# Local Branching: A Tutorial

Andrea Lodi
*University of Bologna, Italy*
alodi@deis.unibo.it

Joint work with Matteo Fischetti

Instances, codes, papers and slides at:
http://www.or.deis.unibo.it/research_pages/ORinstances/MIPs.html

MIC 2003, Kyoto, August 25-28

# Motivation

- Mixed-integer linear programming plays a central role in modeling difficult-to-solve (NP-hard) combinatorial problems.

# Motivation

- Mixed-integer linear programming plays a central role in modeling difficult-to-solve (NP-hard) combinatorial problems.
- However, the exact solution of the resulting models often cannot be carried out for the problem sizes of interest in real-world applications, hence one is interested in effective heuristic methods.

# Motivation

- Mixed-integer linear programming plays a central role in modeling difficult-to-solve (NP-hard) combinatorial problems.
- However, the exact solution of the resulting models often cannot be carried out for the problem sizes of interest in real-world applications, hence one is interested in effective heuristic methods.
- Current MIP solvers nowadays incorporate most of the theoretical and practical results in the field of Integer Programming:
  general-purpose cutting planes, preprocessing, branching mechanisms, pricing methods, . . .

# Motivation

- Mixed-integer linear programming plays a central role in modeling difficult-to-solve (NP-hard) combinatorial problems.
- However, the exact solution of the resulting models often cannot be carried out for the problem sizes of interest in real-world applications, hence one is interested in effective heuristic methods.
- Current MIP solvers nowadays incorporate most of the theoretical and practical results in the field of Integer Programming:
  general-purpose cutting planes, preprocessing, branching mechanisms, pricing methods, . . .
- MIP solvers used without proof of optimality are sometimes among the best heuristics.

# Motivation

- Mixed-integer linear programming plays a central role in modeling difficult-to-solve (NP-hard) combinatorial problems.
- However, the exact solution of the resulting models often cannot be carried out for the problem sizes of interest in real-world applications, hence one is interested in effective heuristic methods.
- Current MIP solvers nowadays incorporate most of the theoretical and practical results in the field of Integer Programming:
  general-purpose cutting planes, preprocessing, branching mechanisms, pricing methods, . . .
- MIP solvers used without proof of optimality are sometimes among the best heuristics.
- The same holds for truncated special-purpose branch-and-bound methods for problems with an exponential number of constraints, e.g., *Asymmetric Traveling Salesman Problem* [Zhang, 2002]

# Motivation

- Mixed-integer linear programming plays a central role in modeling difficult-to-solve (NP-hard) combinatorial problems.
- However, the exact solution of the resulting models often cannot be carried out for the problem sizes of interest in real-world applications, hence one is interested in effective heuristic methods.
- Current MIP solvers nowadays incorporate most of the theoretical and practical results in the field of Integer Programming:
  general-purpose cutting planes, preprocessing, branching mechanisms, pricing methods, . . .
- MIP solvers used without proof of optimality are sometimes among the best heuristics.
- The same holds for truncated special-purpose branch-and-bound methods for problems with an exponential number of constraints, e.g., *Asymmetric Traveling Salesman Problem* [Zhang, 2002]

AIM: integrating local search and metaheuristic ideas within Mixed Integer Programming

# Integrating Local Search and MIP

Three main questions have to be answered:

## Integrating Local Search and MIP

Three main questions have to be answered:

# 1. How to define a neighborhood?

## Integrating Local Search and MIP

Three main questions have to be answered:

1. How to define a neighborhood?

2. How to search the neighborhood?

## Integrating Local Search and MIP

Three main questions have to be answered:

1. How to define a neighborhood?

2. How to search the neighborhood?

3. How to perform diversification?

# Defining a neighborhood

- A commonly used heuristic idea in MIP context is the so-called hard *variable fixing* or *diving*:

# Defining a neighborhood

- A commonly used heuristic idea in MIP context is the so-called hard *variable fixing* or *diving*:
  1. the solution of a continuous relaxation $x^*$ is "analyzed";

# Defining a neighborhood

- A commonly used heuristic idea in MIP context is the so-called hard *variable fixing* or *diving*:
  1. the solution of a continuous relaxation $x^*$ is "analyzed";
  2. some of its nonzero variables are heuristically rounded-up to the nearest integer (if non-integer) and then fixed to this value;

# Defining a neighborhood

- A commonly used heuristic idea in MIP context is the so-called hard *variable fixing* or *diving*:
  1. the solution of a continuous relaxation $x^*$ is "analyzed";
  2. some of its nonzero variables are heuristically rounded-up to the nearest integer (if non-integer) and then fixed to this value;
  3. the method is iterated until either a feasible solution is found or the problem is infeasible.

# Defining a neighborhood

- A commonly used heuristic idea in MIP context is the so-called hard *variable fixing* or *diving*:
  1. the solution of a continuous relaxation $x^*$ is "analyzed";
  2. some of its nonzero variables are heuristically rounded-up to the nearest integer (if non-integer) and then fixed to this value;
  3. the method is iterated until either a feasible solution is found or the problem is infeasible.
- The obvious question related to this mechanism is however:

# Defining a neighborhood

- A commonly used heuristic idea in MIP context is the so-called hard *variable fixing* or *diving*:
  1. the solution of a continuous relaxation $x^*$ is "analyzed";
  2. some of its nonzero variables are heuristically rounded-up to the nearest integer (if non-integer) and then fixed to this value;
  3. the method is iterated until either a feasible solution is found or the problem is infeasible.
- The obvious question related to this mechanism is however:

How should one choose the actual variables to be fixed?

# Defining a neighborhood

- A commonly used heuristic idea in MIP context is the so-called hard *variable fixing* or *diving*:
  1. the solution of a continuous relaxation $x^*$ is "analyzed";
  2. some of its nonzero variables are heuristically rounded-up to the nearest integer (if non-integer) and then fixed to this value;
  3. the method is iterated until either a feasible solution is found or the problem is infeasible.
- The obvious question related to this mechanism is however:

  How should one choose the actual variables to be fixed?

- The idea is simple. In a binary problem in which a current feasible solution $\bar{x}$ is given, impose a soft *variable fixing* constraint, fixing a relevant number of variables without losing the possibility of finding good feasible solutions:

# Defining a neighborhood

- A commonly used heuristic idea in MIP context is the so-called hard *variable fixing* or *diving*:
  1. the solution of a continuous relaxation $x^*$ is "analyzed";
  2. some of its nonzero variables are heuristically rounded-up to the nearest integer (if non-integer) and then fixed to this value;
  3. the method is iterated until either a feasible solution is found or the problem is infeasible.
- The obvious question related to this mechanism is however:

  How should one choose the actual variables to be fixed?

- The idea is simple. In a binary problem in which a current feasible solution $\bar{x}$ is given, impose a soft *variable fixing* constraint, fixing a relevant number of variables without losing the possibility of finding good feasible solutions:

$$\sum_{j=1}^{n} \bar{x}_j \, x_j \geq \lceil 0.9 \sum_{j=1}^{n} \bar{x}_j \rceil \tag{1}$$

# Defining a neighborhood

- A commonly used heuristic idea in MIP context is the so-called hard *variable fixing* or *diving*:
  1. the solution of a continuous relaxation $x^*$ is "analyzed";
  2. some of its nonzero variables are heuristically rounded-up to the nearest integer (if non-integer) and then fixed to this value;
  3. the method is iterated until either a feasible solution is found or the problem is infeasible.
- The obvious question related to this mechanism is however:

  How should one choose the actual variables to be fixed?

- The idea is simple. In a binary problem in which a current feasible solution $\bar{x}$ is given, impose a soft *variable fixing* constraint, fixing a relevant number of variables without losing the possibility of finding good feasible solutions:

$$\sum_{j=1}^{n} \bar{x}_j \, x_j \geq \lceil 0.9 \sum_{j=1}^{n} \bar{x}_j \rceil \tag{1}$$

- Constraint (1) defines a neighborhood of $\bar{x}$, and, since the constraint is linear, the neighborhood can be explored using a generic MIP solver.

# A general MIP with 0-1 variables

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

$$
\begin{aligned}
(P) \qquad & \min c^T x && && (2)\\
& Ax \geq b && && (3)\\
& x_j \in \{0,1\} && \forall j \in \mathcal{B} \neq \emptyset && (4)\\
& x_j \geq 0, \text{ integer} && \forall j \in \mathcal{G} && (5)\\
& x_j \geq 0 && \forall j \in \mathcal{C} && (6)
\end{aligned}
$$

- We consider the case in which $\mathcal{B} \neq \emptyset$ and more precisely, we assume that

> fixing the binary variables strongly simplifies the problem

- Moreover, we assume to have an initial solution, $\bar{x}$ at hand, so-called reference solution, and let $\overline{S} := \{j \in \mathcal{B} : \bar{x}_j = 1\}$ denote the binary support of $\bar{x}$.

# The local branching framework

- 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*:

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \overline{S}} x_j \leq k \tag{7}$$

  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

- 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*:

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \overline{S}} x_j \leq k \tag{7}$$

  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 (7) imposes a maximum Hamming distance of $k$ among the feasible neighbors of $\bar{x}$.

# The local branching framework

- 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*:

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \overline{S}} x_j \leq k \tag{7}$$

  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 (7) imposes a maximum Hamming distance of $k$ among the feasible neighbors of $\bar{x}$.

- When the cardinality of $\overline{S}$ of any feasible solution of $(P)$ is a constant, the local branching constraint assumes the asymmetric form:

$$\sum_{j \in \overline{S}} (1 - x_j) \leq k' \ (= k/2) \tag{8}$$

  which is the classical $k'$-OPT neighborhood for the *Symmetric Traveling Salesman Problem*.

# The local branching framework (cont.d)

- The local branching constraint can be used as a branching criterion within an enumerative scheme for $(P)$.
  Indeed, the following disjunction can be imposed:

# The local branching framework (cont.d)

- The local branching constraint can be used as a branching criterion within an enumerative scheme for $(P)$.
  Indeed, the following disjunction can be imposed:

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

# The local branching framework (cont.d)

- The local branching constraint can be used as a branching criterion within an enumerative scheme for $(P)$.
  Indeed, the following disjunction can be imposed:

$$\Delta(x, \bar{x}) \leq k \quad \text{(left branch)} \qquad \textbf{or} \qquad \Delta(x, \bar{x}) \geq k + 1 \quad \text{(right branch)} \tag{9}$$

# The local branching framework (cont.d)

- The local branching constraint can be used as a branching criterion within an enumerative scheme for $(P)$.
  Indeed, the following disjunction can be imposed:

$$\Delta(x, \bar{x}) \leq k \quad \text{(left branch)} \qquad \textbf{or} \qquad \Delta(x, \bar{x}) \geq k + 1 \quad \text{(right branch)} \tag{9}$$

- The idea is again simple: the neighborhood $\mathcal{N}(\bar{x}, k)$ corresponding to the left branch must be "sufficiently small" to be optimized within short computing time, but still "large enough" to likely contain better solutions than $\bar{x}$.

