta(x, \widetilde{x}) := \sum_{j \in \mathcal{B}: \widetilde{x}_j = 0} x_j + \sum_{j \in \mathcal{B}: \widetilde{x}_j = 1} (1 - x_j) \tag{5}$$

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

$$\min\{\Delta(x, \widetilde{x}) : Ax \geq b\} \tag{6}$$

- We like to see such a distance as a difference of pressure between the two complementary infeasibility of x^* and $\widetilde{x}$, that we try to reduce by pumping the integrality of $\widetilde{x}$ into x^*.
- On the other hand, it is clearly a measure of vicinity, therefore a neighborhood.

FP: A first implementation

- MAIN PROBLEM, stalling issues:
  as soon as $\Delta(x^*, \widetilde{x})$ is not reduced when replacing $\widetilde{x}$ by x^*.

  If $\Delta(x^*, \widetilde{x}) > 0$ we still want to modify $\widetilde{x}$, even if this increases its distance from x^*.

  Hence, we reverse the rounding of some variables $x^*_j,\ j \in \mathcal{B}$ chosen so as to minimize the increase in the current value of $\Delta(x^*, \widetilde{x})$.

FP: A first implementation

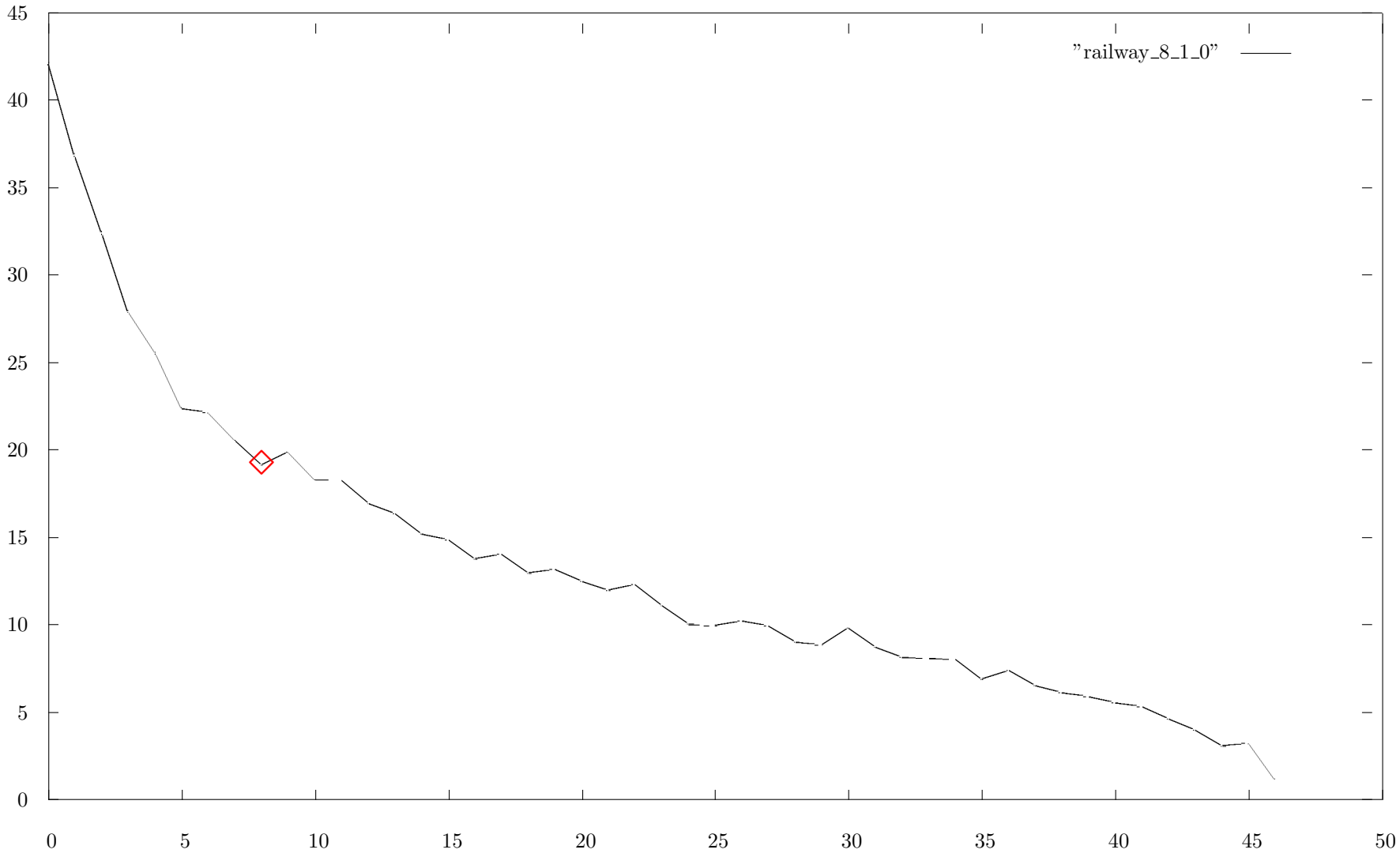
- MAIN PROBLEM, stalling issues:
  as soon as $\Delta(x^*, \widetilde{x})$ is not reduced when replacing $\widetilde{x}$ by x^*.

  If $\Delta(x^*, \widetilde{x}) > 0$ we still want to modify $\widetilde{x}$, even if this increases its distance from x^*.

  Hence, we reverse the rounding of some variables $x^*_j,\ j \in \mathcal{B}$ chosen so as to minimize the increase in the current value of $\Delta(x^*, \widetilde{x})$.

```
1. initialize nIT := 0 and x* := argmin{c^T x : Ax ≥ b};
2. if x* is integer, return(x*);
3. let x̃ := [x*] (= rounding of x*);
4. while (time < TL) do
5.    let nIT := nIT + 1 and compute x* := argmin{Δ(x, x̃) : Ax ≥ b};
6.    if x* is integer, return(x*);
7.    if ∃ j ∈ B : [x*_j] ≠ x̃_j then
8.       x̃ := [x*]
      else
9.       flip the TT = rand(T/2,3T/2) entries x̃_j (j ∈ B) with highest |x*_j − x̃_j|
10.   endif
11. enddo
```

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



FP: better dealing with the objective function

- After the initialization of the algorithm in which the optimal solution x^* of the continuous relaxation is computed, the original objective function $c^T x$ is replaced by the distance function $\Delta(x, \widetilde{x})$.

- It is clear that if the number of FP iterations is large, especially if some random steps are applied, the trajectory can go very "far away" from the initial x^*, with potentially poor values of the original objective function.

- In other words, the search is not guided anymore by $c^T x$.

FP: better dealing with the objective function

- After the initialization of the algorithm in which the optimal solution x^* of the continuous relaxation is computed, the original objective function $c^T x$ is replaced by the distance function $\Delta(x, \widetilde{x})$.

- It is clear that if the number of FP iterations is large, especially if some random steps are applied, the trajectory can go very "far away" from the initial x^*, with potentially poor values of the original objective function.

- In other words, the search is not guided anymore by $c^T x$.

- The above issue can be partially corrected by taking a convex combination of the $\Delta(x, \widetilde{x})$ and $c^T x$ objective functions.

  [Achterberg & Berthold, *DO* 2007]

- Precisely, the combination used is

$$\Delta_\alpha(x, \widetilde{x}) := (1 - \alpha)\Delta(x, \widetilde{x}) + \alpha \frac{\sqrt{|\mathcal{B}|}}{||c||_2} c^T x \quad \text{with } \alpha \in [0, 1] \tag{7}$$

  where α geometrically decreases at every iteration so as to give more and more emphasis on feasibility with respect to optimality.

FP: dealing with general integer variables

- The extension of the FP algorithm from 0-1 MIPs to general integers (variables in $\mathcal{I}$) requires to redefine the objective function $\Delta(x, \widetilde{x})$ appropriately.

FP: dealing with general integer variables

- The extension of the FP algorithm from 0-1 MIPs to general integers (variables in $\mathcal{I}$) requires to redefine the objective function $\Delta(x, \widetilde{x})$ appropriately. Indeed, for a variable, say x_j, which is not sit at a bound, say either ℓ_j or u_j, in an integer infeasible solution $\widetilde{x}$, it is less trivial to penalize its move.

FP: dealing with general integer variables

- The extension of the FP algorithm from 0-1 MIPs to general integers (variables in $\mathcal{I}$) requires to redefine the objective function $\Delta(x, \widetilde{x})$ appropriately. Indeed, for a variable, say x_j, which is not sit at a bound, say either ℓ_j or u_j, in an integer infeasible solution $\widetilde{x}$, it is less trivial to penalize its move.
- A possible updated definition is

$$\Delta(x, \widetilde{x}) := \sum_{j \in \mathcal{I}: \widetilde{x}_j = 0} (x_j - \ell_j) + \sum_{j \in \mathcal{I}: \widetilde{x}_j = 1} (u_j - x_j) + \sum_{j \in \mathcal{I}: \ell_j < \widetilde{x}_j < u_j} d_j \qquad (8)$$

where the artificial variables $d_j (= |x_j - \widetilde{x}_j|)$ satisfy the additional constraints

$$d_j \geq x_j - \widetilde{x}_j \quad \text{and} \quad d_j \geq \widetilde{x}_j - x_j.$$

FP: dealing with general integer variables

- The extension of the FP algorithm from 0-1 MIPs to general integers (variables in $\mathcal{I}$) requires to redefine the objective function $\Delta(x, \widetilde{x})$ appropriately. Indeed, for a variable, say x_j, which is not sit at a bound, say either ℓ_j or u_j, in an integer infeasible solution $\widetilde{x}$, it is less trivial to penalize its move.
- A possible updated definition is

$$\Delta(x, \widetilde{x}) := \sum_{j \in \mathcal{I}: \widetilde{x}_j = 0} (x_j - \ell_j) + \sum_{j \in \mathcal{I}: \widetilde{x}_j = 1} (u_j - x_j) + \sum_{j \in \mathcal{I}: \ell_j < \widetilde{x}_j < u_j} d_j \qquad (8)$$

  where the artificial variables $d_j (= |x_j - \widetilde{x}_j|)$ satisfy the additional constraints

$$d_j \geq x_j - \widetilde{x}_j \quad \text{and} \quad d_j \geq \widetilde{x}_j - x_j.$$

  [Bertacco, Fischetti & L., *DO* 2007]

- In addition, the FP procedure is split into three parts:
  1. Binaries first: the integrality requirement of the variables in $\mathcal{I} \setminus \mathcal{B}$ is relaxed.
  2. The general integer: the requirement is reinstalled.
  3. An (truncated) enumeration phase is performed by solving the MIP with the original constraint and the objective function (8) which uses as $\widetilde{x}$ the "least infeasible" integer solution obtained so far.

The Mixed-Integer NON Linear case

- We consider here a Mixed Integer Non Linear Program of the form:

$$\text{MINLP} \begin{cases} \min f(x,y) \\ s.t.: \\ g(x,y) \leq b \\ x \in \mathbb{Z}^{n_1} \\ y \in \mathbb{R}^{n_2} \end{cases}$$

where f is a function from $\mathbb{R}^{n_1} \times \mathbb{R}^{n_2}$ to $\mathbb{R}$ and g is a function from $\mathbb{R}^{n_1} \times \mathbb{R}^{n_2}$ to $\mathbb{R}^m$. We assume the feasible region $g(x,y) \leq b$ to be convex but we allow the single functions g_i to be nonconvex.

The Mixed-Integer NON Linear case

- We consider here a Mixed Integer Non Linear Program of the form:

$$\text{MINLP} \begin{cases} \min f(x,y) \\ s.t.: \\ g(x,y) \leq b \\ x \in \mathbb{Z}^{n_1} \\ y \in \mathbb{R}^{n_2} \end{cases}$$

  where f is a function from $\mathbb{R}^{n_1} \times \mathbb{R}^{n_2}$ to $\mathbb{R}$ and g is a function from $\mathbb{R}^{n_1} \times \mathbb{R}^{n_2}$ to $\mathbb{R}^m$. We assume the feasible region $g(x,y) \leq b$ to be convex but we allow the single functions g_i to be nonconvex.

- The Outer Approximation (OA) technique linearizes the constraints of the continuous relaxation of MINLP to build a mixed integer linear relaxation of MINLP. [Duran & Grossmann, 1986]

The Mixed-Integer NON Linear case

- We consider here a Mixed Integer Non Linear Program of the form:

$$\text{MINLP} \begin{cases} \min f(x,y) \\ s.t.: \\ g(x,y) \leq b \\ x \in \mathbb{Z}^{n_1} \\ y \in \mathbb{R}^{n_2} \end{cases}$$

  where f is a function from $\mathbb{R}^{n_1} \times \mathbb{R}^{n_2}$ to $\mathbb{R}$ and g is a function from $\mathbb{R}^{n_1} \times \mathbb{R}^{n_2}$ to $\mathbb{R}^m$. We assume the feasible region $g(x,y) \leq b$ to be convex but we allow the single functions g_i to be nonconvex.

- The Outer Approximation (OA) technique linearizes the constraints of the continuous relaxation of MINLP to build a mixed integer linear relaxation of MINLP. [Duran & Grossmann, 1986]

- The idea is to combine such a linearization technique with a FP-type algorithm to obtain feasible solutions for MINLPs. Let's call such an algorithm FP-NLP.

  [Bonami, Cornuéjols, L. & Margot, *MPA* 2009]

FP-NLP: the basic scheme

- We start by any feasible solution of the continuous relaxation of MINLP, $(\overline{x}^0, \overline{y}^0)$.

FP-NLP: the basic scheme

- We start by any feasible solution of the continuous relaxation of MINLP, $(\overline{x}^0, \overline{y}^0)$.
- In each iteration $i \geq 1$, we compute $(\hat{x}^i, \hat{y}^i)$ by finding the closest point to $\overline{x}^{i-1}$ (using the L_1 norm) on the current outer approximation $(OA)^i$ of the problem:

FP-NLP: the basic scheme

- We start by any feasible solution of the continuous relaxation of MINLP, $(\overline{x}^0, \overline{y}^0)$.
- In each iteration $i \geq 1$, we compute $(\hat{x}^i, \hat{y}^i)$ by finding the closest point to $\overline{x}^{i-1}$ (using the L_1 norm) on the current outer approximation $(OA)^i$ of the problem:

$$\text{(FOA)}^i \quad \begin{cases} \min \|x - \overline{x}^{i-1}\|_1 \\ g(\overline{x}^k, \overline{y}^k) + J_g(\overline{x}^k, \overline{y}^k) \left(\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} \overline{x}^k \\ \overline{y}^k \end{pmatrix} \right) \leq b \quad \forall k = 0, \ldots, i-1 \\ x \in \mathbb{Z}^{n_1} \\ y \in \mathbb{R}^{n_2}. \end{cases}$$

FP-NLP: the basic scheme

- We start by any feasible solution of the continuous relaxation of MINLP, $(\overline{x}^0, \overline{y}^0)$.
- In each iteration $i \geq 1$, we compute $(\hat{x}^i, \hat{y}^i)$ by finding the closest point to $\overline{x}^{i-1}$ (using the L_1 norm) on the current outer approximation $(OA)^i$ of the problem:

$$(\text{FOA})^i \quad \begin{cases} \min \|x - \overline{x}^{i-1}\|_1 \\ g(\overline{x}^k, \overline{y}^k) + J_g(\overline{x}^k, \overline{y}^k) \left(\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} \overline{x}^k \\ \overline{y}^k \end{pmatrix} \right) \leq b \quad \forall k = 0, \ldots, i-1 \\ x \in \mathbb{Z}^{n_1} \\ y \in \mathbb{R}^{n_2}. \end{cases}$$

- We then compute the next $(\overline{x}^i, \overline{y}^i)$ by solving the NLP:

FP-NLP: the basic scheme

- We start by any feasible solution of the continuous relaxation of MINLP, $(\overline{x}^0, \overline{y}^0)$.
- In each iteration $i \geq 1$, we compute $(\hat{x}^i, \hat{y}^i)$ by finding the closest point to $\overline{x}^{i-1}$ (using the L_1 norm) on the current outer approximation $(OA)^i$ of the problem:

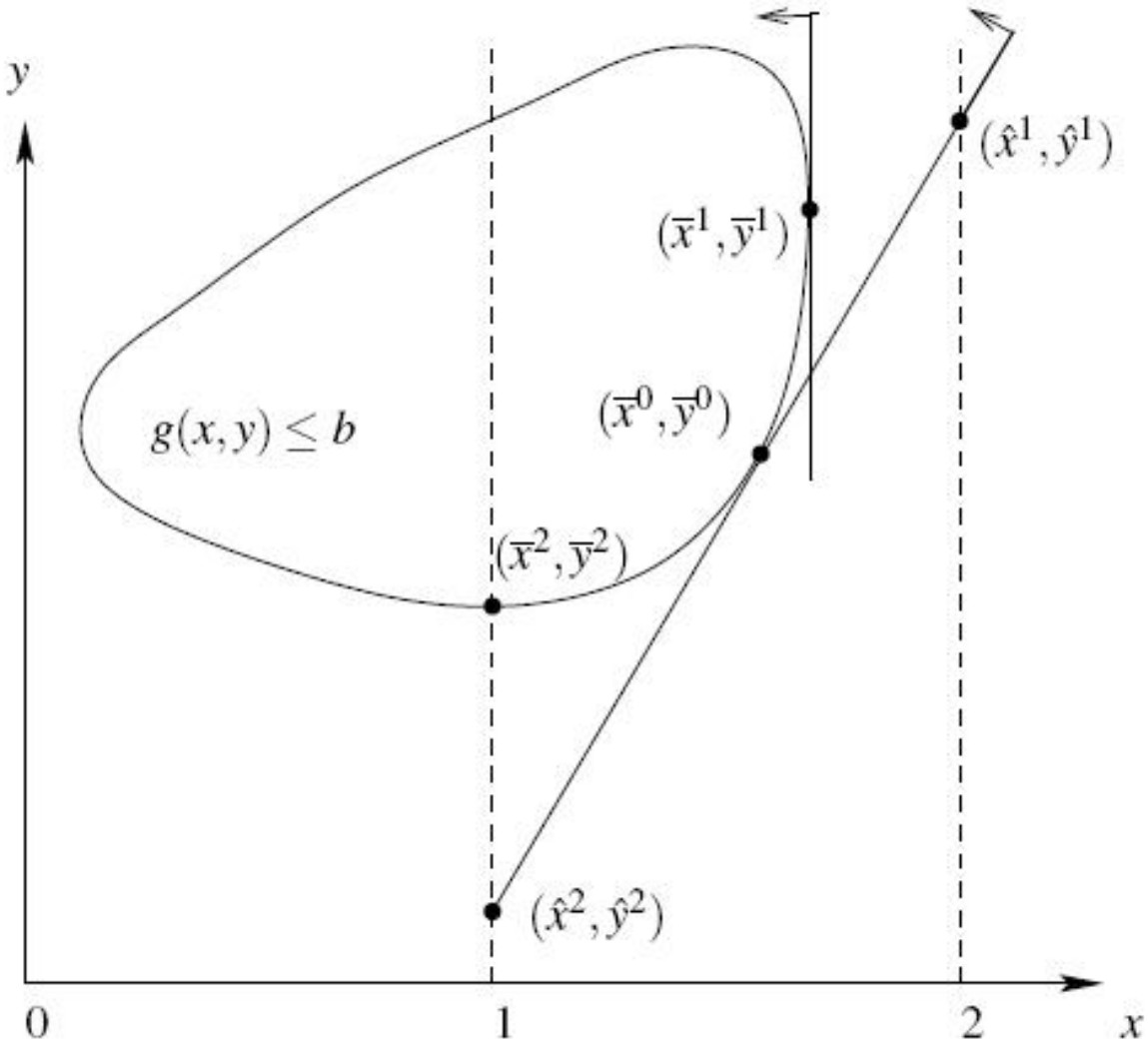
$$(\text{FOA})^i \quad \begin{cases} \min \|x - \overline{x}^{i-1}\|_1 \\ g(\overline{x}^k, \overline{y}^k) + J_g(\overline{x}^k, \overline{y}^k) \left(\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} \overline{x}^k \\ \overline{y}^k \end{pmatrix} \right) \leq b \quad \forall k = 0, \ldots, i-1 \\ x \in \mathbb{Z}^{n_1} \\ y \in \mathbb{R}^{n_2}. \end{cases}$$

- We then compute the next $(\overline{x}^i, \overline{y}^i)$ by solving the NLP:

$$(\text{FP-NLP})^i \quad \begin{cases} \min \|x - \hat{x}^i\|_2 \\ g(x, y) \leq b \\ x \in \mathbb{R}^{n_1} \\ y \in \mathbb{R}^{n_2}. \end{cases}$$

- The procedure iterates between solving $(\text{FOA})^i$ and $(\text{FP-NLP})^i$ until either a feasible solution of MINLP is found or $(\text{FOA})^i$ becomes infeasible.

FP-NLP: a pictorial explanation



FP-NLP: theoretical results

- The next theorem shows that if constraint qualifications hold at each point $(\overline{x}^i, \overline{y}^i)$ the method cannot cycle:

FP-NLP: theoretical results

- The next theorem shows that if constraint qualifications hold at each point $(\overline{x}^i, \overline{y}^i)$ the method cannot cycle:

  Thm: *In the basic FP, let* $(\hat{x}^i, \hat{y}^i)$ *be an optimal solution of (FOA)*^{i} *and* $(\overline{x}^i, \overline{y}^i)$ *an optimal solution of (FP-NLP)*^{i}*. If the constraint qualification for* $(FP - NLP)^i$ *holds at* $(\overline{x}^i, \overline{y}^i)$*, then* $\overline{x}^i \neq \overline{x}^k$ *for all* $k = 0, \ldots, i-1$.

FP-NLP: theoretical results

- The next theorem shows that if constraint qualifications hold at each point $(\overline{x}^i, \overline{y}^i)$ the method cannot cycle:

  Thm: *In the basic FP, let $(\hat{x}^i, \hat{y}^i)$ be an optimal solution of (FOA)^i and $(\overline{x}^i, \overline{y}^i)$ an optimal solution of (FP-NLP)^i. If the constraint qualification for $(FP - NLP)^i$ holds at $(\overline{x}^i, \overline{y}^i)$, then $\overline{x}^i \neq \overline{x}^k$ for all $k = 0, \ldots, i-1$.*

- Otherwise, when the point $(\overline{x}^i, \overline{y}^i)$ is not integer feasible and does not satisfy the constraint qualifications of (FP-NLP)^i, let $h(x) = ||x - \hat{x}^i||_2$.

FP-NLP: theoretical results

- The next theorem shows that if constraint qualifications hold at each point $(\overline{x}^i, \overline{y}^i)$ the method cannot cycle:

  Thm: *In the basic FP, let $(\hat{x}^i, \hat{y}^i)$ be an optimal solution of (FOA)^i and $(\overline{x}^i, \overline{y}^i)$ an optimal solution of (FP-NLP)^i. If the constraint qualification for $(FP - NLP)^i$ holds at $(\overline{x}^i, \overline{y}^i)$, then $\overline{x}^i \neq \overline{x}^k$ for all $k = 0, \ldots, i-1$.*

- Otherwise, when the point $(\overline{x}^i, \overline{y}^i)$ is not integer feasible and does not satisfy the constraint qualifications of (FP-NLP)^i, let $h(x) = ||x - \hat{x}^i||_2$.
- Then, we add, at iteration k, the following inequality to (FOA)^i:

$$(\overline{x}^k - \hat{x}^k)^T (x - \overline{x}^k) \geq 0 \tag{9}$$

FP-NLP: theoretical results

- The next theorem shows that if constraint qualifications hold at each point $(\overline{x}^i, \overline{y}^i)$ the method cannot cycle:

  Thm: *In the basic FP, let $(\hat{x}^i, \hat{y}^i)$ be an optimal solution of (FOA)^i and $(\overline{x}^i, \overline{y}^i)$ an optimal solution of (FP-NLP)^i. If the constraint qualification for $(FP - NLP)^i$ holds at $(\overline{x}^i, \overline{y}^i)$, then $\overline{x}^i \neq \overline{x}^k$ for all $k = 0, \ldots, i-1$.*

- Otherwise, when the point $(\overline{x}^i, \overline{y}^i)$ is not integer feasible and does not satisfy the constraint qualifications of (FP-NLP)^i, let $h(x) = ||x - \hat{x}^i||_2$.
- Then, we add, at iteration k, the following inequality to (FOA)^i:

$$(\overline{x}^k - \hat{x}^k)^T (x - \overline{x}^k) \geq 0 \tag{9}$$

- This inequality is valid for the continuous relaxation of MINLP and therefore can be added in the outer approximation constraints in (FOA)^i.

FP-NLP: theoretical results

- The next theorem shows that if constraint qualifications hold at each point $(\overline{x}^i, \overline{y}^i)$ the method cannot cycle:

  Thm: *In the basic FP, let $(\hat{x}^i, \hat{y}^i)$ be an optimal solution of (FOA)^i and $(\overline{x}^i, \overline{y}^i)$ an optimal solution of (FP-NLP)^i. If the constraint qualification for $(FP - NLP)^i$ holds at $(\overline{x}^i, \overline{y}^i)$, then $\overline{x}^i \neq \overline{x}^k$ for all $k = 0, \ldots, i-1$.*

- Otherwise, when the point $(\overline{x}^i, \overline{y}^i)$ is not integer feasible and does not satisfy the constraint qualifications of (FP-NLP)^i, let $h(x) = ||x - \hat{x}^i||_2$.
- Then, we add, at iteration k, the following inequality to (FOA)^i:

$$(\overline{x}^k - \hat{x}^k)^T (x - \overline{x}^k) \geq 0 \tag{9}$$

- This inequality is valid for the continuous relaxation of MINLP and therefore can be added in the outer approximation constraints in (FOA)^i.
- Then, we can prove that:

  Thm: *If the integer variables x are bounded, the algorithm enhanced by constraints (9) terminates in a finite number of iterations.*

Heuristics for Feasibility and Optimality in Mixed Integer Programming

John Chinneck

Carleton University, Canada
chinneck@sce.carleton.ca

Andrea Lodi

University of Bologna, Italy
andrea.lodi@unibo.it

CIRRELT Spring School on Logistics, Montréal, May 17th, 2010

Outline

1. Introduction and Orientation (Lodi)

Part I: Achieving Integer-Feasibility Quickly

2. Classic Feasibility-Seeking Algorithms (Chinneck)
3. Active Constraint Variable Selection (Chinneck)
4. Branching to Force Change (Chinneck)
5. The Feasibility Pump (Lodi)

Part II: Reaching (quasi-)Optimality Quickly

6. New Node Selection Rules (Chinneck)
7. Local Branching and RINS (Lodi)

Part III: Analyzing Infeasible MIPs

8. Isolating Infeasible Subsystems (Chinneck)
9. Repairing MIP Infeasibility via Local Branching (Lodi)
10. Conclusions (Chinneck)

Improving the incumbent (feasible) solution

- We consider a generic 0-1 MIP of the form:

$$(P) \qquad \min c^T x \tag{1}$$

$$Ax \geq b \tag{2}$$

$$x_j \in \{0,1\} \quad \forall j \in \mathcal{B} \tag{3}$$

$$x_j \geq 0 \qquad \forall j \in \mathcal{C} \tag{4}$$

Improving the incumbent (feasible) solution

- We consider a generic 0-1 MIP of the form:

$$(P) \qquad \min c^T x \tag{1}$$

$$Ax \geq b \tag{2}$$

$$x_j \in \{0,1\} \quad \forall j \in \mathcal{B} \tag{3}$$

$$x_j \geq 0 \qquad \forall j \in \mathcal{C} \tag{4}$$

and assume now to have a feasible solution, $\bar{x}$ at hand, so-called reference solution (generally the *incumbent*, i.e., the best solution encountered so far), we look for better and better through the `Local Branching` algorithm (LB). [Fischetti & L., *MPB* 2002]

The Local Branching framework

- Given reference solution $\bar{x}$, let $\overline{S} := \{j \in \mathcal{B} : \bar{x}_j = 1\}$ denote the binary support of $\bar{x}$.

The Local Branching framework

- Given reference solution $\bar{x}$, let $\overline{S} := \{j \in \mathcal{B} : \bar{x}_j = 1\}$ denote the binary support of $\bar{x}$.
- For a given positive integer parameter k, we define the *k-OPT neighborhood* $\mathcal{N}(\bar{x}, k)$ of $\bar{x}$ as the set of the feasible solutions of (P) satisfying the additional *local branching constraint*:

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \overline{S}} x_j$$

The Local Branching framework

- Given reference solution $\bar{x}$, let $\overline{S} := \{j \in \mathcal{B} : \bar{x}_j = 1\}$ denote the binary support of $\bar{x}$.
- For a given positive integer parameter k, we define the *k-OPT neighborhood* $\mathcal{N}(\bar{x}, k)$ of $\bar{x}$ as the set of the feasible solutions of (P) satisfying the additional *local branching constraint*:

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \overline{S}} x_j \leq k \tag{5}$$

  where the two terms in left-hand side count the number of binary variables flipping their value (with respect to $\bar{x}$) either from 1 to 0 or from 0 to 1, respectively.

The Local Branching framework

- Given reference solution $\bar{x}$, let $\overline{S} := \{j \in \mathcal{B} : \bar{x}_j = 1\}$ denote the binary support of $\bar{x}$.
- For a given positive integer parameter k, we define the *k-OPT neighborhood* $\mathcal{N}(\bar{x}, k)$ of $\bar{x}$ as the set of the feasible solutions of (P) satisfying the additional *local branching constraint*:

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \overline{S}} x_j \leq k \tag{5}$$

  where the two terms in left-hand side count the number of binary variables flipping their value (with respect to $\bar{x}$) either from 1 to 0 or from 0 to 1, respectively.
- Constraint (5) imposes a maximum Hamming distance of k among the feasible neighbors of $\bar{x}$ and can be used as a branching criterion within an enumerative scheme for (P):

The Local Branching framework

- Given reference solution $\bar{x}$, let $\overline{S} := \{j \in \mathcal{B} : \bar{x}_j = 1\}$ denote the binary support of $\bar{x}$.
- For a given positive integer parameter k, we define the *k-OPT neighborhood* $\mathcal{N}(\bar{x}, k)$ of $\bar{x}$ as the set of the feasible solutions of (P) satisfying the additional *local branching constraint*:

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \overline{S}} x_j \leq k \tag{5}$$

  where the two terms in left-hand side count the number of binary variables flipping their value (with respect to $\bar{x}$) either from 1 to 0 or from 0 to 1, respectively.
- Constraint (5) imposes a maximum Hamming distance of k among the feasible neighbors of $\bar{x}$ and can be used as a branching criterion within an enumerative scheme for (P):

  $\Delta(x, \bar{x}) \leq k$ (left branch)

The Local Branching framework

- Given reference solution $\bar{x}$, let $\overline{S} := \{j \in \mathcal{B} : \bar{x}_j = 1\}$ denote the binary support of $\bar{x}$.
- For a given positive integer parameter k, we define the *k-OPT neighborhood* $\mathcal{N}(\bar{x}, k)$ of $\bar{x}$ as the set of the feasible solutions of (P) satisfying the additional *local branching constraint*:

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \overline{S}} x_j \leq k \tag{5}$$

  where the two terms in left-hand side count the number of binary variables flipping their value (with respect to $\bar{x}$) either from 1 to 0 or from 0 to 1, respectively.
- Constraint (5) imposes a maximum Hamming distance of k among the feasible neighbors of $\bar{x}$ and can be used as a branching criterion within an enumerative scheme for (P):

$$\Delta(x, \bar{x}) \leq k \quad \text{(left branch)} \qquad \textbf{or} \qquad \Delta(x, \bar{x}) \geq k + 1 \quad \text{(right branch)} \tag{6}$$

The Local Branching framework

- Given reference solution $\bar{x}$, let $\overline{S} := \{j \in \mathcal{B} : \bar{x}_j = 1\}$ denote the binary support of $\bar{x}$.
- For a given positive integer parameter k, we define the *k-OPT neighborhood* $\mathcal{N}(\bar{x}, k)$ of $\bar{x}$ as the set of the feasible solutions of (P) satisfying the additional *local branching constraint*:

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \overline{S}} x_j \leq k \tag{5}$$

  where the two terms in left-hand side count the number of binary variables flipping their value (with respect to $\bar{x}$) either from 1 to 0 or from 0 to 1, respectively.