# The local branching framework (cont.d)

- The local branching constraint can be used as a branching criterion within an enumerative scheme for $(P)$.
  Indeed, the following disjunction can be imposed:

$$\Delta(x, \bar{x}) \leq k \quad \text{(left branch)} \qquad \textbf{or} \qquad \Delta(x, \bar{x}) \geq k + 1 \quad \text{(right branch)} \tag{9}$$

- The idea is again simple: the neighborhood $\mathcal{N}(\bar{x}, k)$ corresponding to the left branch must be "sufficiently small" to be optimized within short computing time, but still "large enough" to likely contain better solutions than $\bar{x}$.

- Obviously, the choice of $k$ is a problem by itself, but values of $k$ in range $[10, 20]$ proved effective in most cases.

# The local branching framework (cont.d)

- The local branching constraint can be used as a branching criterion within an enumerative scheme for $(P)$.
  Indeed, the following disjunction can be imposed:

$$\Delta(x, \bar{x}) \leq k \quad \text{(left branch)} \qquad \textbf{or} \qquad \Delta(x, \bar{x}) \geq k + 1 \quad \text{(right branch)} \tag{9}$$

- The idea is again simple: the neighborhood $\mathcal{N}(\bar{x}, k)$ corresponding to the left branch must be "sufficiently small" to be optimized within short computing time, but still "large enough" to likely contain better solutions than $\bar{x}$.

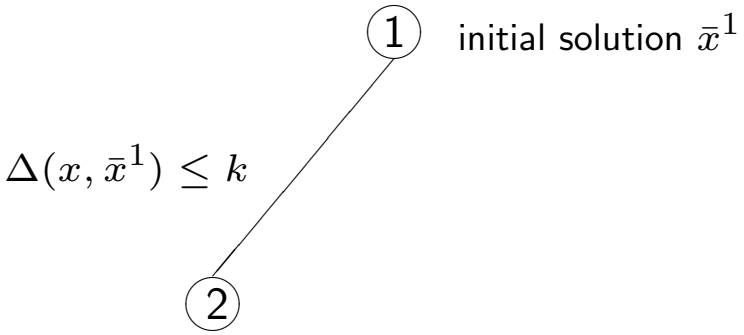
- Obviously, the choice of $k$ is a problem by itself, but values of $k$ in range $[10, 20]$ proved effective in most cases.

- The neighborhoods defined by the local branching constraints can be explored by using, as a black-box, a MIP solver, i.e., a standard tactical branching criterion such as, e.g., branching on fractional variables.

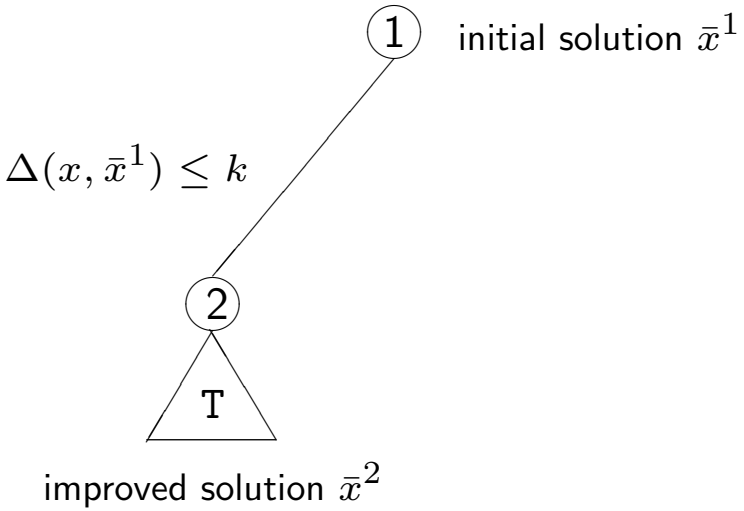
# The basic local branching scheme

① initial solution $\bar{x}^1$

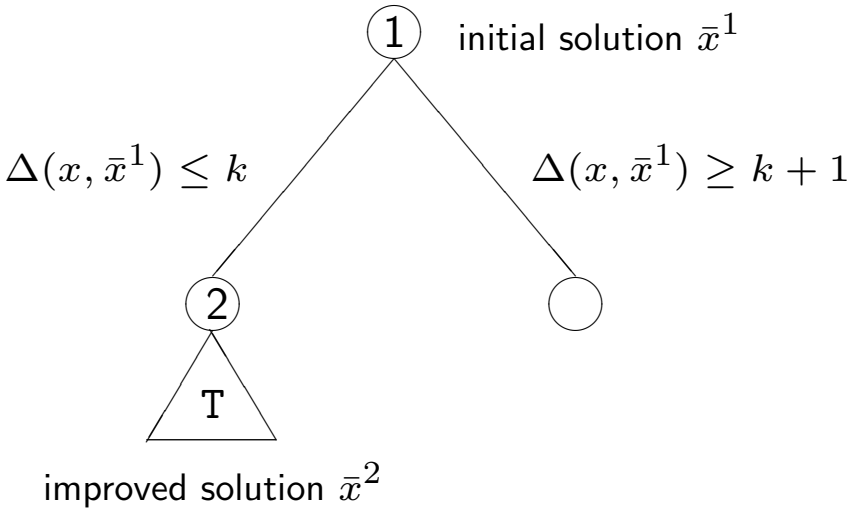
# The basic local branching scheme



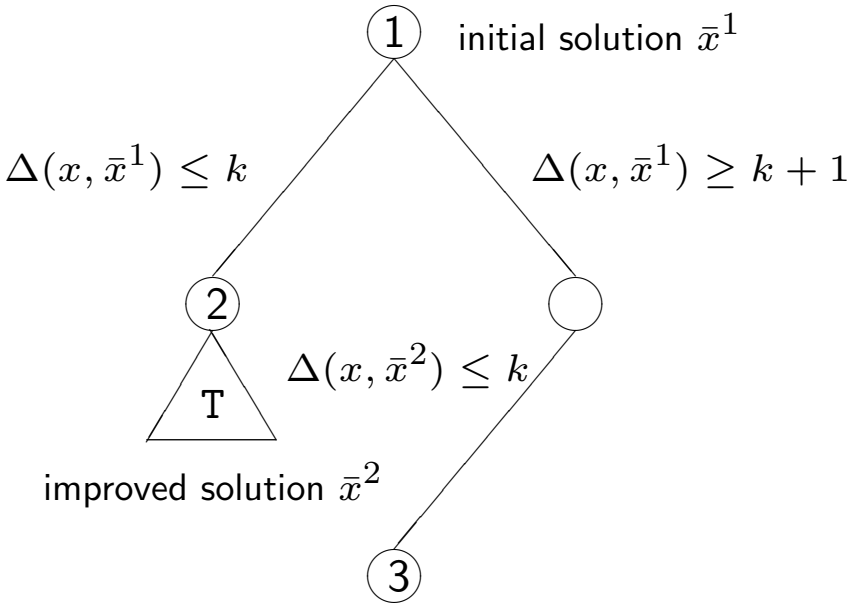
# The basic local branching scheme



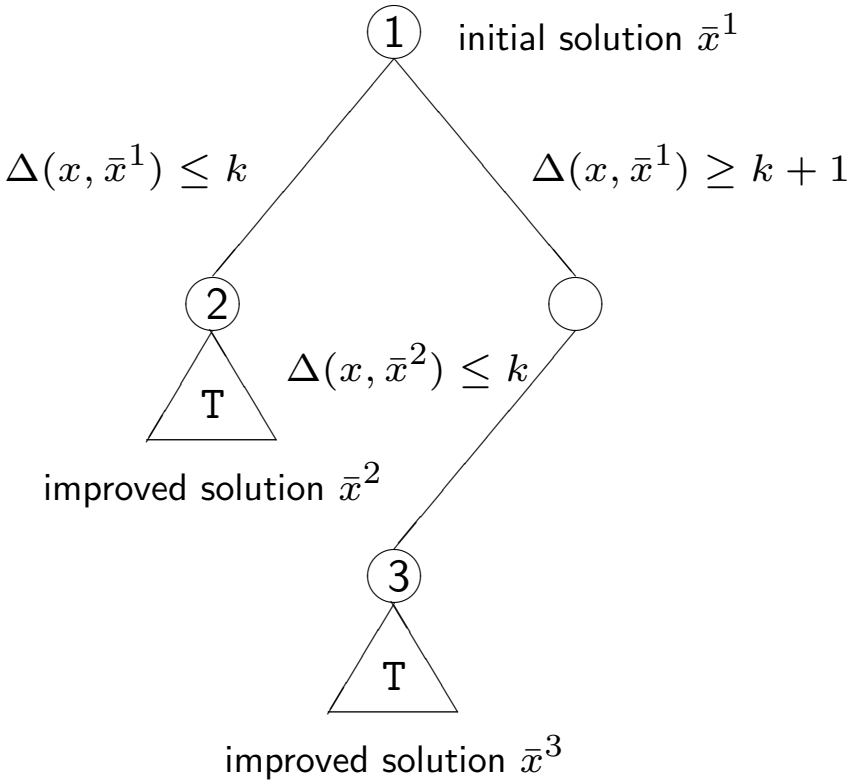
# The basic local branching scheme



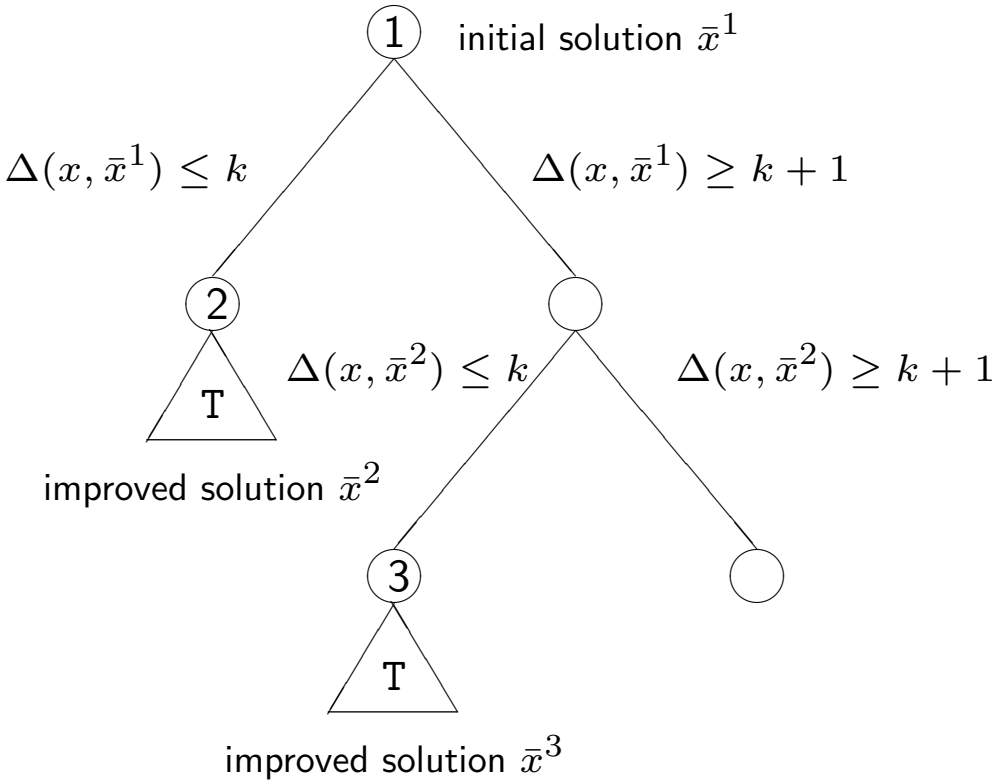
# The basic local branching scheme



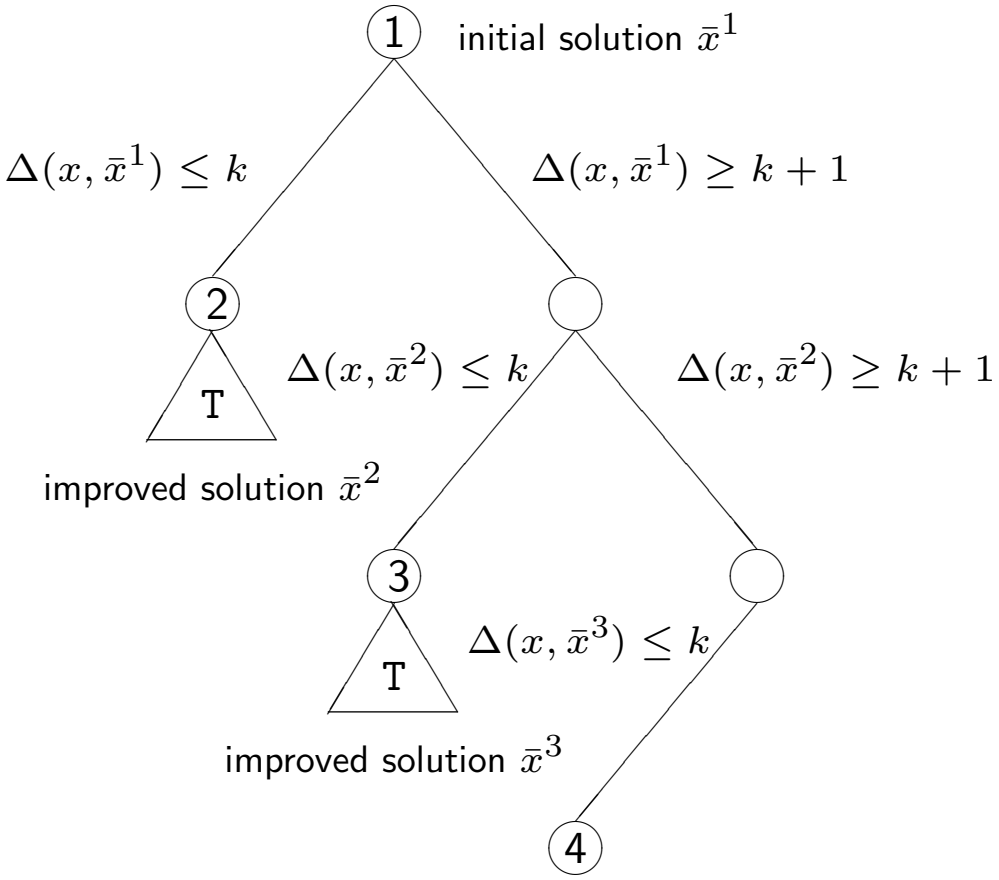
# The basic local branching scheme



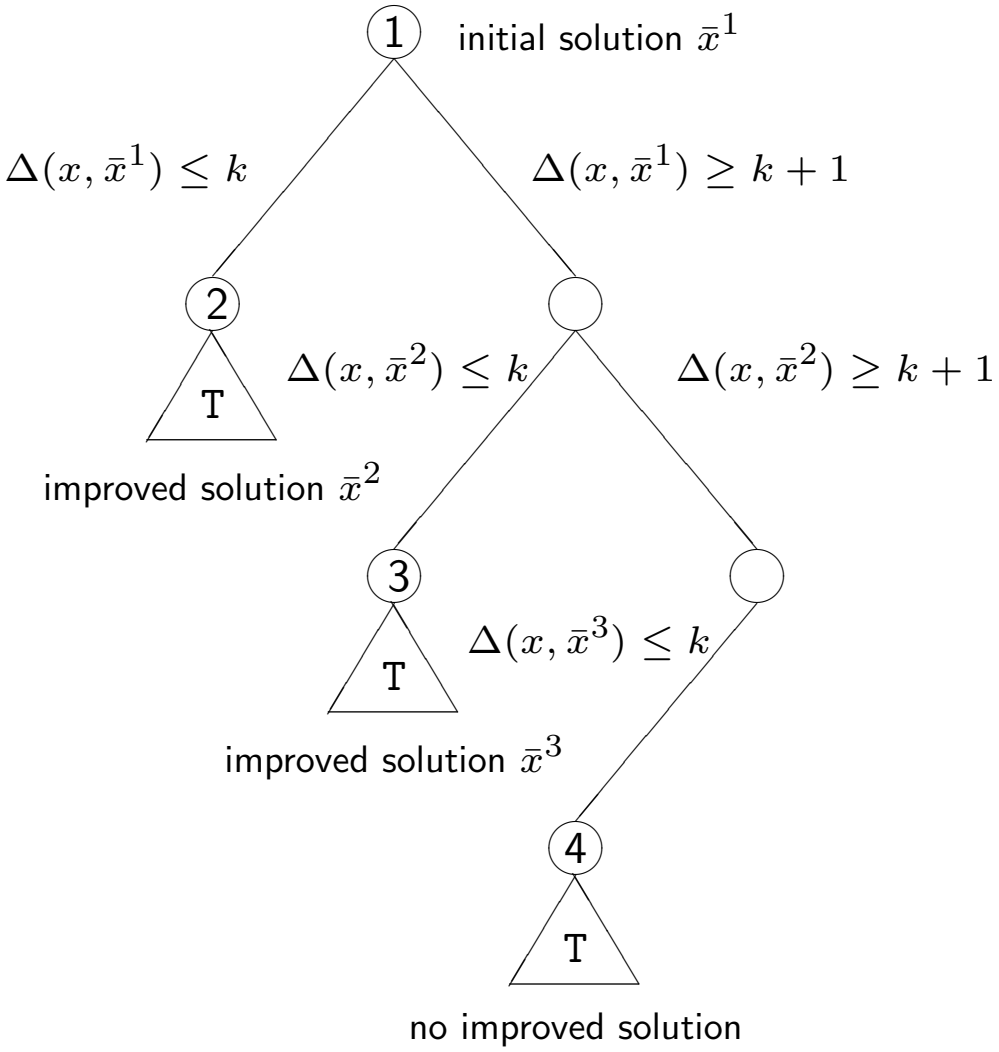
# The basic local branching scheme



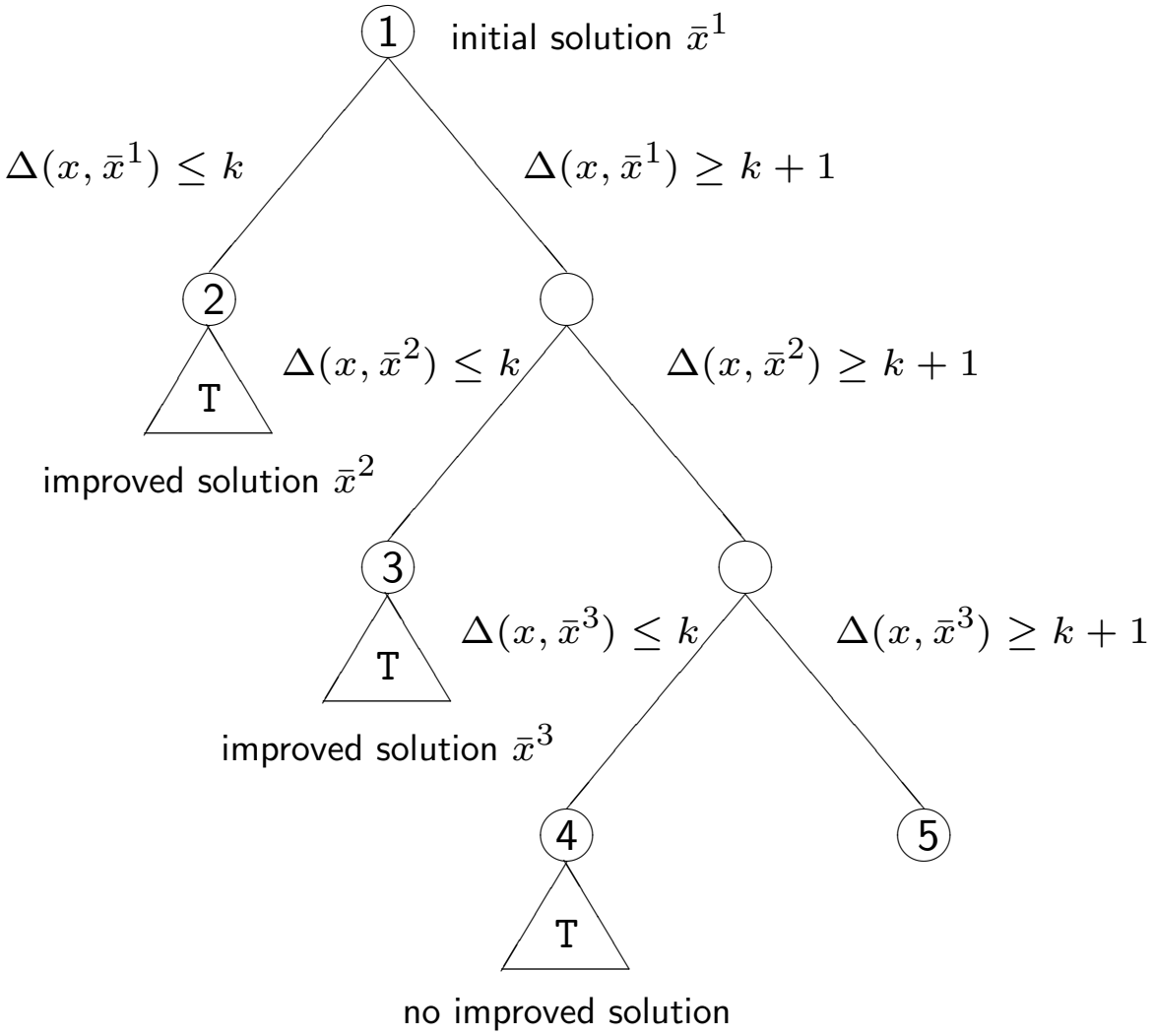
# The basic local branching scheme



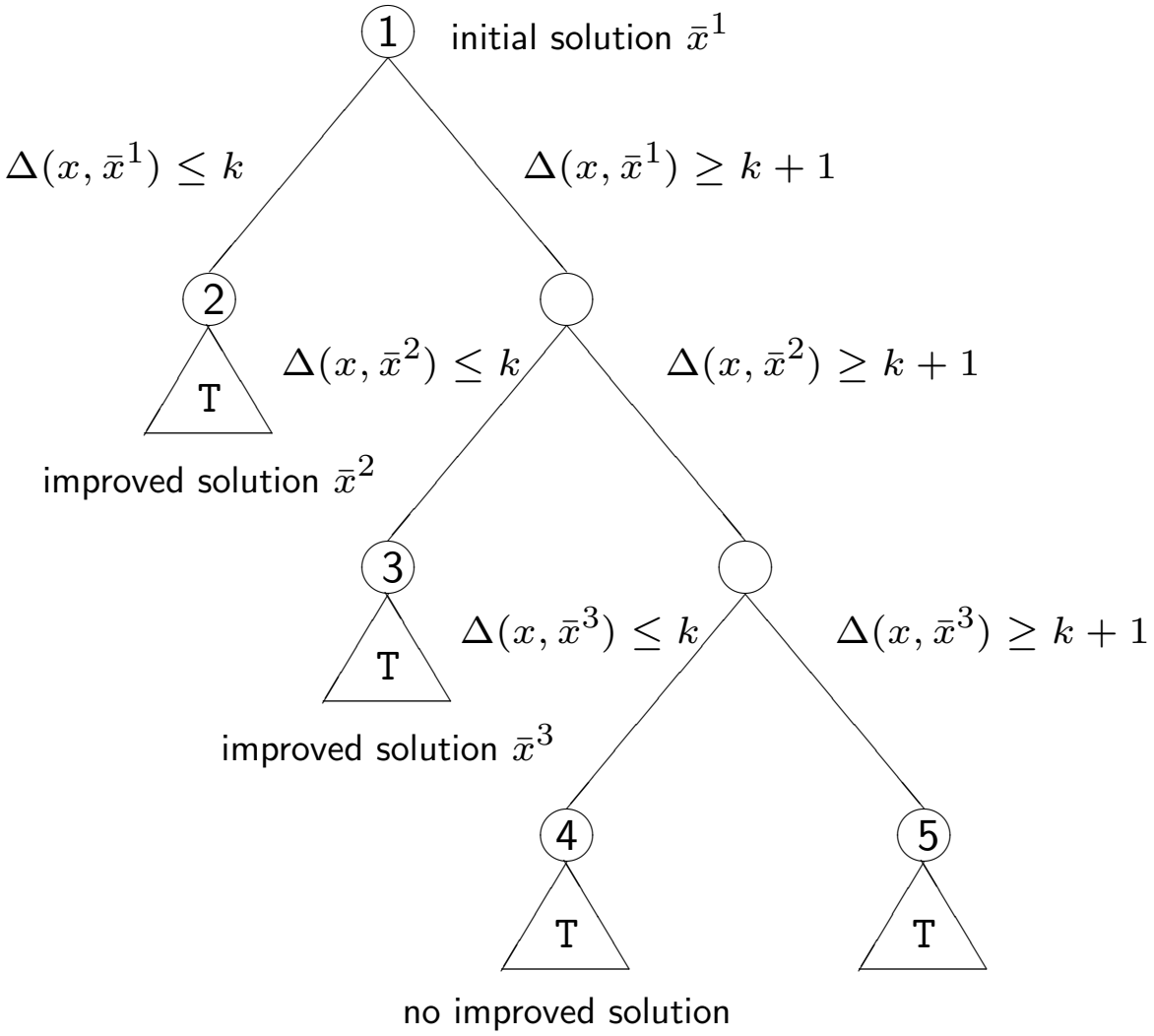
# The basic local branching scheme



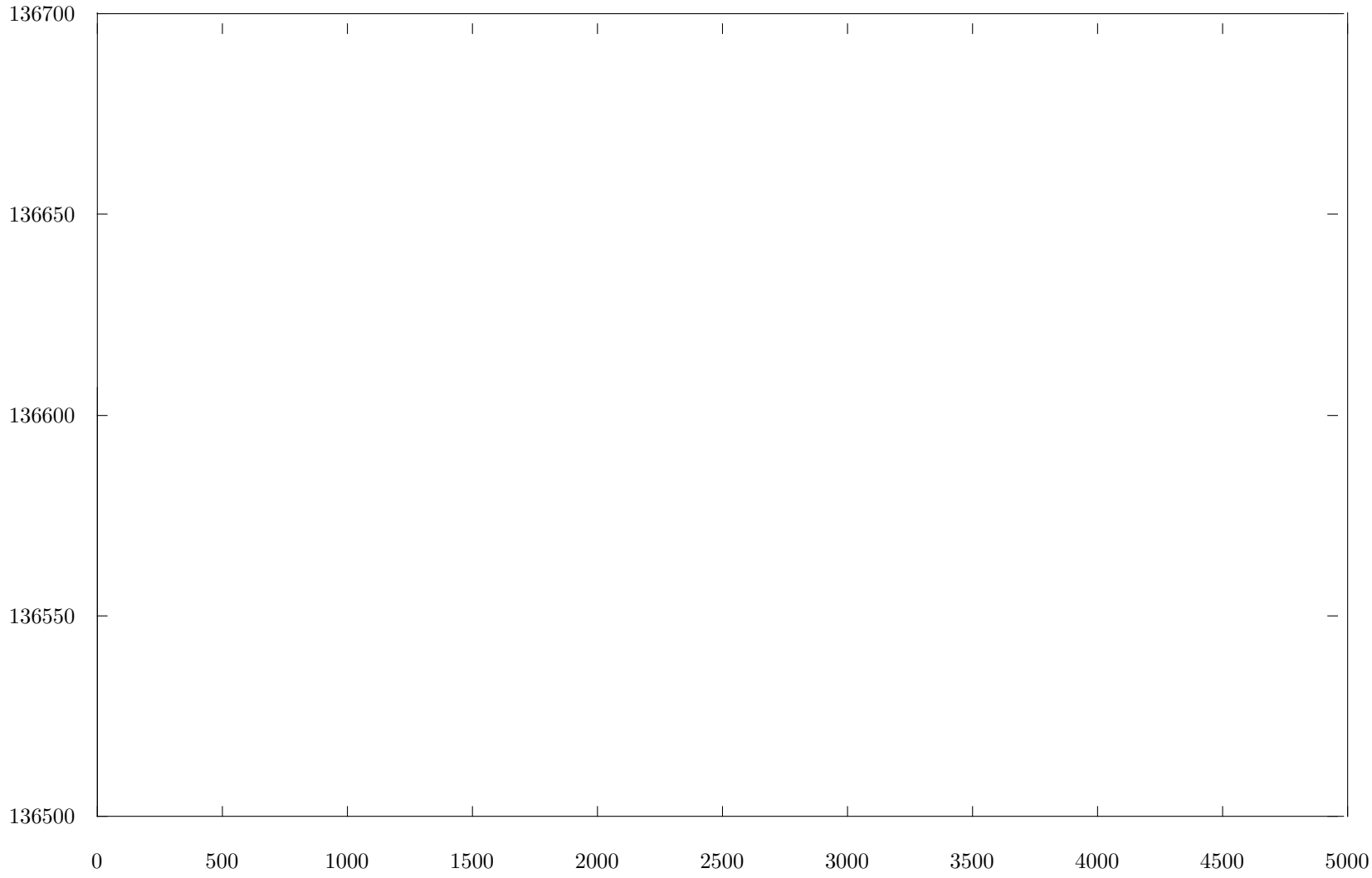
# The basic local branching scheme



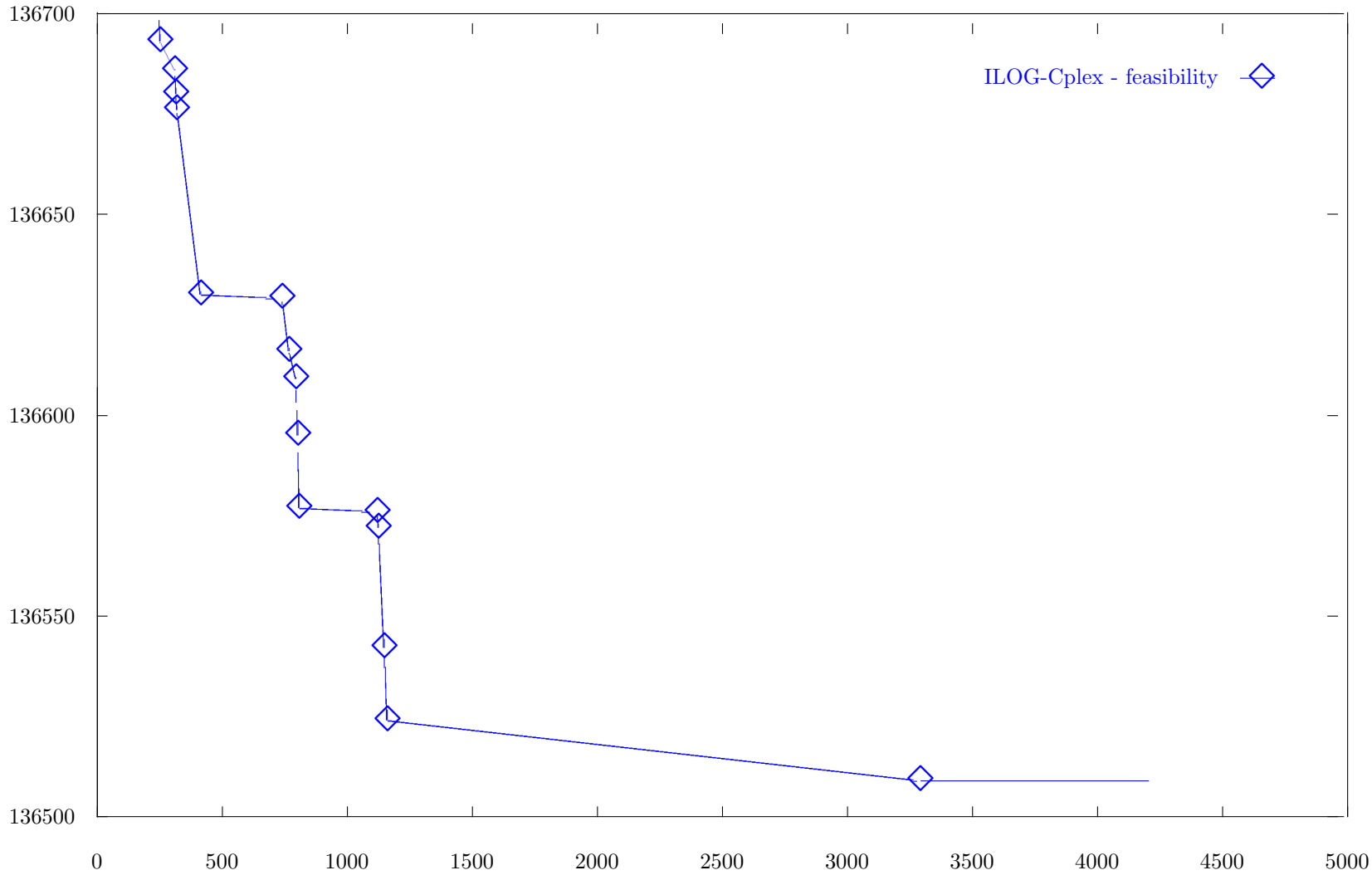
# The basic local branching scheme



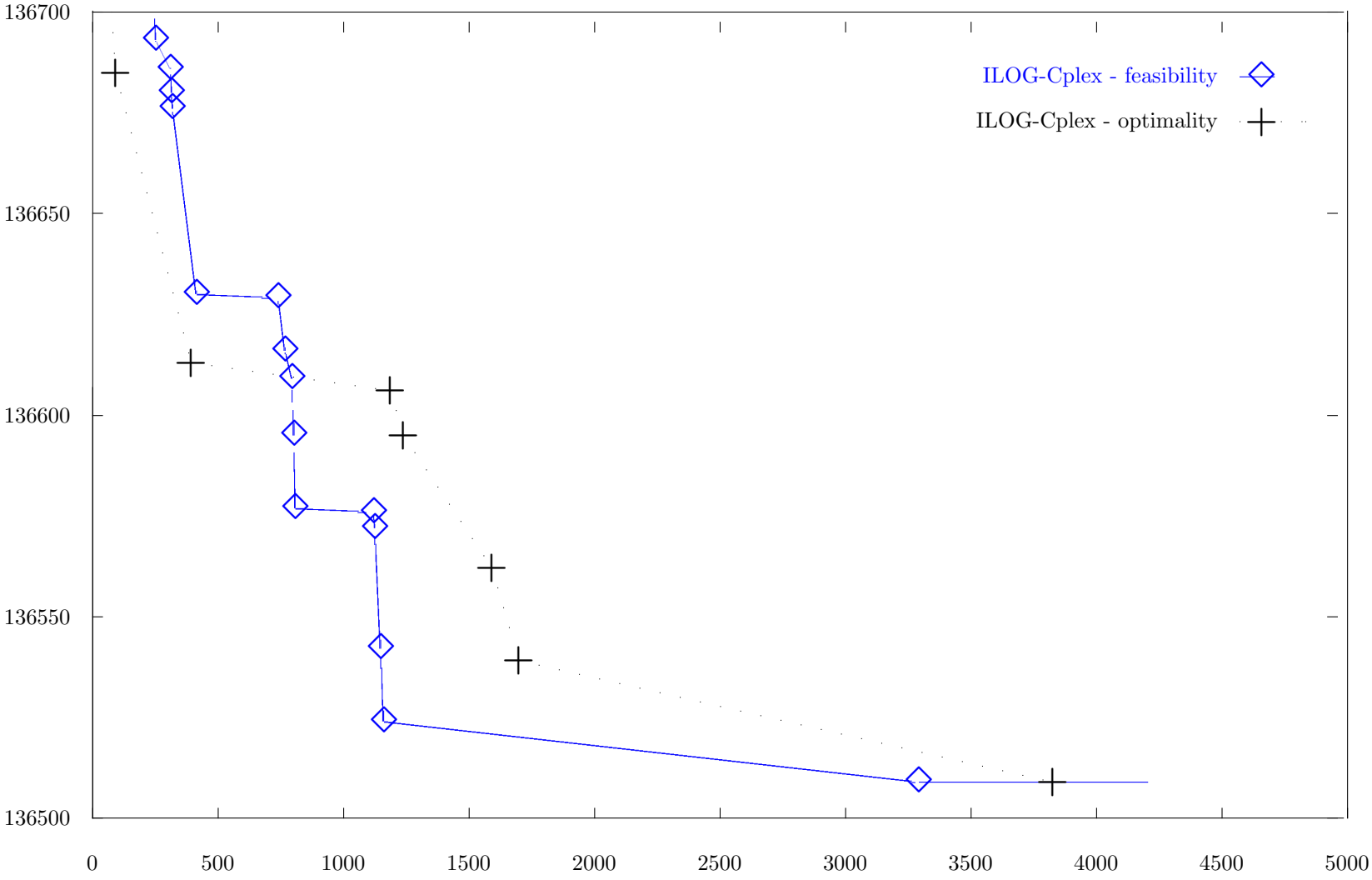
# Solving MIP instance `tr24-15` (solution value vs. CPU seconds)



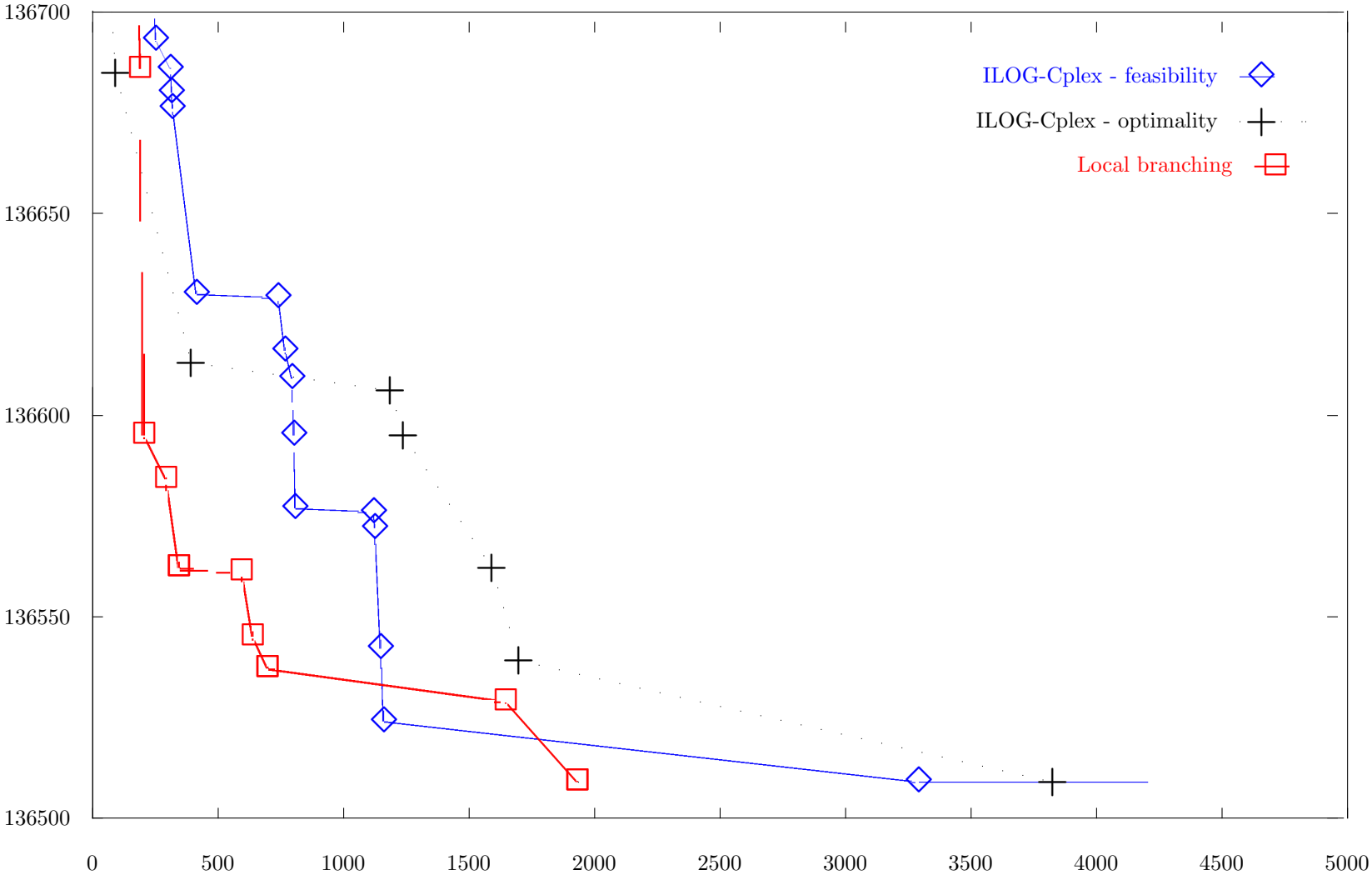
# Solving MIP instance `tr24-15` (solution value vs. CPU seconds)



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