- Constraint (5) imposes a maximum Hamming distance of k among the feasible neighbors of $\bar{x}$ and can be used as a branching criterion within an enumerative scheme for (P):

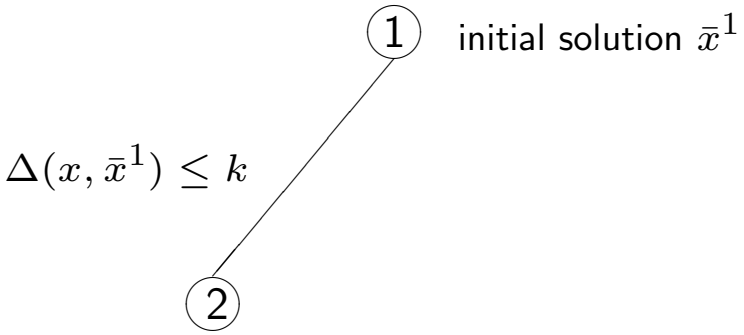
$$\Delta(x, \bar{x}) \leq k \quad \text{(left branch)} \qquad \textbf{or} \qquad \Delta(x, \bar{x}) \geq k + 1 \quad \text{(right branch)} \tag{6}$$

- The neighborhoods defined by the local branching constraints can be searched by using, as a black-box, a MIP solver.