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



# Local branching as an exact method: comments

- MIP solver: ILOG-Cplex 7.0,
  computer: Digital Alpha Ultimate Workstation 533 MHz.

# Local branching as an exact method: comments

- MIP solver: ILOG-Cplex 7.0,
  computer: Digital Alpha Ultimate Workstation 533 MHz.
- The reference solution $\bar{x}^1$ is obtained by running the MIP solver until the "first" solution is found.

# Local branching as an exact method: comments

- MIP solver: ILOG-Cplex 7.0,
  computer: Digital Alpha Ultimate Workstation 533 MHz.
- The reference solution $\bar{x}^1$ is obtained by running the MIP solver until the "first" solution is found.
- The local branching method concludes its run after 1,878 CPU seconds, whereas ILOG-Cplex 7.0 in its optimization version converges to optimality within 3,827 CPU seconds (the feasibility version is unable to prove optimality within a time limit of 6,000 CPU seconds).

# Local branching as an exact method: comments

- MIP solver: ILOG-Cplex 7.0,
  computer: Digital Alpha Ultimate Workstation 533 MHz.
- The reference solution $\bar{x}^1$ is obtained by running the MIP solver until the "first" solution is found.
- The local branching method concludes its run after 1,878 CPU seconds, whereas ILOG-Cplex 7.0 in its optimization version converges to optimality within 3,827 CPU seconds (the feasibility version is unable to prove optimality within a time limit of 6,000 CPU seconds).

- The method highly depends on the reference solution $\bar{x}$.

# Local branching as an exact method: comments

- MIP solver: ILOG-Cplex 7.0,
  computer: Digital Alpha Ultimate Workstation 533 MHz.
- The reference solution $\bar{x}^1$ is obtained by running the MIP solver until the "first" solution is found.
- The local branching method concludes its run after 1,878 CPU seconds, whereas ILOG-Cplex 7.0 in its optimization version converges to optimality within 3,827 CPU seconds (the feasibility version is unable to prove optimality within a time limit of 6,000 CPU seconds).

- The method highly depends on the reference solution $\bar{x}$.
- In the nodes of the scheme which are explored through tactical branching (T-nodes), a large number of branch-and-bound nodes are enumerated but the information associated with them is in some sense "lost" in the following.

# Local branching as an exact method: comments

- MIP solver: ILOG-Cplex 7.0,
  computer: Digital Alpha Ultimate Workstation 533 MHz.
- The reference solution $\bar{x}^1$ is obtained by running the MIP solver until the "first" solution is found.
- The local branching method concludes its run after 1,878 CPU seconds, whereas ILOG-Cplex 7.0 in its optimization version converges to optimality within 3,827 CPU seconds (the feasibility version is unable to prove optimality within a time limit of 6,000 CPU seconds).

- The method highly depends on the reference solution $\bar{x}$.
- In the nodes of the scheme which are explored through tactical branching (T-nodes), a large number of branch-and-bound nodes are enumerated but the information associated with them is in some sense "lost" in the following.
- The enhanced convergence behavior of the local branching scheme in proving optimality cannot be guaranteed in all cases: we are currently working to a project devoted to this specific matter.

# An enhanced heuristic solution scheme

- Despite the nice behavior shown, the main objective is to devise a general-purpose heuristic approach combining local search and MIP.

# An enhanced heuristic solution scheme

- Despite the nice behavior shown, the main objective is to devise a general-purpose heuristic approach combining local search and MIP.
- **Imposing a time limit on the left-branch nodes**:

  In some cases, the exact solution of the left-branch node can be very 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 limit for the left-branch computation. TL

# An enhanced heuristic solution scheme (cont.d)

- Despite the nice behavior shown, the main objective is to devise a general-purpose heuristic approach combining local search and MIP.
- **Imposing a time limit on the left-branch nodes**:

  In some cases, the exact solution of the left-branch node can be very 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 limit for the left-branch computation. TL

# An enhanced heuristic solution scheme (cont.d)

- Despite the nice behavior shown, the main objective is to devise a general-purpose heuristic approach combining local search and MIP.

- **Imposing a time limit on the left-branch nodes**:

  In some cases, the exact solution of the left-branch node can be very 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 limit for the left-branch computation. TL

- **Diversification**:

  A further improvement of the heuristic performance of the method can be obtained by exploiting well-known diversification mechanisms borrowed from local search metaheuristics.

  In our scheme, diversification is worth applying whenever the current left-node is proved to contain no improving solutions. Div

# Working with a node time limit

In case the time limit is exceeded, we have two cases:

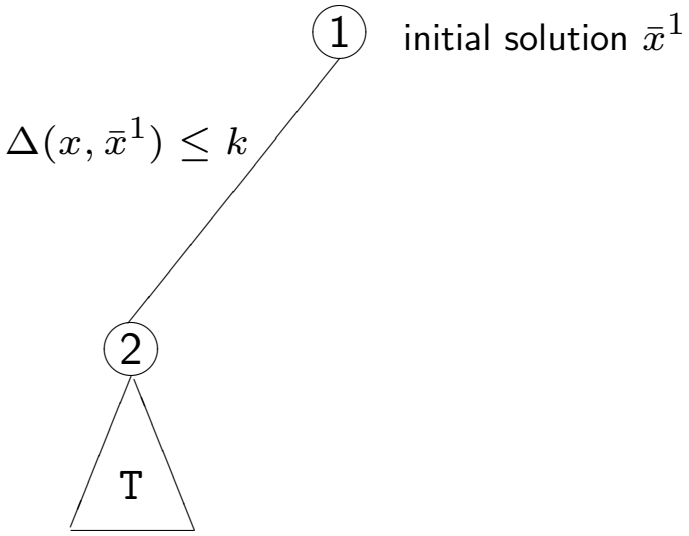
- **Case (a)**:
  If the incumbent solution has been improved, we backtrack to the father node and create a new left-branch node associated with the new incumbent solution, without modifying the value of parameter $k$. case (a)

- **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)

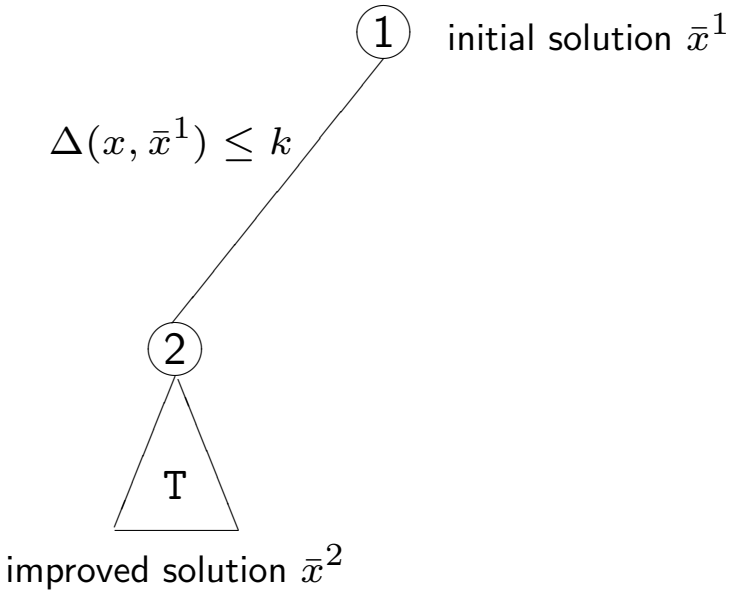
# Working with a node time limit: case (a)

① initial solution $\bar{x}^1$

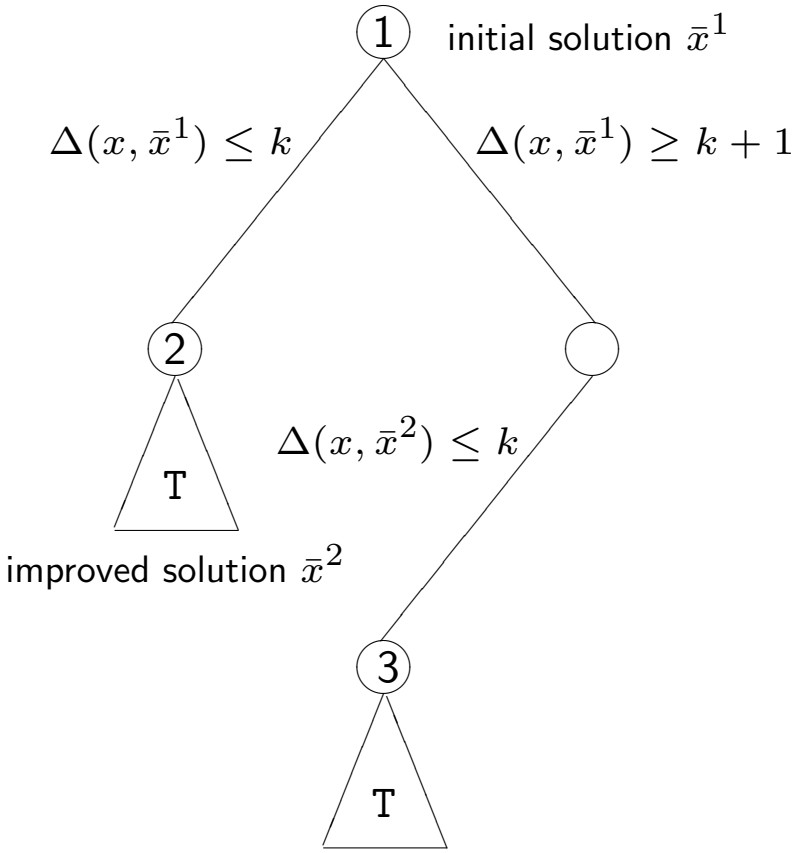
# Working with a node time limit: case (a)



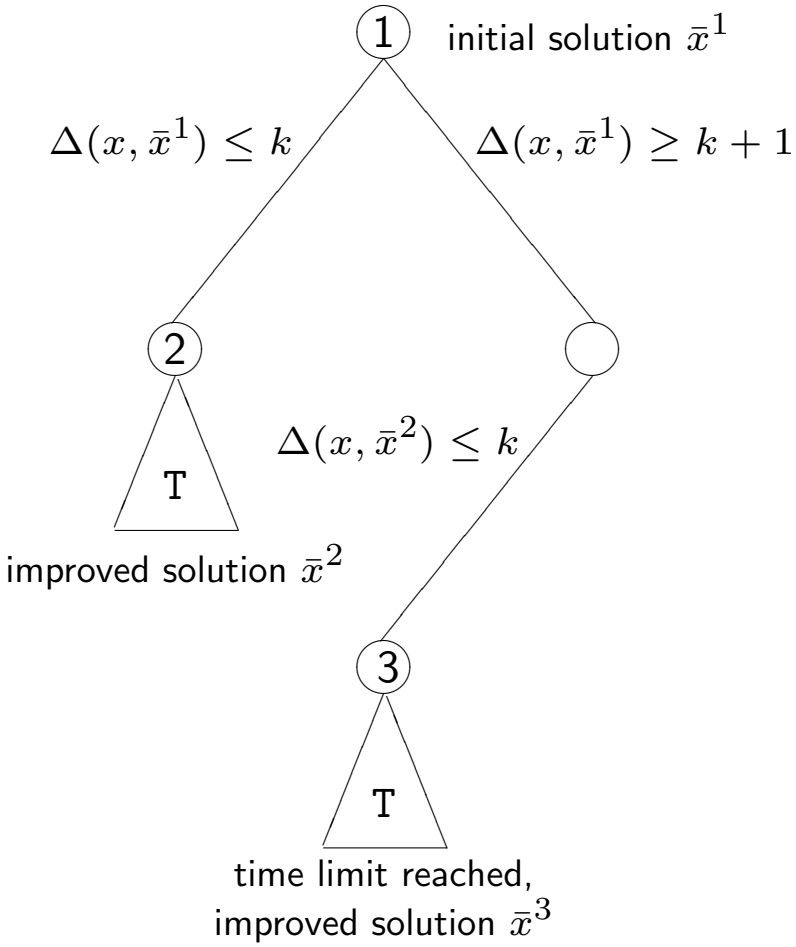
# Working with a node time limit: case (a)