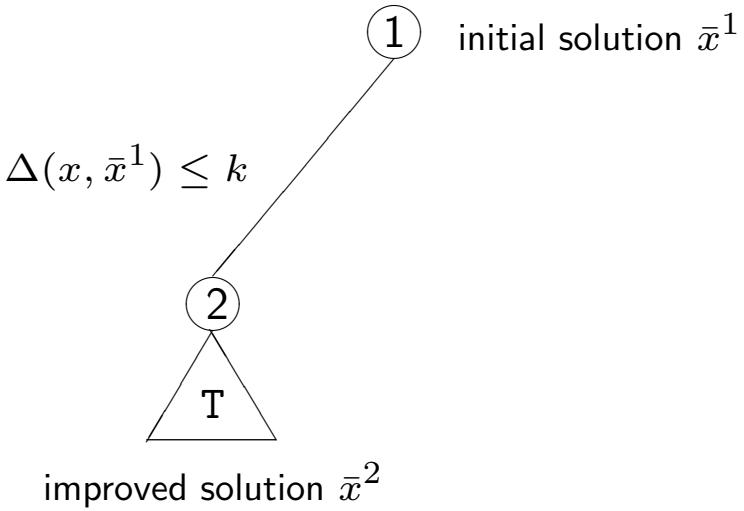
LB: the basic scheme

① initial solution $\bar{x}^1$

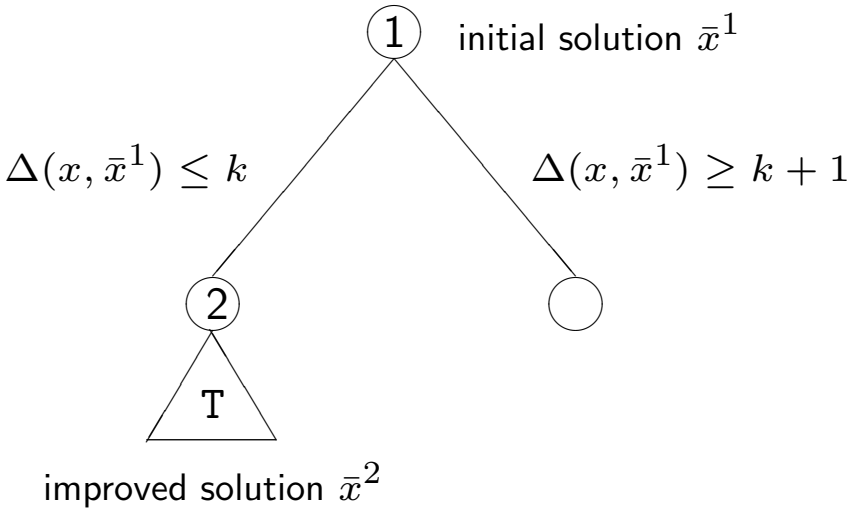
LB: the basic scheme



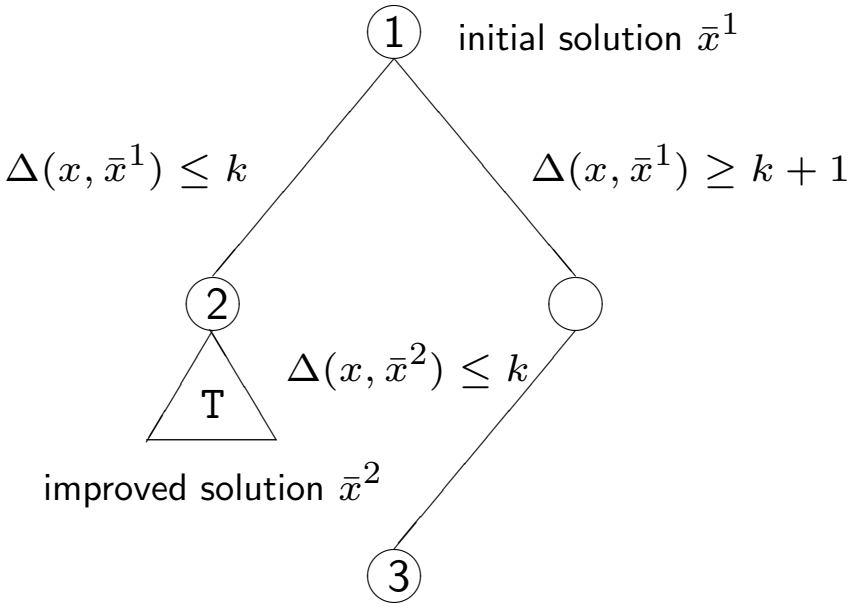
LB: the basic scheme



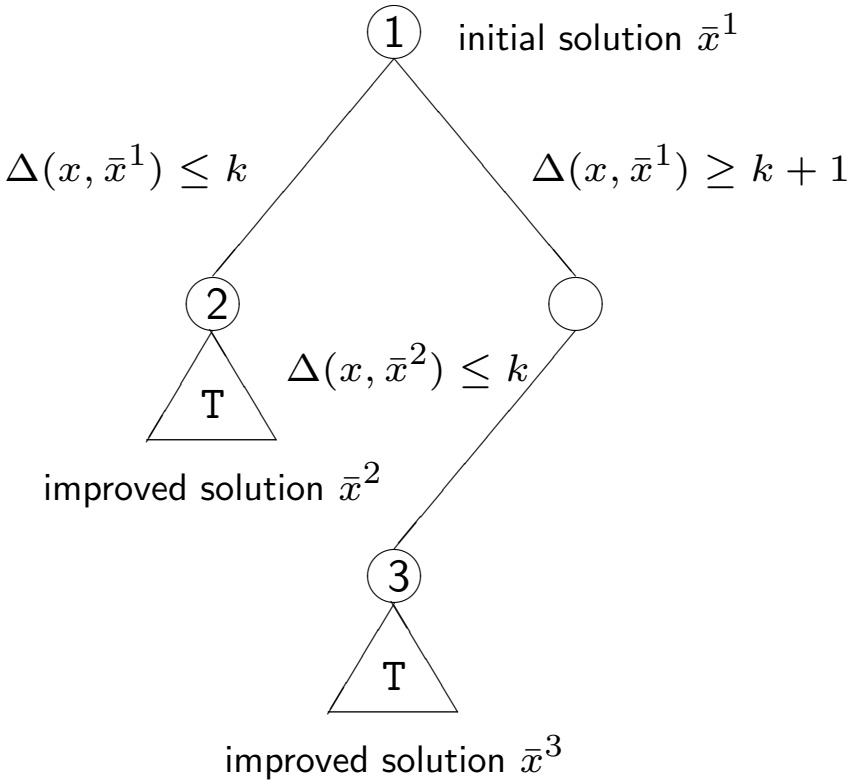
LB: the basic scheme



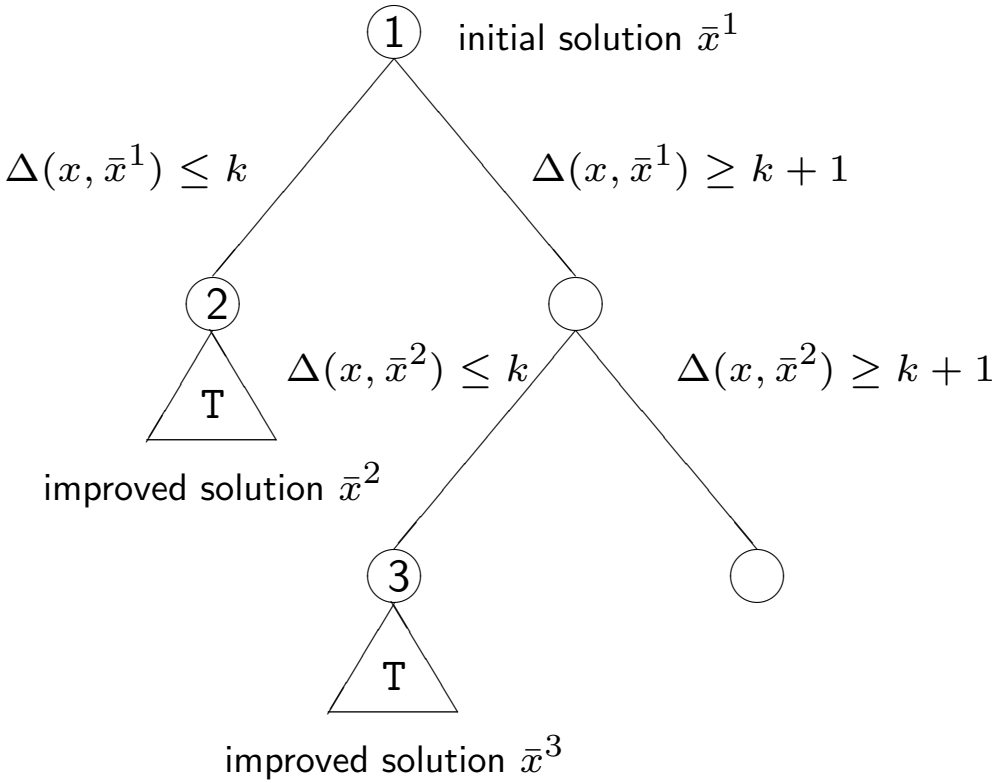
LB: the basic scheme



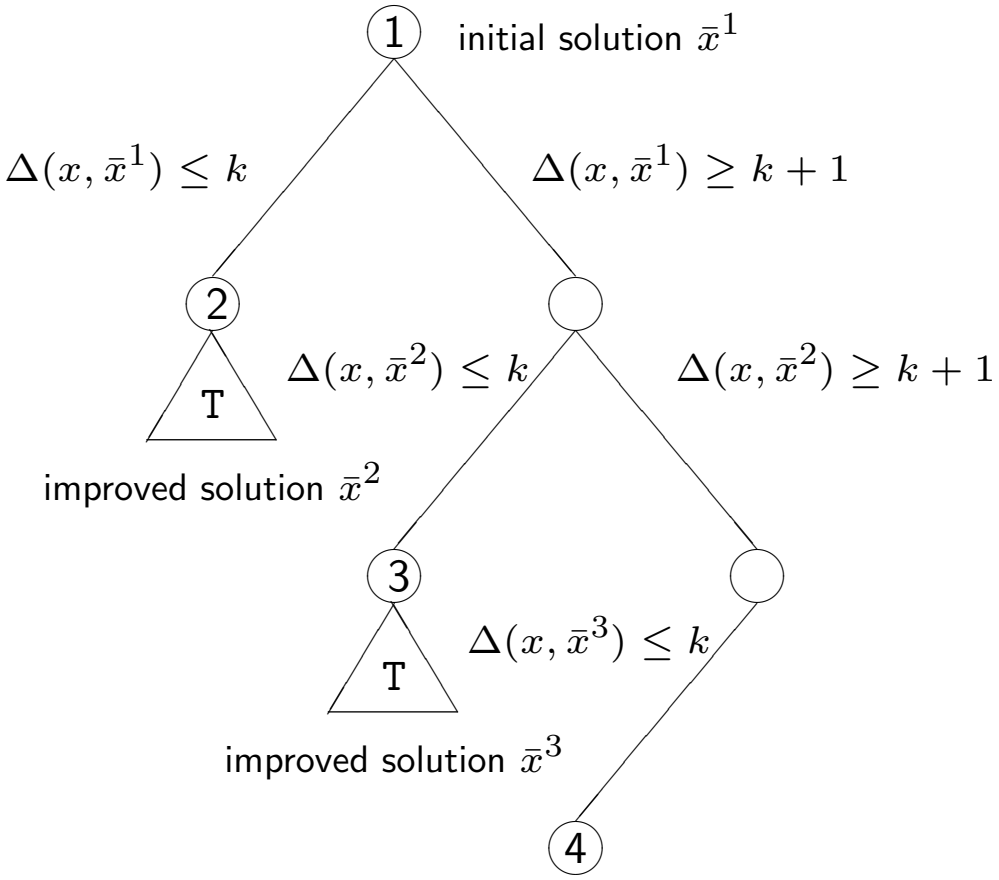
LB: the basic scheme



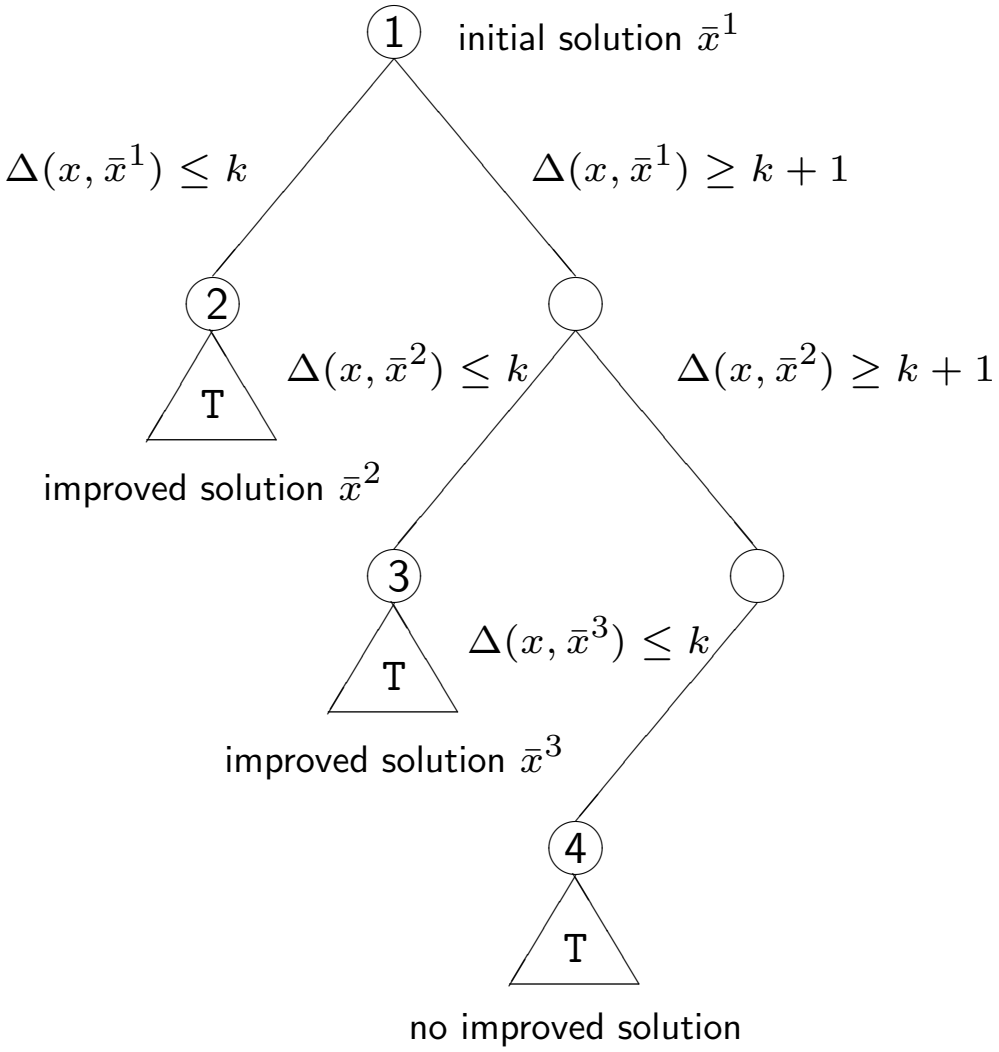
LB: the basic scheme



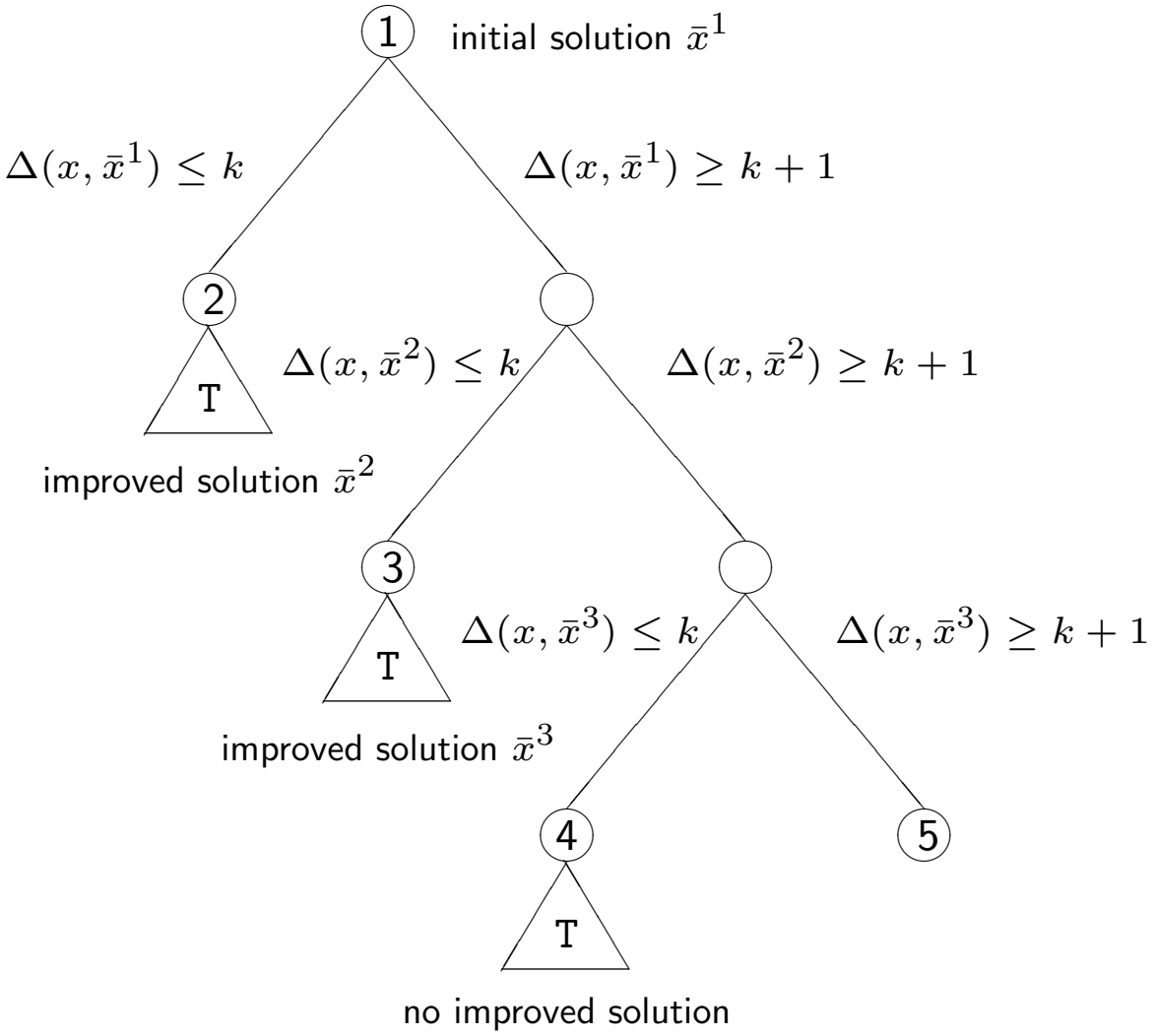
LB: the basic scheme



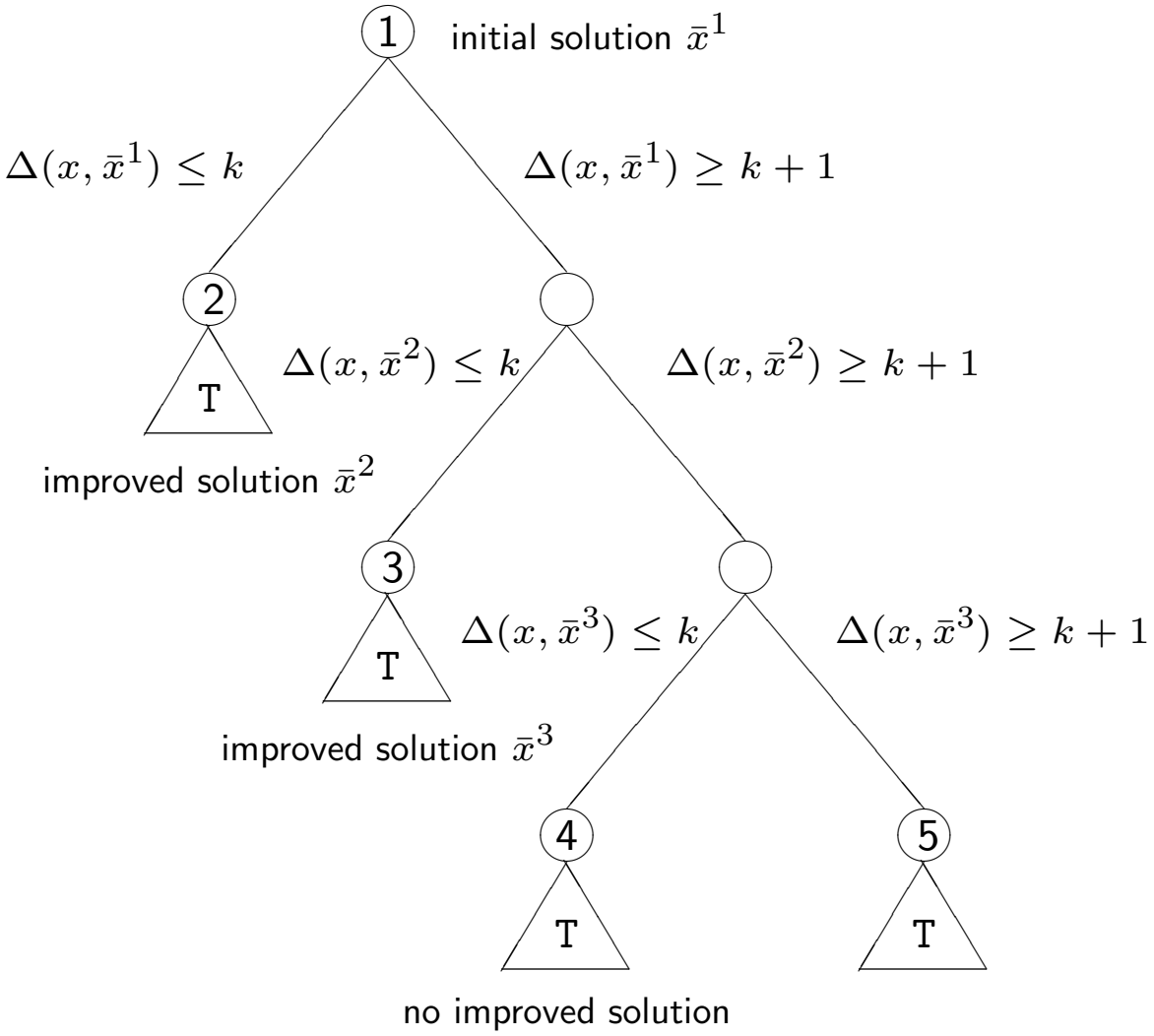
LB: the basic scheme



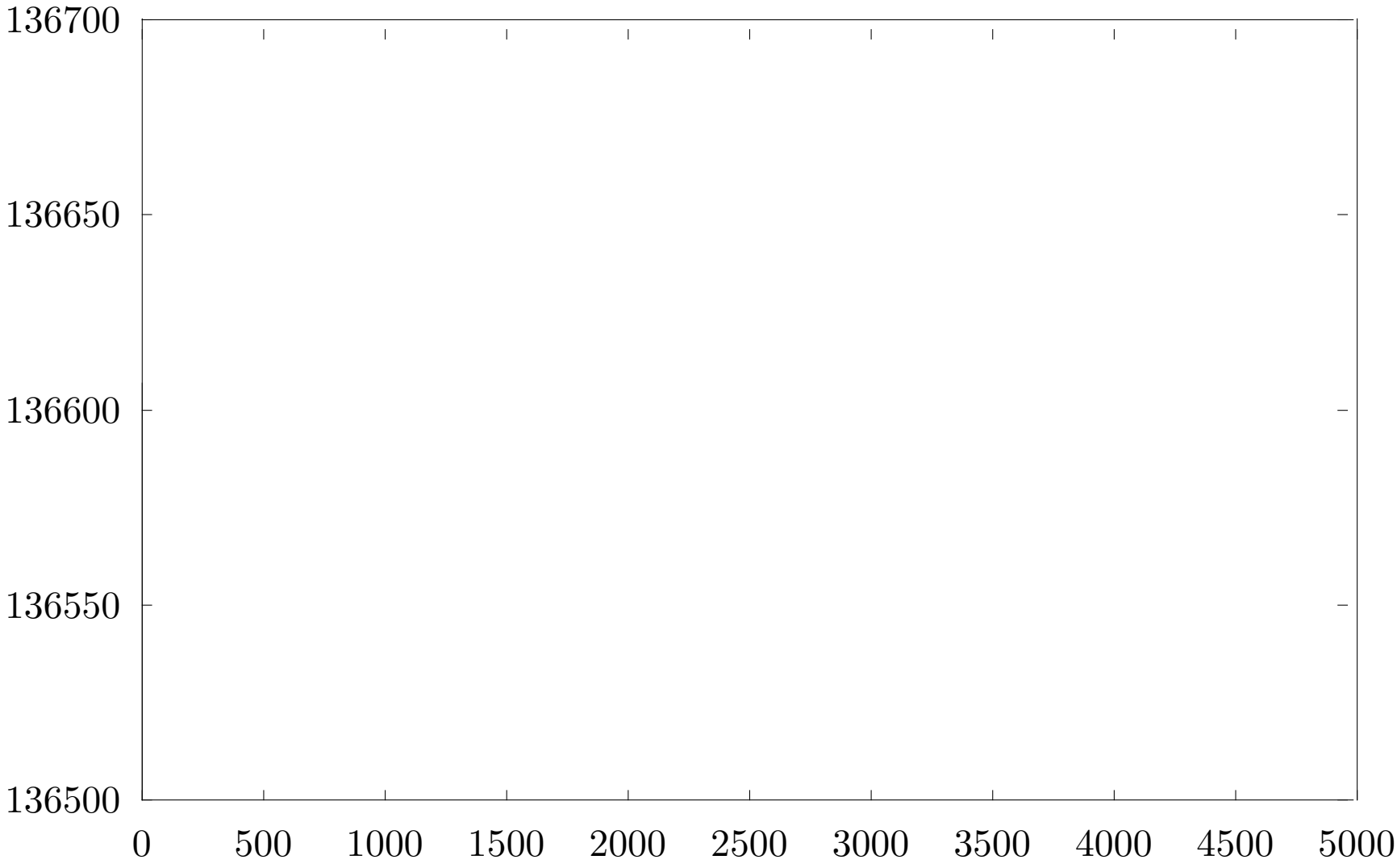
LB: the basic scheme



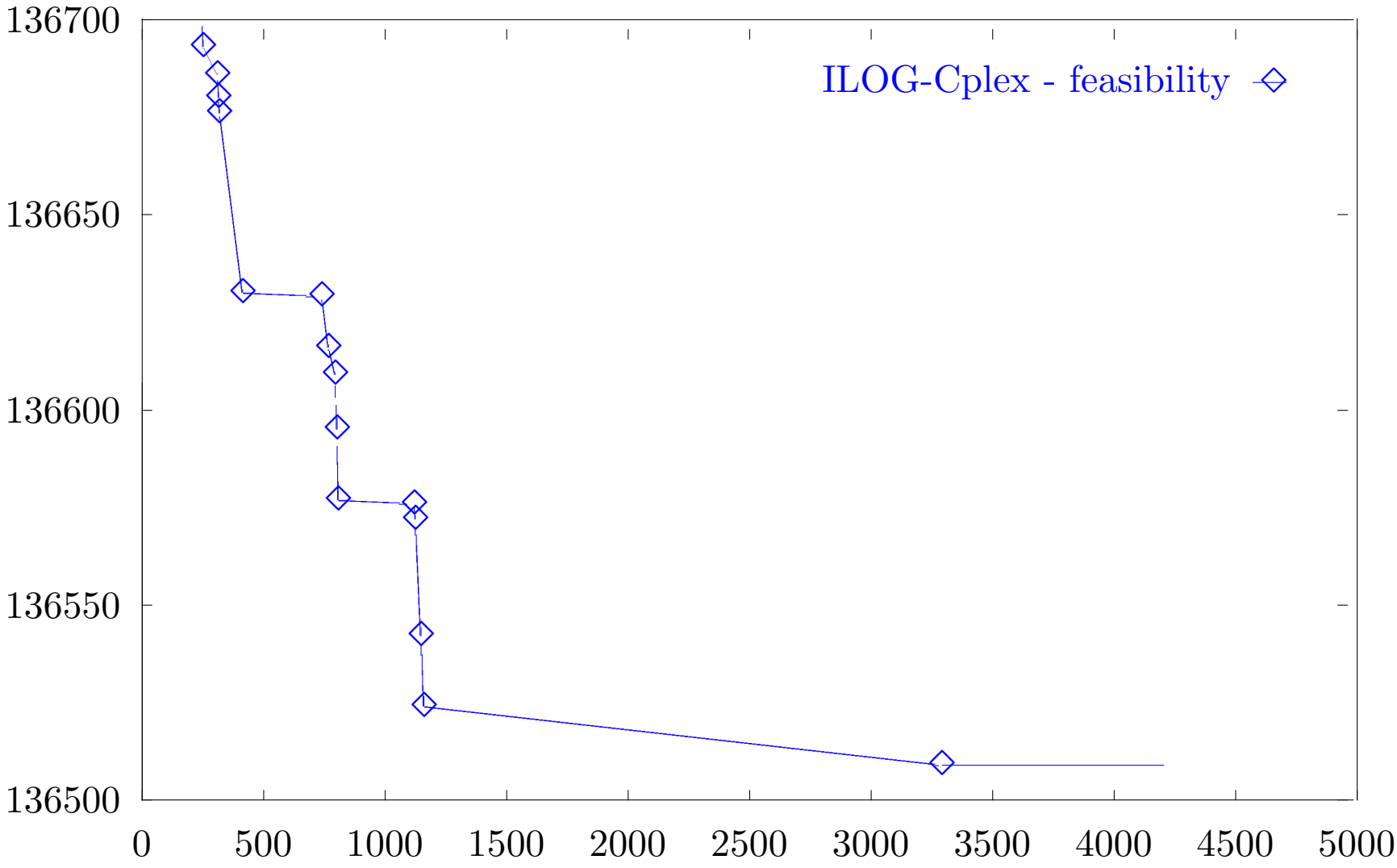
LB: the basic scheme



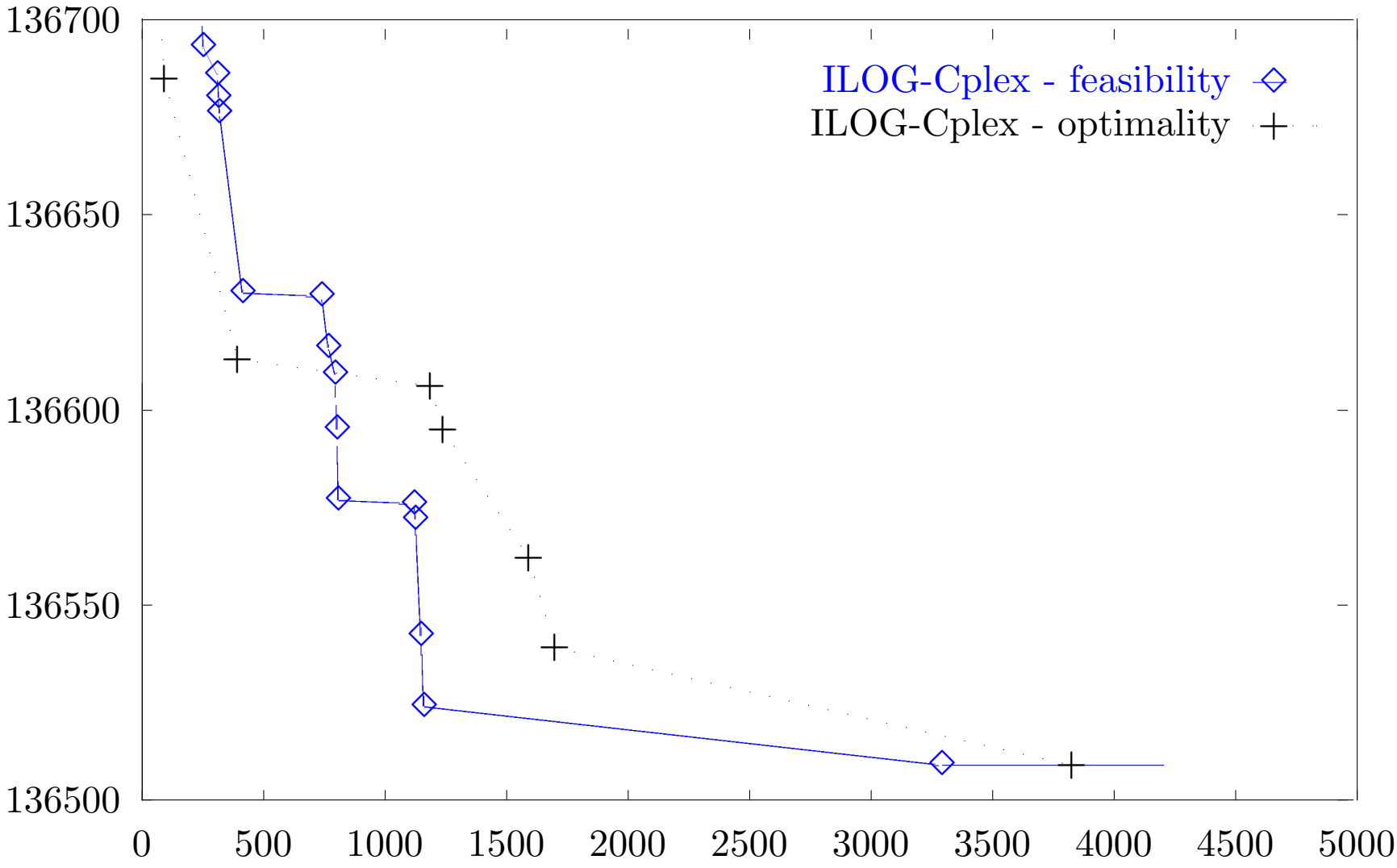
Solving MIP instance `tr24-15` (solution value vs. CPU seconds)



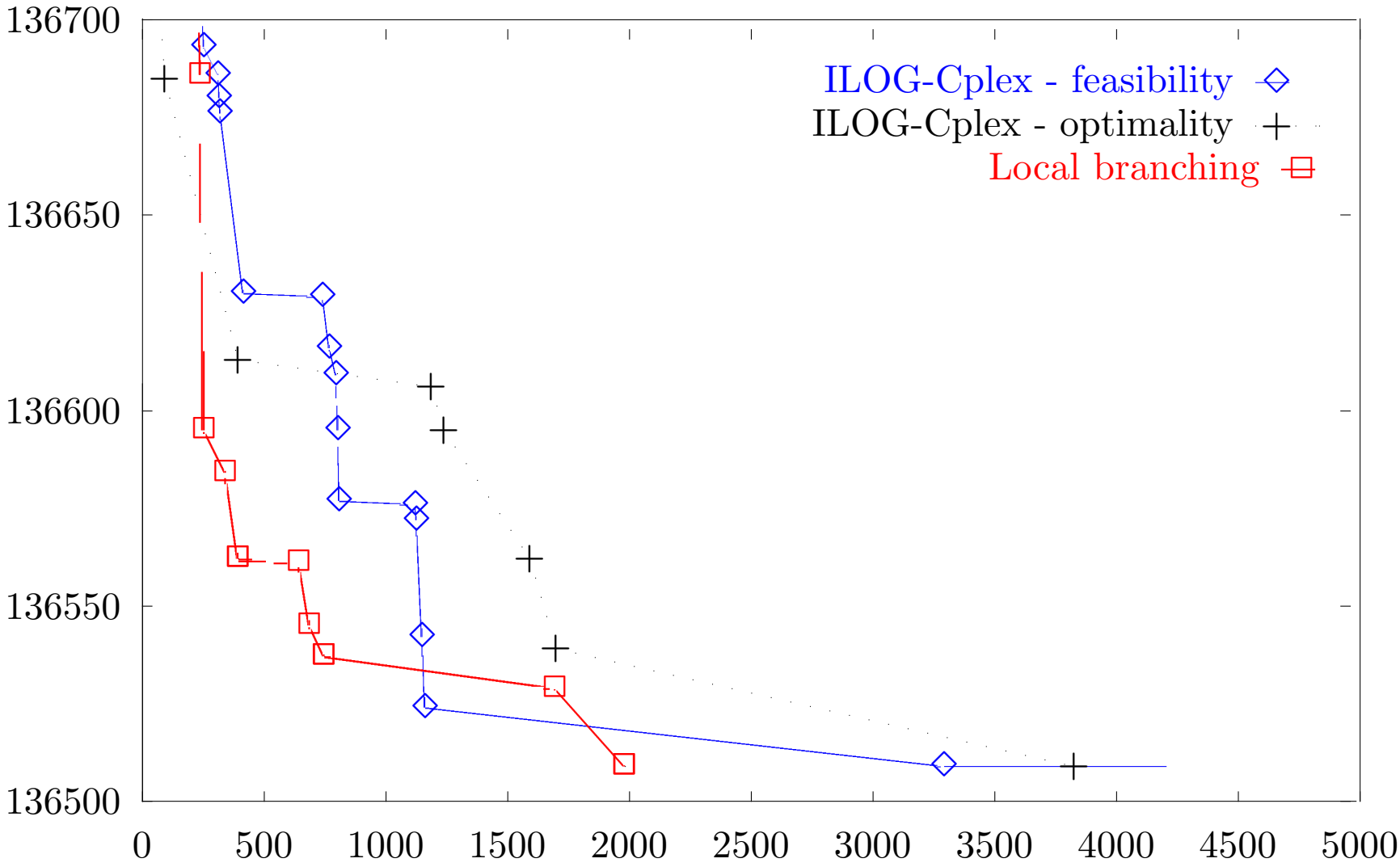
Solving MIP instance `tr24-15` (solution value vs. CPU seconds)



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



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



LB: the heuristic version, time limit at nodes

- The previous scheme can be enhanced from a heuristic viewpoint in two ways:

LB: the heuristic version, time limit at nodes

- The previous scheme can be enhanced from a heuristic viewpoint in two ways:
  - Imposing a time/node limit on the left-branch nodes:

LB: the heuristic version, time limit at nodes

- The previous scheme can be enhanced from a heuristic viewpoint in two ways:
  - Imposing a time/node limit on the left-branch nodes:

    In some cases, the exact solution of the left-branch node can be too time consuming for the value of the parameter k at hand.
    Hence, from the point of view of a heuristic, it is reasonable to impose a time/node limit for the left-branch computation.

LB: the heuristic version, time limit at nodes

- The previous scheme can be enhanced from a heuristic viewpoint in two ways:
  - Imposing a time/node limit on the left-branch nodes:

    In some cases, the exact solution of the left-branch node can be too time consuming for the value of the parameter k at hand.
    Hence, from the point of view of a heuristic, it is reasonable to impose a time/node limit for the left-branch computation.
    In case the time limit is exceeded, we have two cases:
    - **Case (a)**:
      If the incumbent solution has been improved, we backtrack to the father node and create a new left-branch node associated with the new incumbent solution, without modifying the value of parameter k. case (a)

LB: the heuristic version, time limit at nodes

- The previous scheme can be enhanced from a heuristic viewpoint in two ways:
  - Imposing a time/node limit on the left-branch nodes:

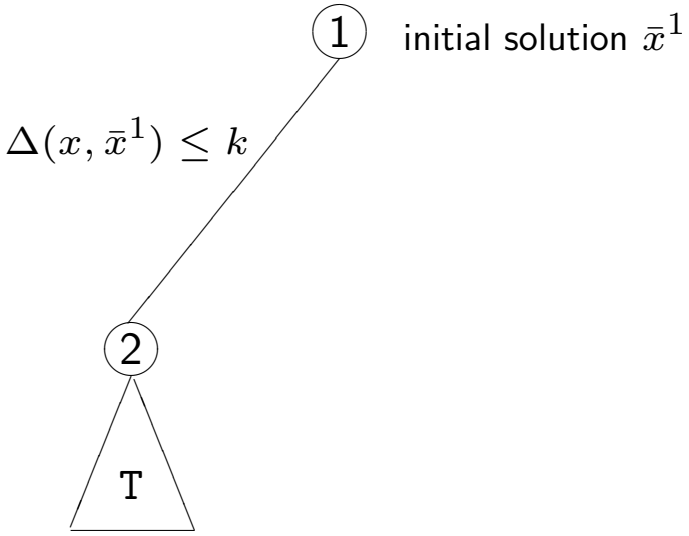
    In some cases, the exact solution of the left-branch node can be too time consuming for the value of the parameter k at hand.
    Hence, from the point of view of a heuristic, it is reasonable to impose a time/node limit for the left-branch computation.
    In case the time limit is exceeded, we have two cases:
    - **Case (a)**:
      If the incumbent solution has been improved, we backtrack to the father node and create a new left-branch node associated with the new incumbent solution, without modifying the value of parameter k. case (a)

    - **Case (b)**:
      If the time limit is reached with no improved solution, instead, we reduce the size of the neighborhood in an attempt to speed-up its exploration.
      This is obtained by reducing the right-hand side term by, e.g., $\lceil k/2 \rceil$. case (b)

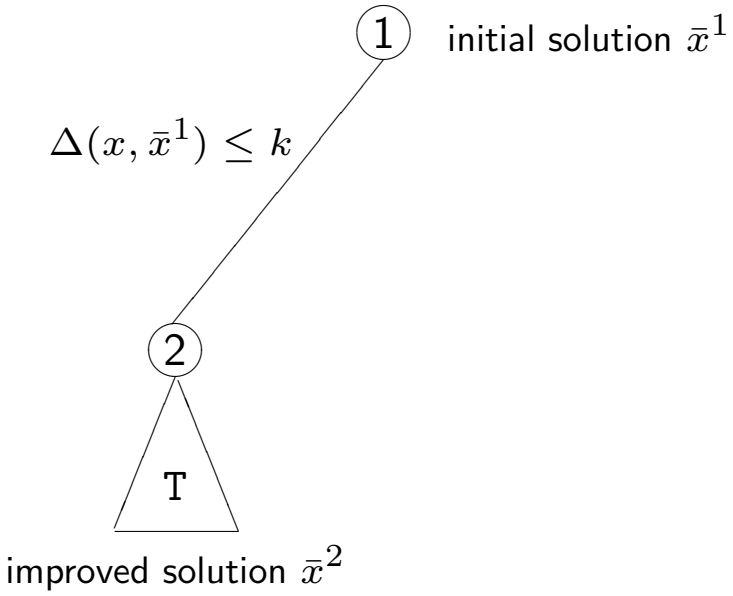
Working with a node time limit: case (a)

① initial solution $\bar{x}^1$

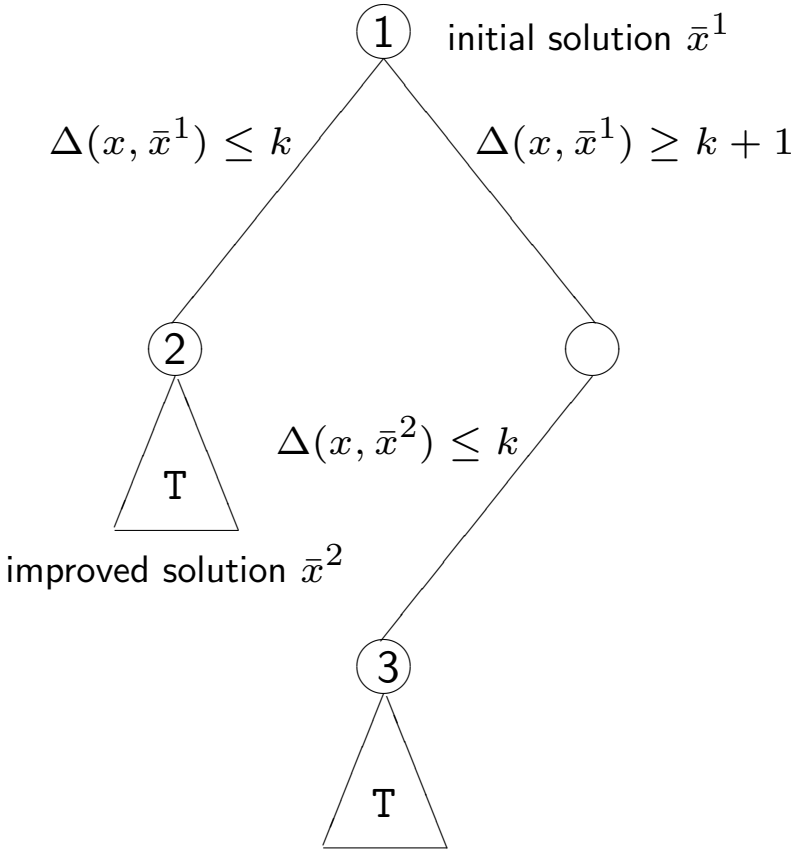
Working with a node time limit: case (a)



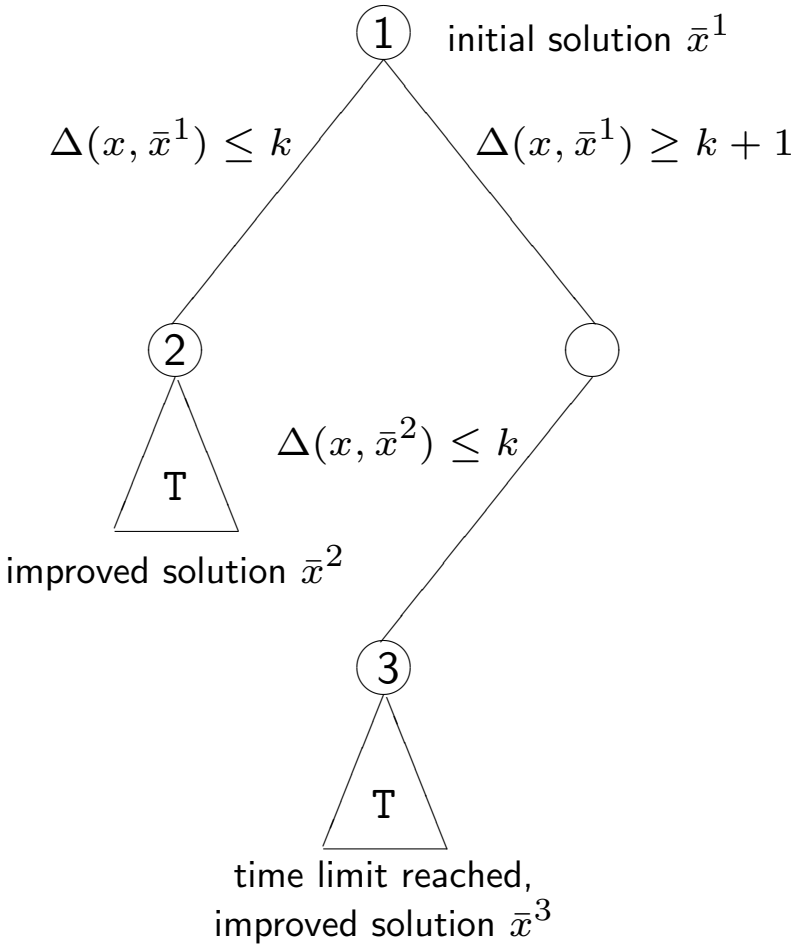
Working with a node time limit: case (a)