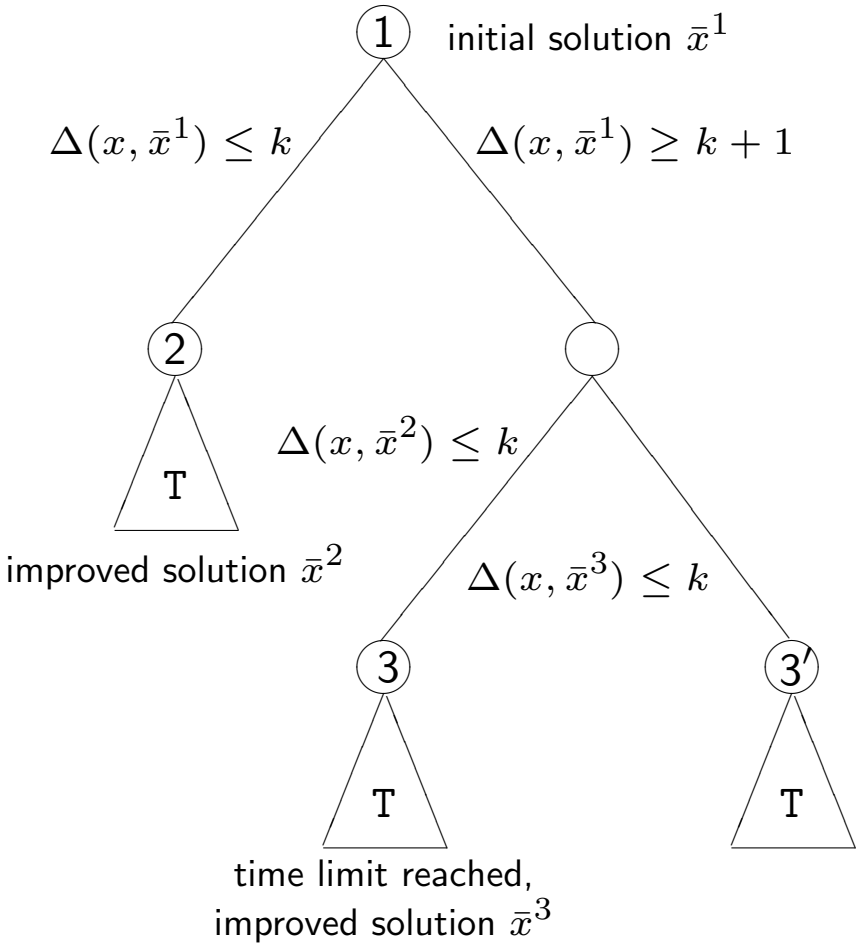
# Working with a node time limit: case (a)



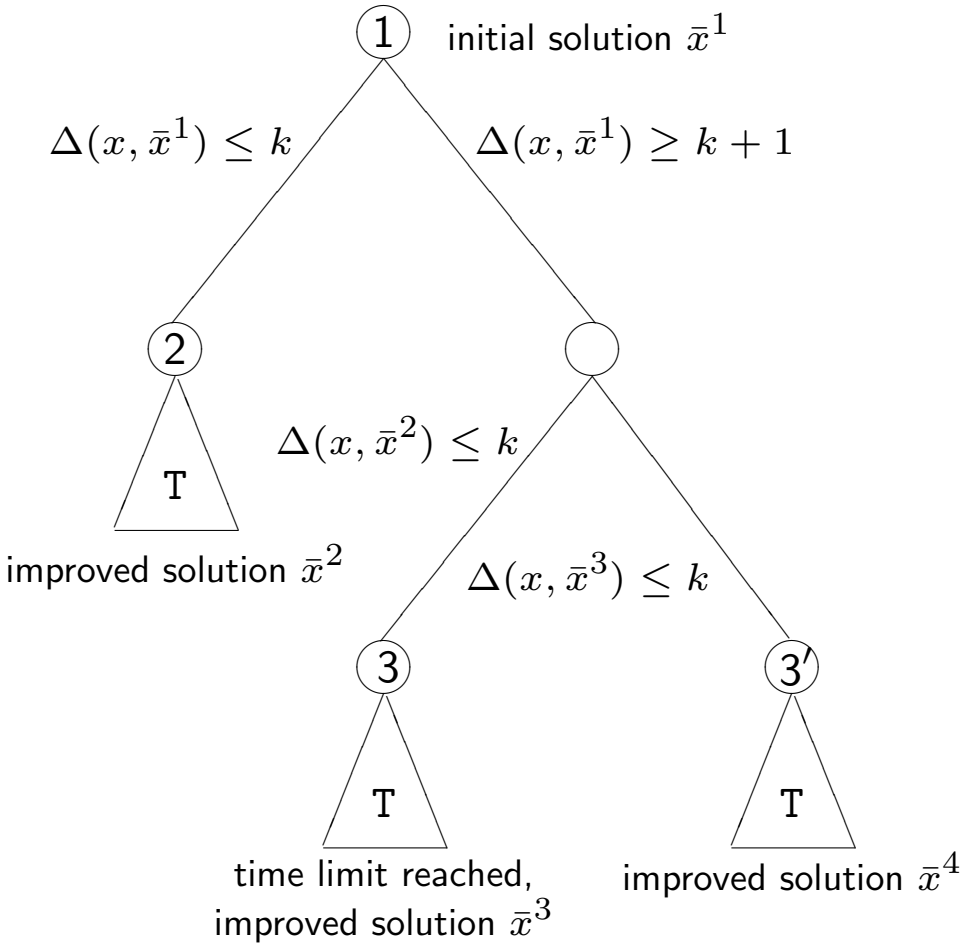
# Working with a node time limit: case (a)



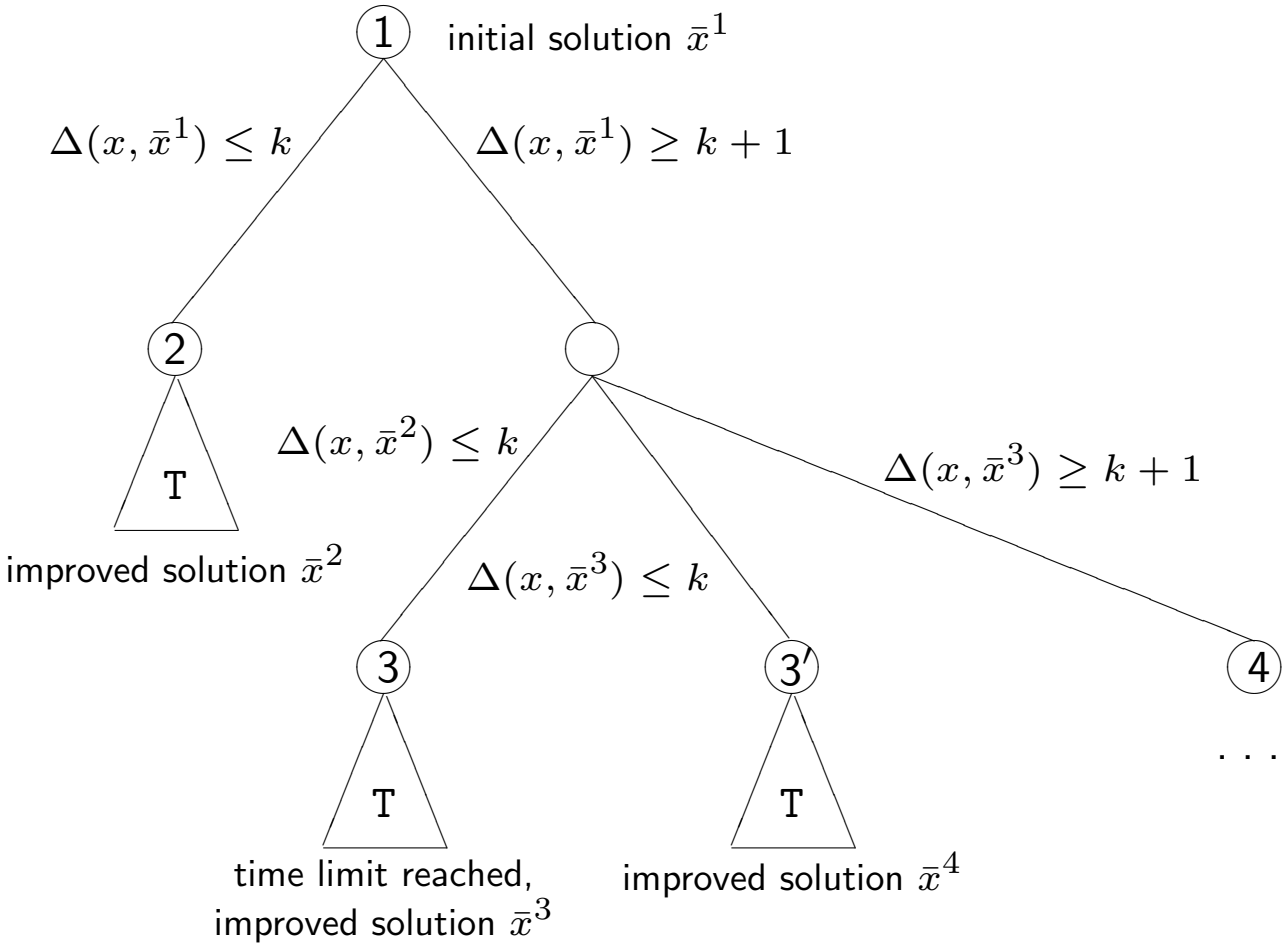
# Working with a node time limit: case (a)



# Working with a node time limit: case (a)



# Working with a node time limit: case (a)

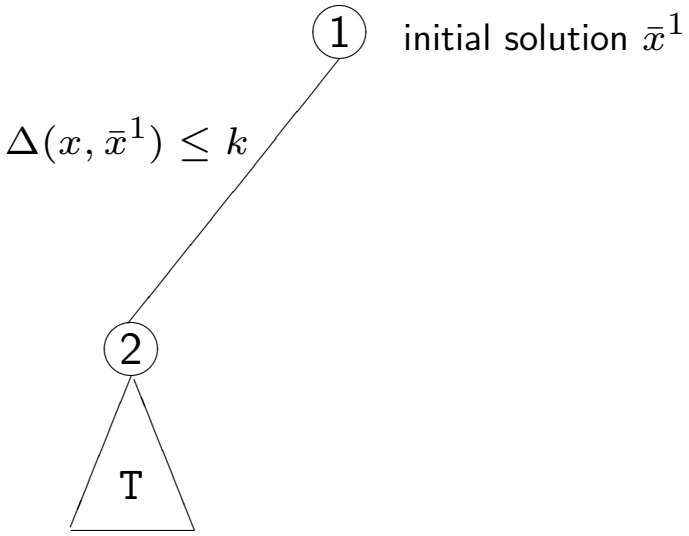



go back

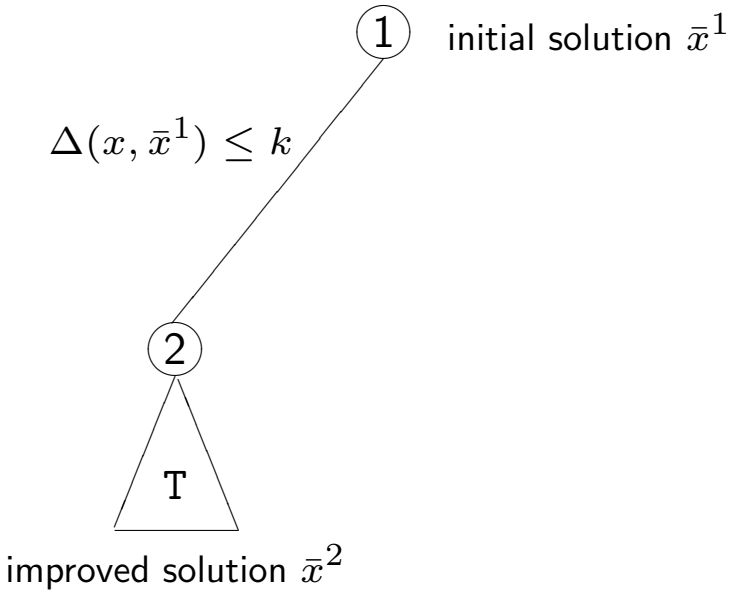
# Working with a node time limit: case (b)