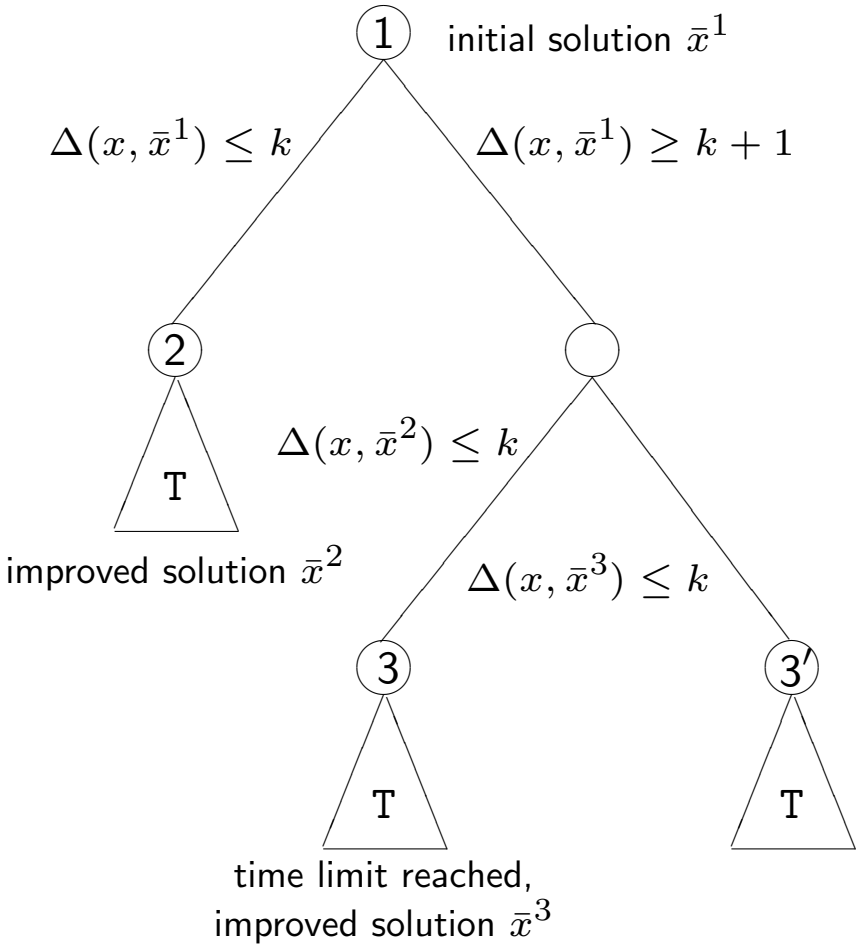
Working with a node time limit: case (a)



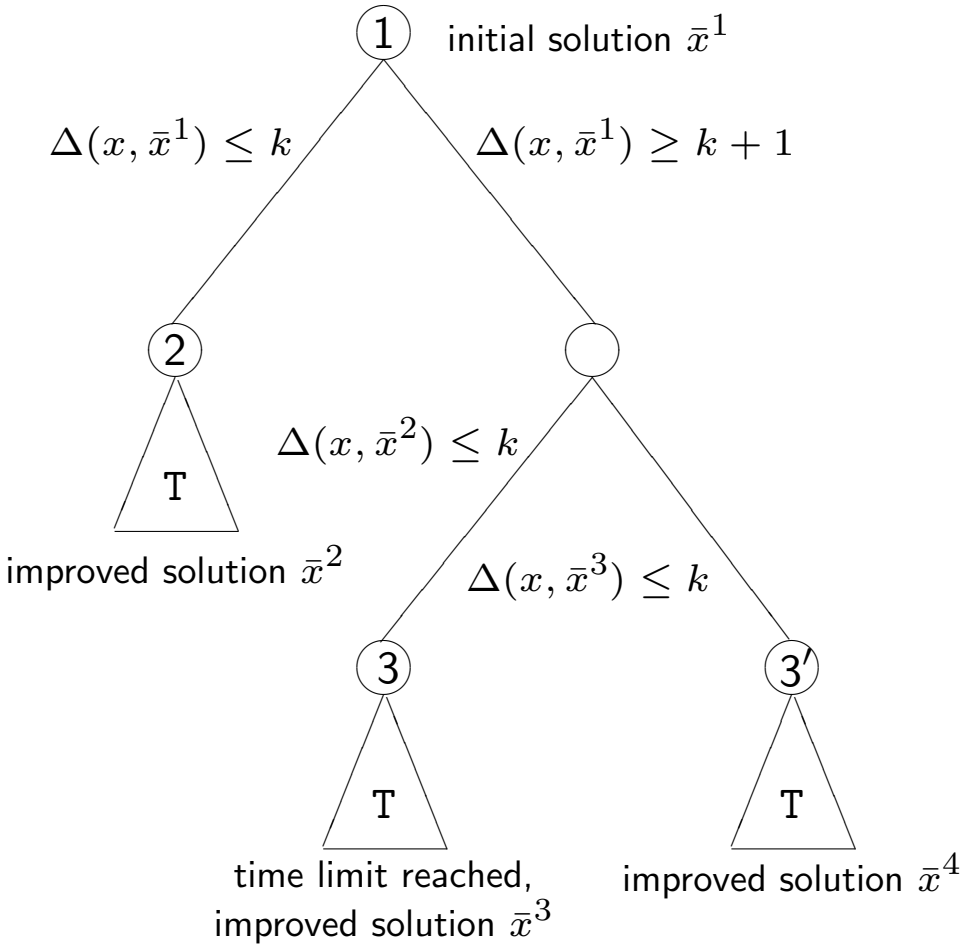
Working with a node time limit: case (a)



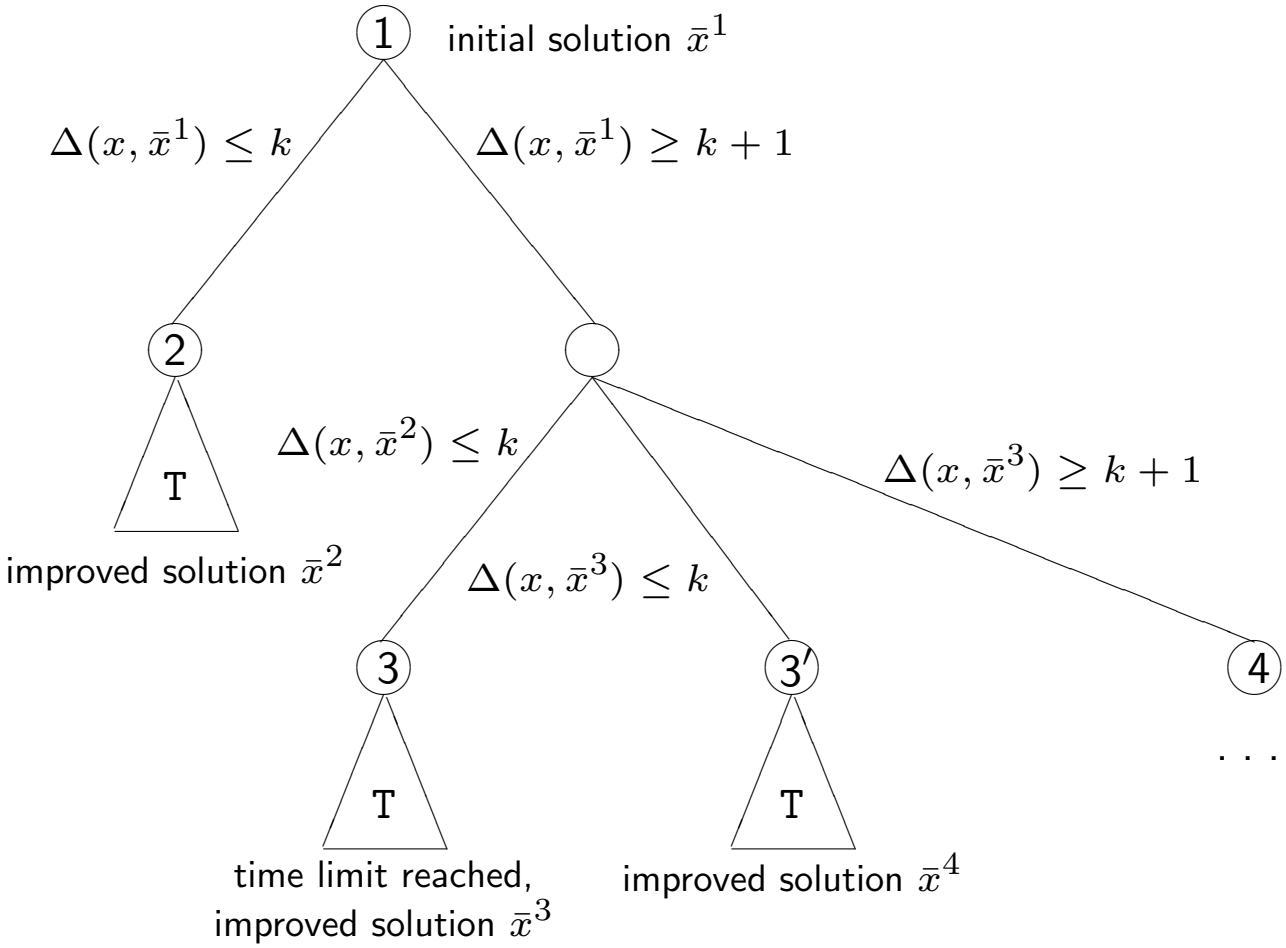
Working with a node time limit: case (a)



Working with a node time limit: case (a)



Working with a node time limit: case (a)

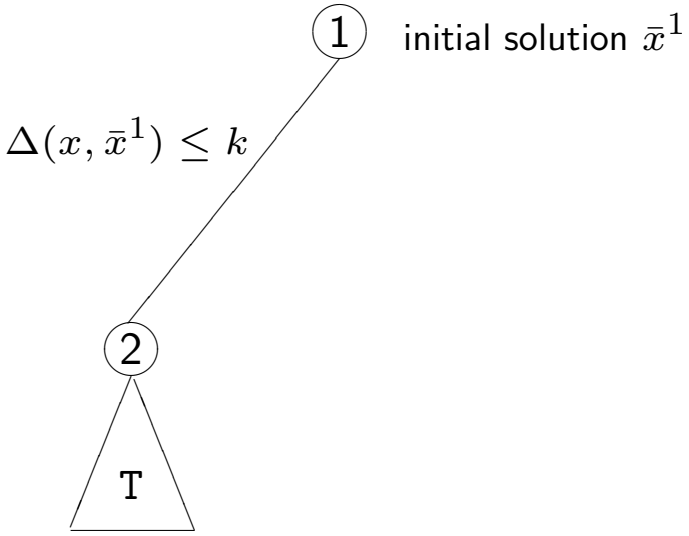


go back

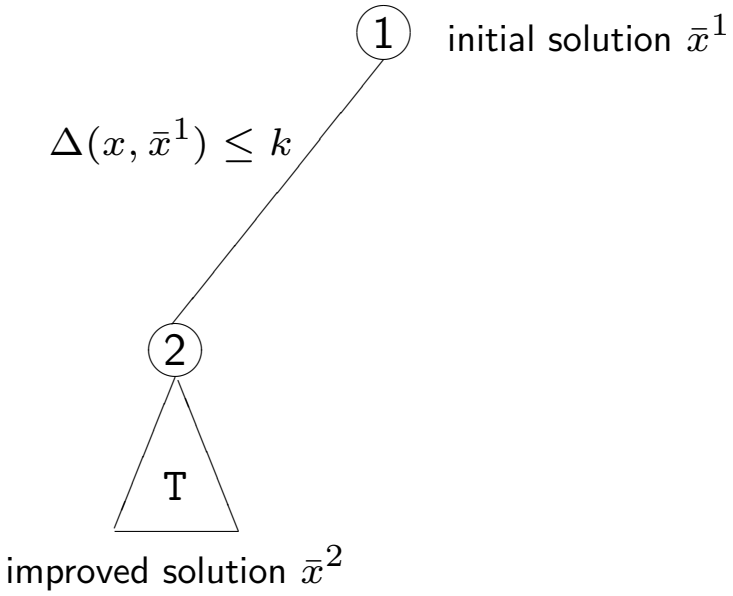
Working with a node time limit: case (b)

① initial solution $\bar{x}^1$

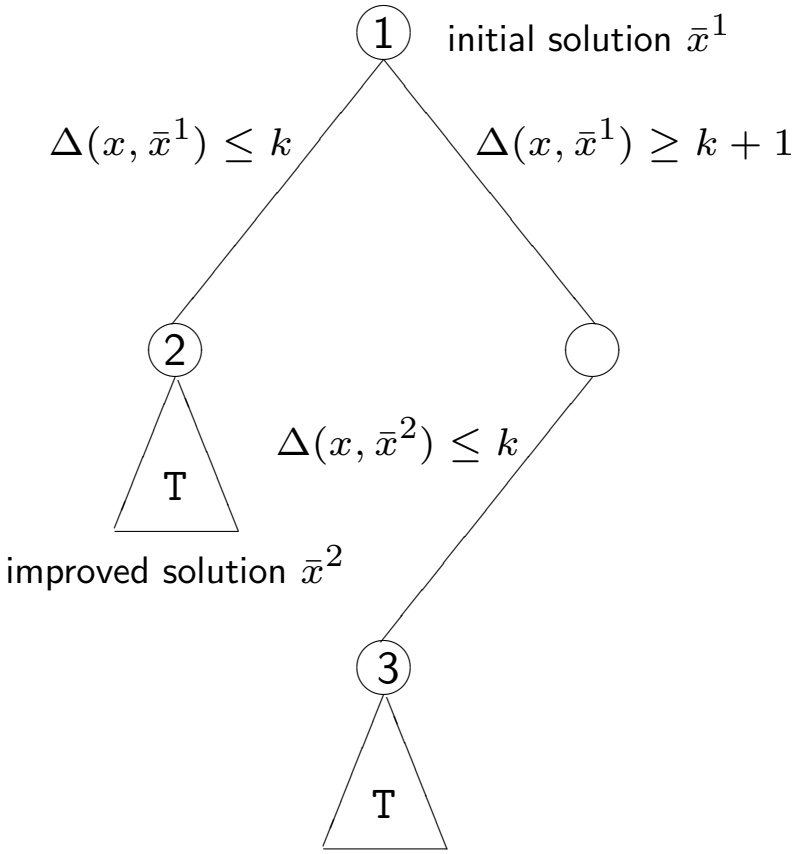
Working with a node time limit: case (b)



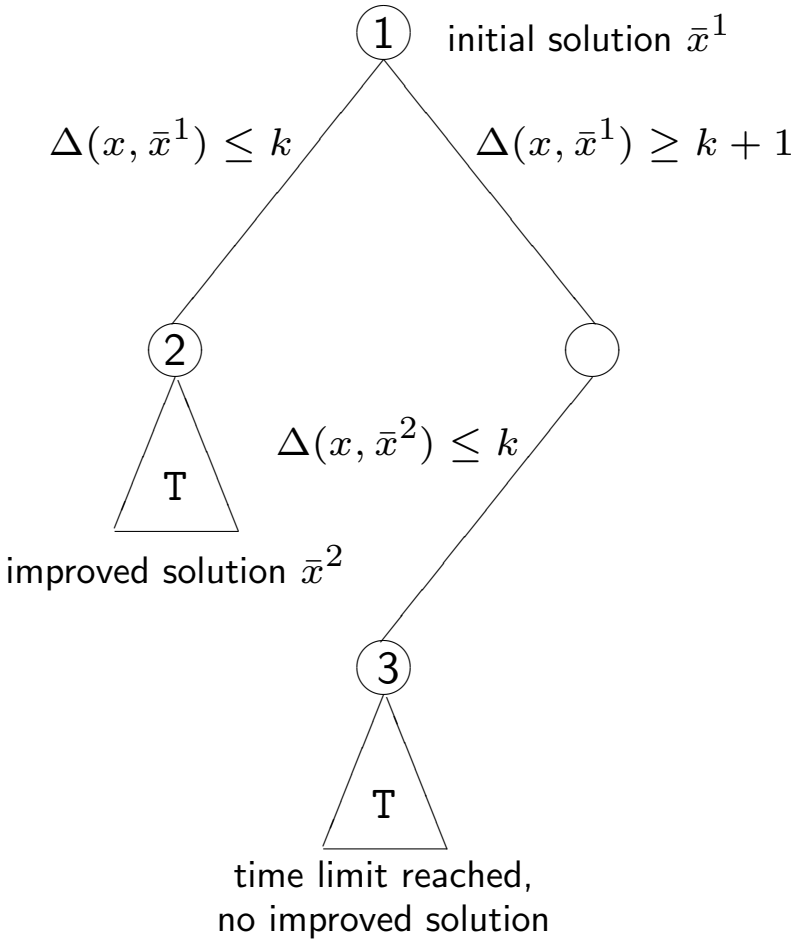
Working with a node time limit: case (b)



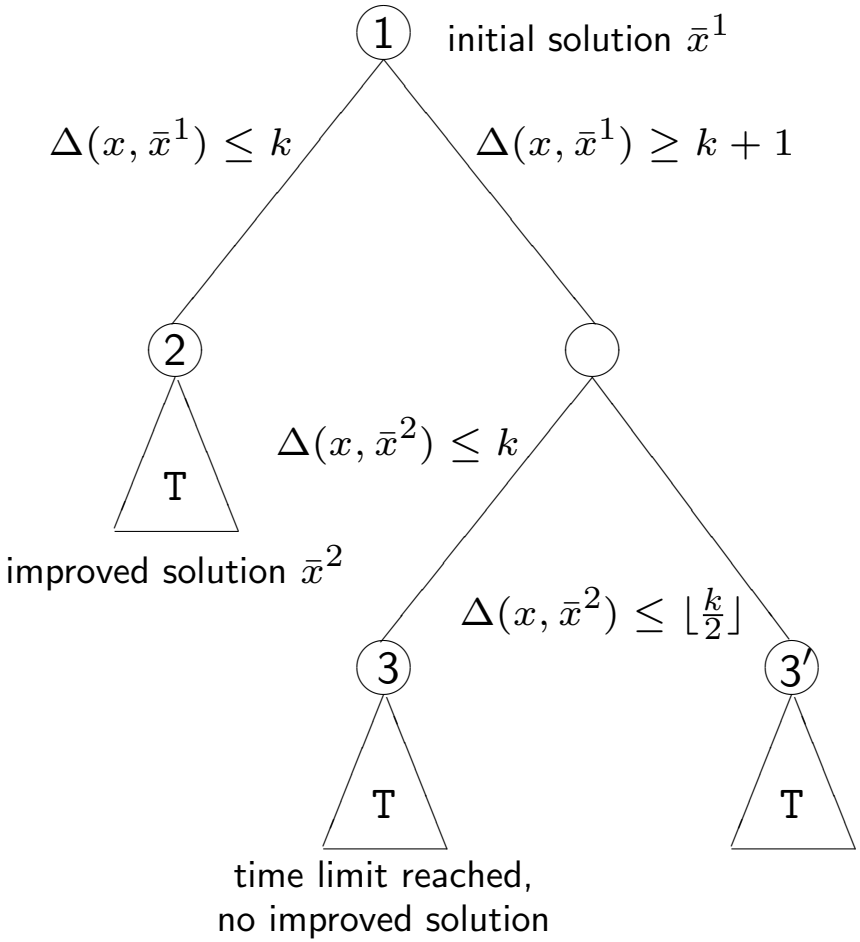
Working with a node time limit: case (b)



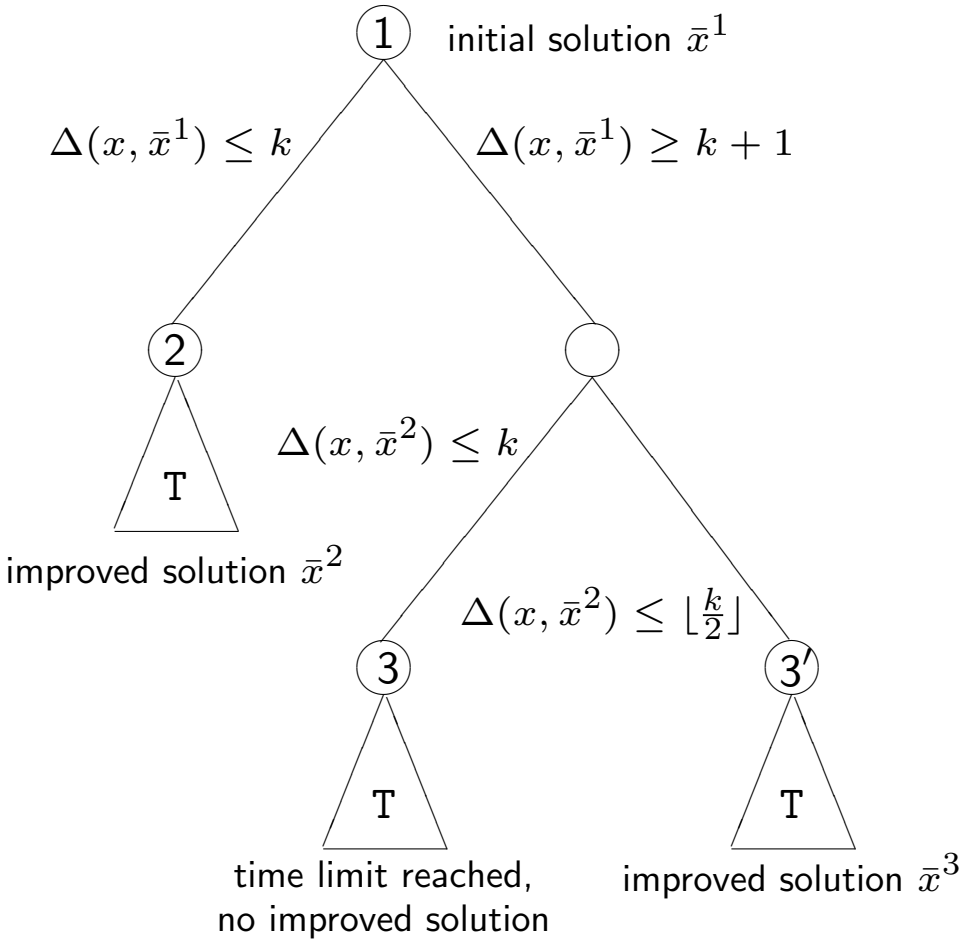
Working with a node time limit: case (b)



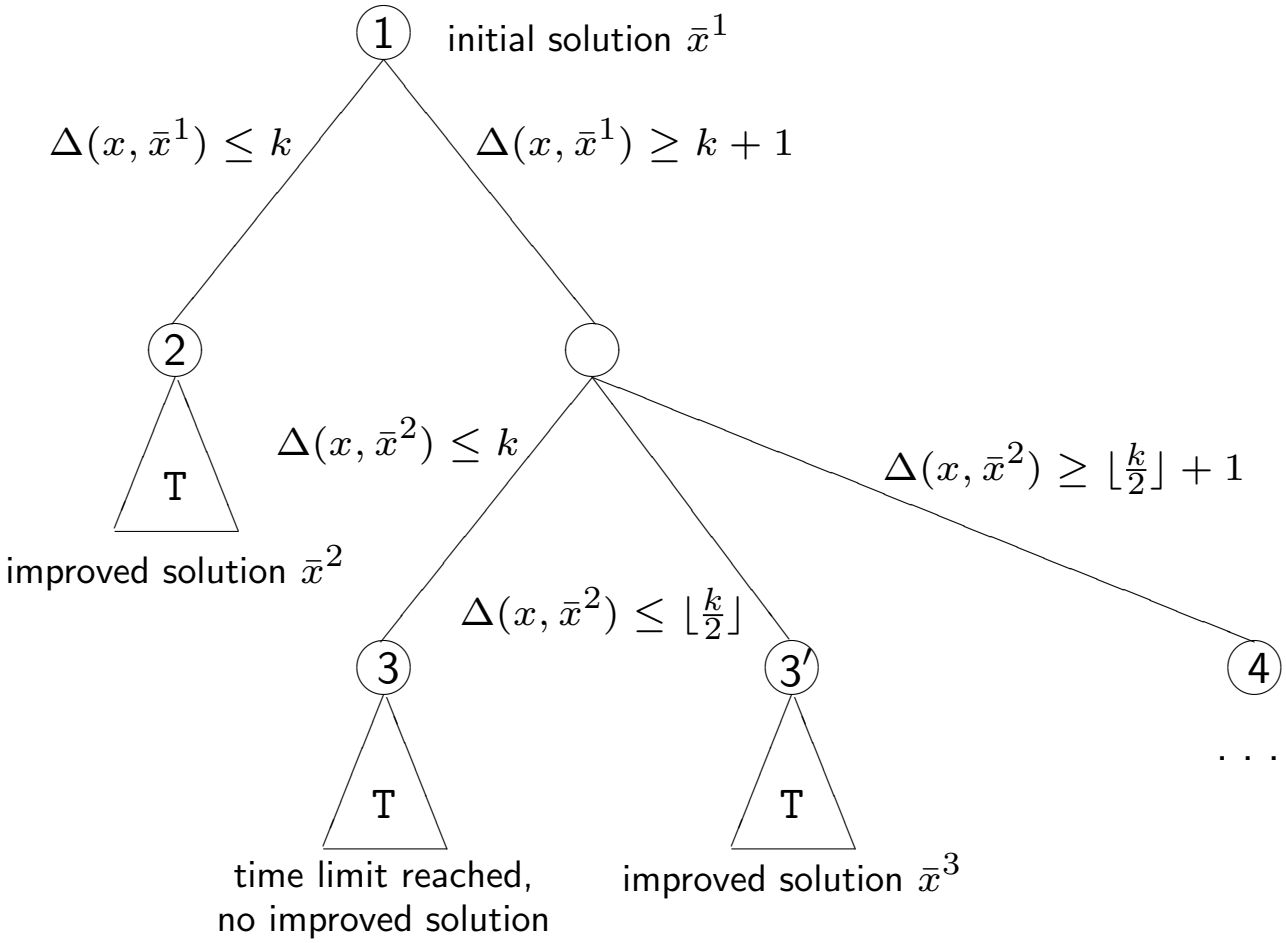
Working with a node time limit: case (b)



Working with a node time limit: case (b)



Working with a node time limit: case (b)



LB: the heuristic version, diversification

- The second improvement from a heuristic viewpoint is:
  - Diversification

LB: the heuristic version, diversification

- The second improvement from a heuristic viewpoint is:
  - Diversification

    Two mechanisms are applied in the spirit of metaheuristic techniques.

LB: the heuristic version, diversification

- The second improvement from a heuristic viewpoint is:
  - Diversification

    Two mechanisms are applied in the spirit of metaheuristic techniques.
    * **Soft** diversification:

      We first apply a "soft" action consisting in *enlarging* the current neighborhood by increasing its size by, e.g., $\lceil k/2 \rceil$.

      A new "left-branch" is then explored and in case no improved solution is found even in the enlarged neighborhood (within the time limit), we apply a stronger action in the spirit of *Variable Neighborhood Search*. [Mladenovíc & Hansen, 1997]

LB: the heuristic version, diversification

- The second improvement from a heuristic viewpoint is:
  - Diversification

    Two mechanisms are applied in the spirit of metaheuristic techniques.

    * **Soft** diversification:

      We first apply a "soft" action consisting in *enlarging* the current neighborhood by increasing its size by, e.g., $\lceil k/2 \rceil$.

      A new "left-branch" is then explored and in case no improved solution is found even in the enlarged neighborhood (within the time limit), we apply a stronger action in the spirit of *Variable Neighborhood Search*. [Mladenovíc & Hansen, 1997]

    * **Strong** diversification:

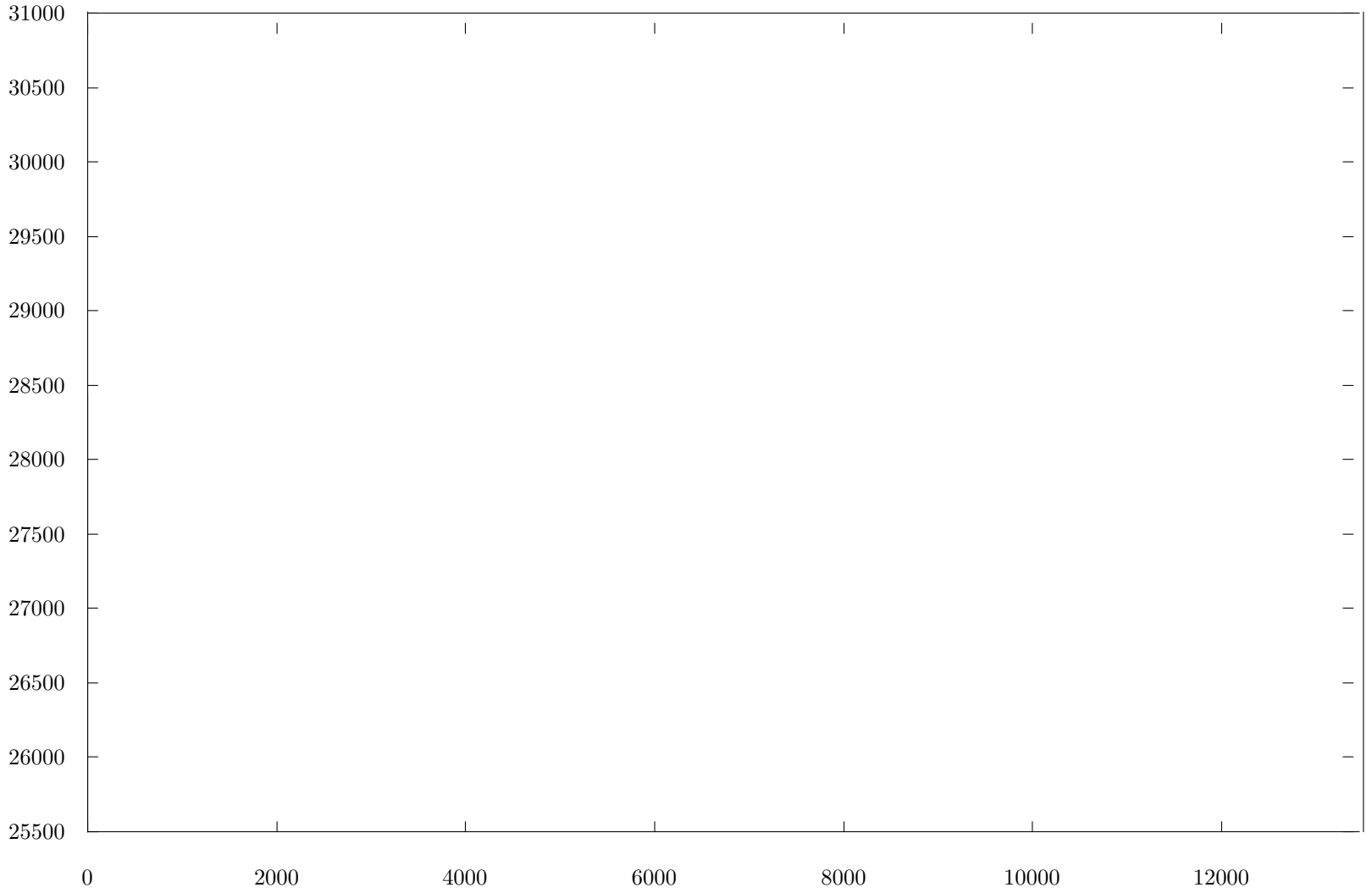
      We look for a solution (typically worse than the incumbent one) which is not "too far" from the current reference solution.

      We apply tactical branching to the current problem amended by $\Delta(x, \bar{x}^i) \leq k + 2\lceil k/2 \rceil$, but without imposing any upper bound on the optimal solution value.

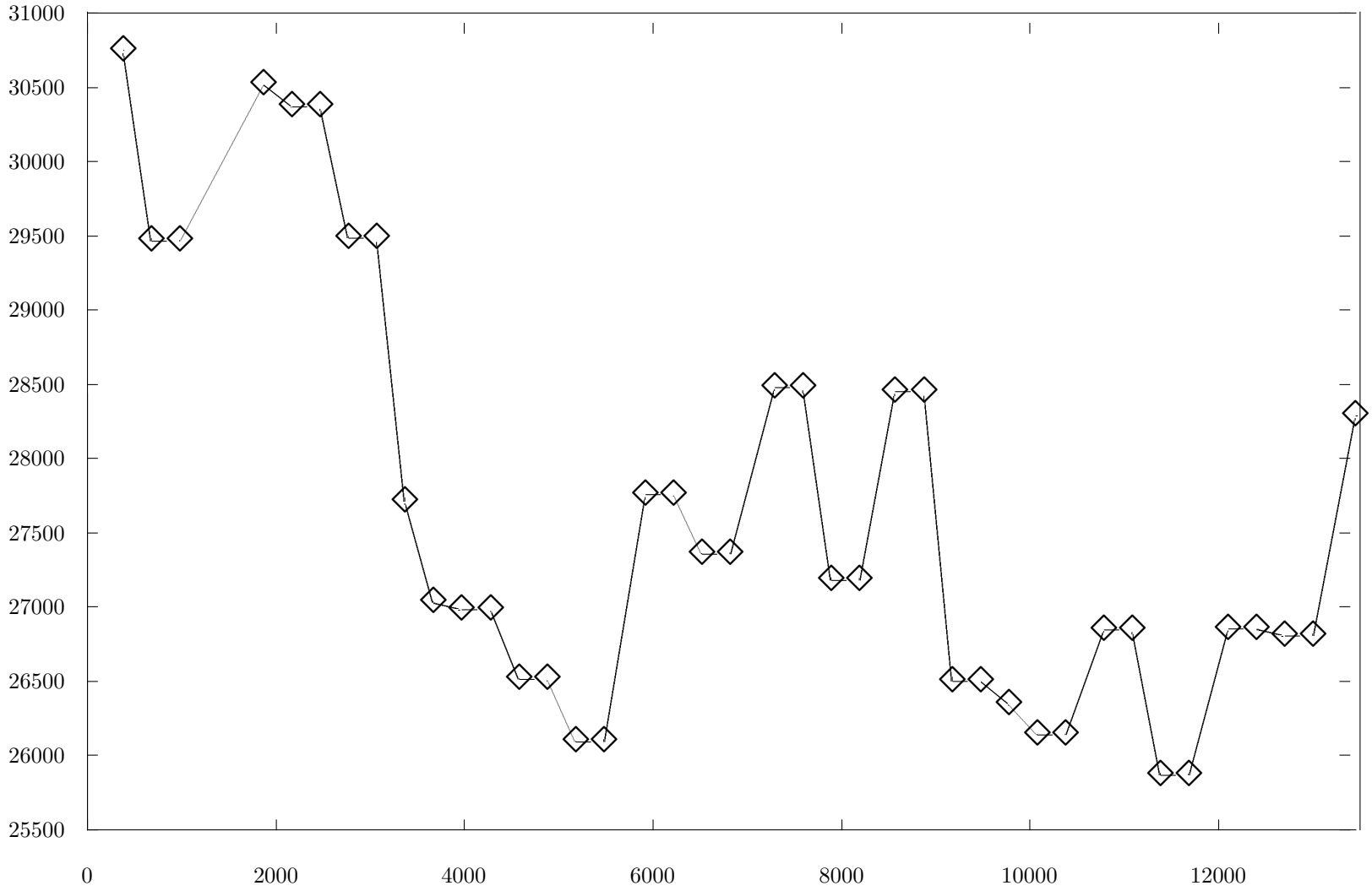
      The exploration is aborted as soon as the first feasible solution is found.

      This solution (typically worse than the current best one) is used as the new reference one.

LB: solution value vs. CPU seconds for instance B1C1S1



LB: solution value vs. CPU seconds for instance B1C1S1



LB: dealing with general integer variables

- Local branching is based on the assumption that $\mathcal{B} \neq \emptyset$, i.e., there is a set of binary variables, and moreover, this set is highly important.
- According to the computational experience, this is true even in case of MIPs involving general integer variables, in that the 0-1 variables (which are often associated with big-M terms) are often largely responsible for the difficulty of the model.

LB: dealing with general integer variables

- Local branching is based on the assumption that $\mathcal{B} \neq \emptyset$, i.e., there is a set of binary variables, and moreover, this set is highly important.
- According to the computational experience, this is true even in case of MIPs involving general integer variables, in that the 0-1 variables (which are often associated with big-M terms) are often largely responsible for the difficulty of the model.

- However, in the general case of integer variables $x_j \mid l_j \leq \bar{x}_j \leq u_j$, the local branching constraint can be written as:

$$\Delta_1(x, \bar{x}) := \sum_{j \in \mathcal{I}: \bar{x}_j = l_j} \mu_j (x_j - l_j) + \sum_{j \in \mathcal{I}: \bar{x}_j = u_j} \mu_j (u_j - x_j) + \sum_{j \in \mathcal{I}: l_j < \bar{x}_j < u_j} \mu_j (x_j^+ + x_j^-) \leq k$$

where weights μ_j are defined, e.g., as $\mu_j = 1/(u_j - l_j)\ \forall\ j \in \mathcal{I}$, while the variation terms x_j^+ and x_j^- require the introduction into the MIP model of additional constraints of the form:

$$x_j = \bar{x}_j + x_j^+ - x_j^-, \quad x_j^+ \geq 0,\ x_j^- \geq 0, \quad \forall j \in \mathcal{I} : l_j < \bar{x}_j < u_j.$$

LB: implementation within a MIP solver

- The (final) scheme above enhances (or at least enhanced back in 2002) the heuristic behavior of the MIP solver at the price of giving up optimality.

- On the other hand, it is easy to see that an alternative implementation would be within the branch-and-cut tree of a MIP solver, in the line of a classical local search improvement procedure.

LB: implementation within a MIP solver

- The (final) scheme above enhances (or at least enhanced back in 2002) the heuristic behavior of the MIP solver at the price of giving up optimality.

- On the other hand, it is easy to see that an alternative implementation would be within the branch-and-cut tree of a MIP solver, in the line of a classical local search improvement procedure.

  More precisely, whenever a new feasible solution has been found, an alternative MIP amended by the LB constraint (*MIPping*) is searched by branch-and-cut for a fixed number of nodes in the aim of improving such a solution.

LB: implementation within a MIP solver

- The (final) scheme above enhances (or at least enhanced back in 2002) the heuristic behavior of the MIP solver at the price of giving up optimality.

- On the other hand, it is easy to see that an alternative implementation would be within the branch-and-cut tree of a MIP solver, in the line of a classical local search improvement procedure.

  More precisely, whenever a new feasible solution has been found, an alternative MIP amended by the LB constraint (*MIPping*) is searched by branch-and-cut for a fixed number of nodes in the aim of improving such a solution.

- This is indeed the way in which it is implemented in CPLEX since version 9.

Relaxation Induced Neighborhood Search (RINS)

- In the spirit of solving an associated MIP to improve the incumbent, an enhancement has been proposed which takes simultaneously into account both the incumbent solution, $\bar{x}$, and the the solution of the continuous relaxation, say x^*, at a given node of the branch-and-bound tree.

  [Danna, Rothberg & Le Pape, *MPA* 2005]

- Then, $\bar{x}$ and x^* are compared and all the binary variables which assume the same value are hard-fixed in an associated MIP.

- This associated MIP is then solved by using the MIP solver as a black-box, and in case the incumbent solution is improved, $\bar{x}$ is updated in the rest of the tree.

Relaxation Induced Neighborhood Search (RINS)

- In the spirit of solving an associated MIP to improve the incumbent, an enhancement has been proposed which takes simultaneously into account both the incumbent solution, $\bar{x}$, and the the solution of the continuous relaxation, say x^*, at a given node of the branch-and-bound tree.

  [Danna, Rothberg & Le Pape, *MPA* 2005]

- Then, $\bar{x}$ and x^* are compared and all the binary variables which assume the same value are hard-fixed in an associated MIP.
- This associated MIP is then solved by using the MIP solver as a black-box, and in case the incumbent solution is improved, $\bar{x}$ is updated in the rest of the tree.
- The advantages of RINS with respect to LB are essentially two:

Relaxation Induced Neighborhood Search (RINS)

- In the spirit of solving an associated MIP to improve the incumbent, an enhancement has been proposed which takes simultaneously into account both the incumbent solution, $\bar{x}$, and the the solution of the continuous relaxation, say x^*, at a given node of the branch-and-bound tree.

  [Danna, Rothberg & Le Pape, *MPA* 2005]

- Then, $\bar{x}$ and x^* are compared and all the binary variables which assume the same value are hard-fixed in an associated MIP.

- This associated MIP is then solved by using the MIP solver as a black-box, and in case the incumbent solution is improved, $\bar{x}$ is updated in the rest of the tree.

- The advantages of RINS with respect to LB are essentially two:
  1. On the one side, because the solution of continuous relaxation x^* changes at each branch-and-bound node, RINScan be invoked more often, potentially every node, thus leading, in principle, to faster improvements.

Relaxation Induced Neighborhood Search (RINS)

- In the spirit of solving an associated MIP to improve the incumbent, an enhancement has been proposed which takes simultaneously into account both the incumbent solution, $\bar{x}$, and the the solution of the continuous relaxation, say x^*, at a given node of the branch-and-bound tree.

  [Danna, Rothberg & Le Pape, *MPA* 2005]

- Then, $\bar{x}$ and x^* are compared and all the binary variables which assume the same value are hard-fixed in an associated MIP.

- This associated MIP is then solved by using the MIP solver as a black-box, and in case the incumbent solution is improved, $\bar{x}$ is updated in the rest of the tree.

- The advantages of RINS with respect to LB are essentially two:
  1. On the one side, because the solution of continuous relaxation x^* changes at each branch-and-bound node, RINScan be invoked more often, potentially every node, thus leading, in principle, to faster improvements.
  2. Moreover, the sub-MIPs explored by RINS are generally smaller than those of LB because the former fixes variables (thus removing them from the MIP) while the latter imposes a soft fixing based on the addition of a linear inequality.

Heuristics for Feasibility and Optimality in Mixed Integer Programming

John Chinneck

Carleton University, Canada
chinneck@sce.carleton.ca

Andrea Lodi

University of Bologna, Italy
andrea.lodi@unibo.it

CIRRELT Spring School on Logistics, Montréal, May 17th, 2010

Outline

1. Introduction and Orientation (Lodi)

Part I: Achieving Integer-Feasibility Quickly

2. Classic Feasibility-Seeking Algorithms (Chinneck)
3. Active Constraint Variable Selection (Chinneck)
4. Branching to Force Change (Chinneck)
5. The Feasibility Pump (Lodi)

Part II: Reaching (quasi-)Optimality Quickly

6. New Node Selection Rules (Chinneck)
7. Local Branching and RINS (Lodi)

Part III: Analyzing Infeasible MIPs

8. Isolating Infeasible Subsystems (Chinneck)
9. Repairing MIP Infeasibility via Local Branching (Lodi)
10. Conclusions (Chinneck)

Working with infeasible solutions

- Sometimes waiting to have a fully feasible solution before starting a local search approach, for example, within a branch-and-bound tree, is unnecessary.

Working with infeasible solutions

- Sometimes waiting to have a fully feasible solution before starting a local search approach, for example, within a branch-and-bound tree, is unnecessary.

- In addition, during the exploration of the enumeration tree, one can collect solutions which, although "slightly" infeasible, are potentially of good quality.

- These solutions are (generally) simply discarded while a careful use of local search could lead to repair them and driving to feasibility.

- One flexible way of doing this is combining FP and LB. [Fischetti & L., *C&OR* 2008]

Combining FP and LB

- FP is executed for a limited number of iterations and the integer (infeasible) solution $\widetilde{x}$ with minimum distance Δ to a feasible solution x^* of the continuous relaxation of P is stored;

Combining FP and LB

- FP is executed for a limited number of iterations and the integer (infeasible) solution $\widetilde{x}$ with minimum distance Δ to a feasible solution x^* of the continuous relaxation of P is stored;

- Besides the value of $\Delta(x, \widetilde{x})$, the infeasibility of $\widetilde{x}$ can be combinatorially measured in terms of the number of constraints of the original MIP which violates.

- Let us call T set of the indices of the violated constraints.

Combining FP and LB

- FP is executed for a limited number of iterations and the integer (infeasible) solution $\widetilde{x}$ with minimum distance Δ to a feasible solution x^* of the continuous relaxation of P is stored;

- Besides the value of $\Delta(x, \widetilde{x})$, the infeasibility of $\widetilde{x}$ can be combinatorially measured in terms of the number of constraints of the original MIP which violates.

- Let us call T set of the indices of the violated constraints.

- Clearly, in the spirit of phase 1 of the Simplex algorithm, one can modify the original MIP to make such a $\widetilde{x}$ feasible by:

Combining FP and LB

- FP is executed for a limited number of iterations and the integer (infeasible) solution $\widetilde{x}$ with minimum distance Δ to a feasible solution x^* of the continuous relaxation of P is stored;

- Besides the value of $\Delta(x, \widetilde{x})$, the infeasibility of $\widetilde{x}$ can be combinatorially measured in terms of the number of constraints of the original MIP which violates.

- Let us call T set of the indices of the violated constraints.

- Clearly, in the spirit of phase 1 of the Simplex algorithm, one can modify the original MIP to make such a $\widetilde{x}$ feasible by:

  1. adding a continuous non-negative artificial variable s_i to each $i \in T$

Combining FP and LB

- FP is executed for a limited number of iterations and the integer (infeasible) solution $\widetilde{x}$ with minimum distance Δ to a feasible solution x^* of the continuous relaxation of P is stored;
- Besides the value of $\Delta(x, \widetilde{x})$, the infeasibility of $\widetilde{x}$ can be combinatorially measured in terms of the number of constraints of the original MIP which violates.
- Let us call T set of the indices of the violated constraints.
- Clearly, in the spirit of phase 1 of the Simplex algorithm, one can modify the original MIP to make such a $\widetilde{x}$ feasible by:
  1. adding a continuous non-negative artificial variable s_i to each $i \in T$
  2. adding the constraint $s_i \leq M y_i$ with $y_i \in \{01, 1\}$.
- The artificial variable s_i is used to satisfy the constraint but if and only if the corresponding y_i variable is set to 1 (M is a classical big-M coefficient).

Combining FP and LB(cont.d)

- LB starts by using $\widetilde{x}$ as a reference solution and replacing the original objective function with

$$\min \sum_{i \in T} y_i$$

- The solution $\widetilde{x}$ is clearly feasible for such a new MIP and its objective function value is $|T|$.

Combining FP and LB(cont.d)

- LB starts by using $\widetilde{x}$ as a reference solution and replacing the original objective function with

$$\min \sum_{i \in T} y_i$$

- The solution $\widetilde{x}$ is clearly feasible for such a new MIP and its objective function value is $|T|$.

- Hence, in a first phase, LB attempts to improve the current infeasible solution by reducing the number of infeasible constraints in the spirit of the first phase of the simplex algorithm.

Combining FP and LB(cont.d)

- LB starts by using $\widetilde{x}$ as a reference solution and replacing the original objective function with

$$\min \sum_{i \in T} y_i$$

- The solution $\widetilde{x}$ is clearly feasible for such a new MIP and its objective function value is $|T|$.

- Hence, in a first phase, LB attempts to improve the current infeasible solution by reducing the number of infeasible constraints in the spirit of the first phase of the simplex algorithm.

- In the second phase, once a feasible solution has been found, the original objective function is then restored and LB takes care of improving the quality of such a solution.