① initial solution $\bar{x}^1$

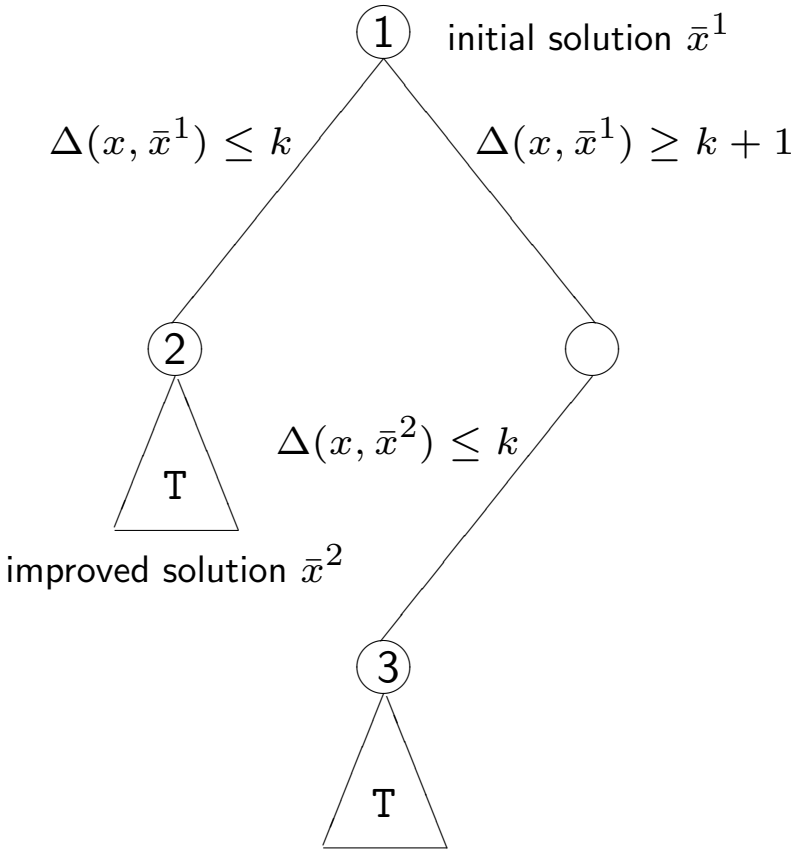
# Working with a node time limit: case (b)



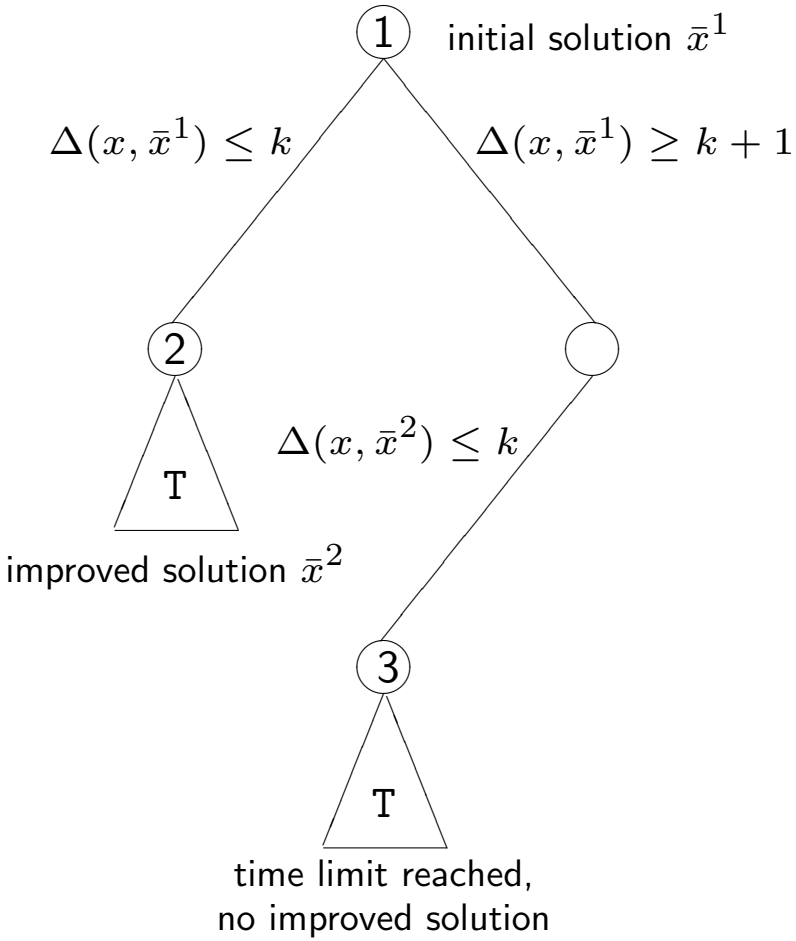
# Working with a node time limit: case (b)



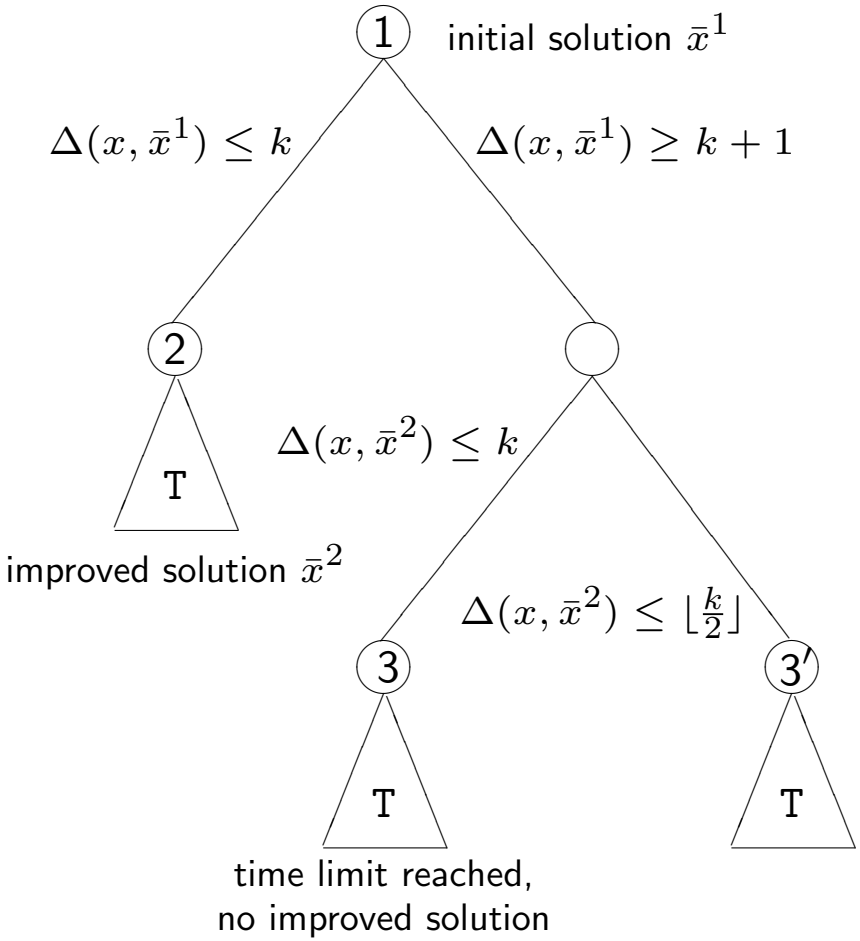
# Working with a node time limit: case (b)



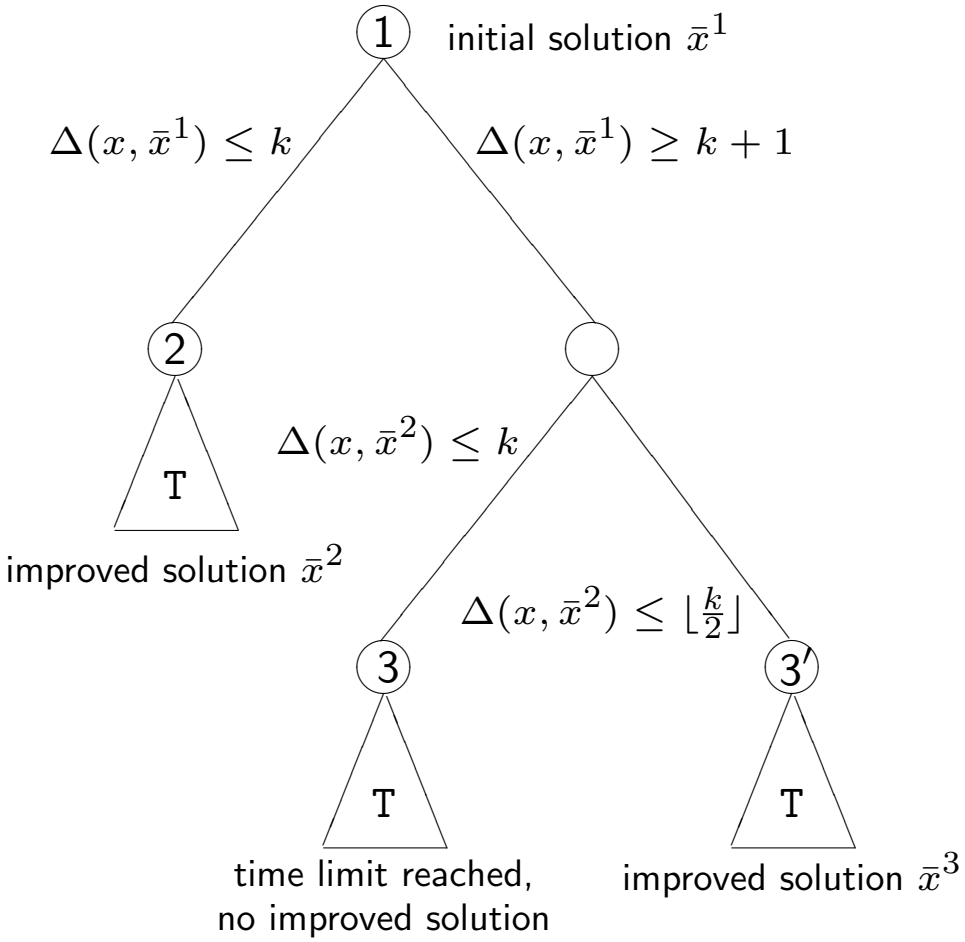
# Working with a node time limit: case (b)



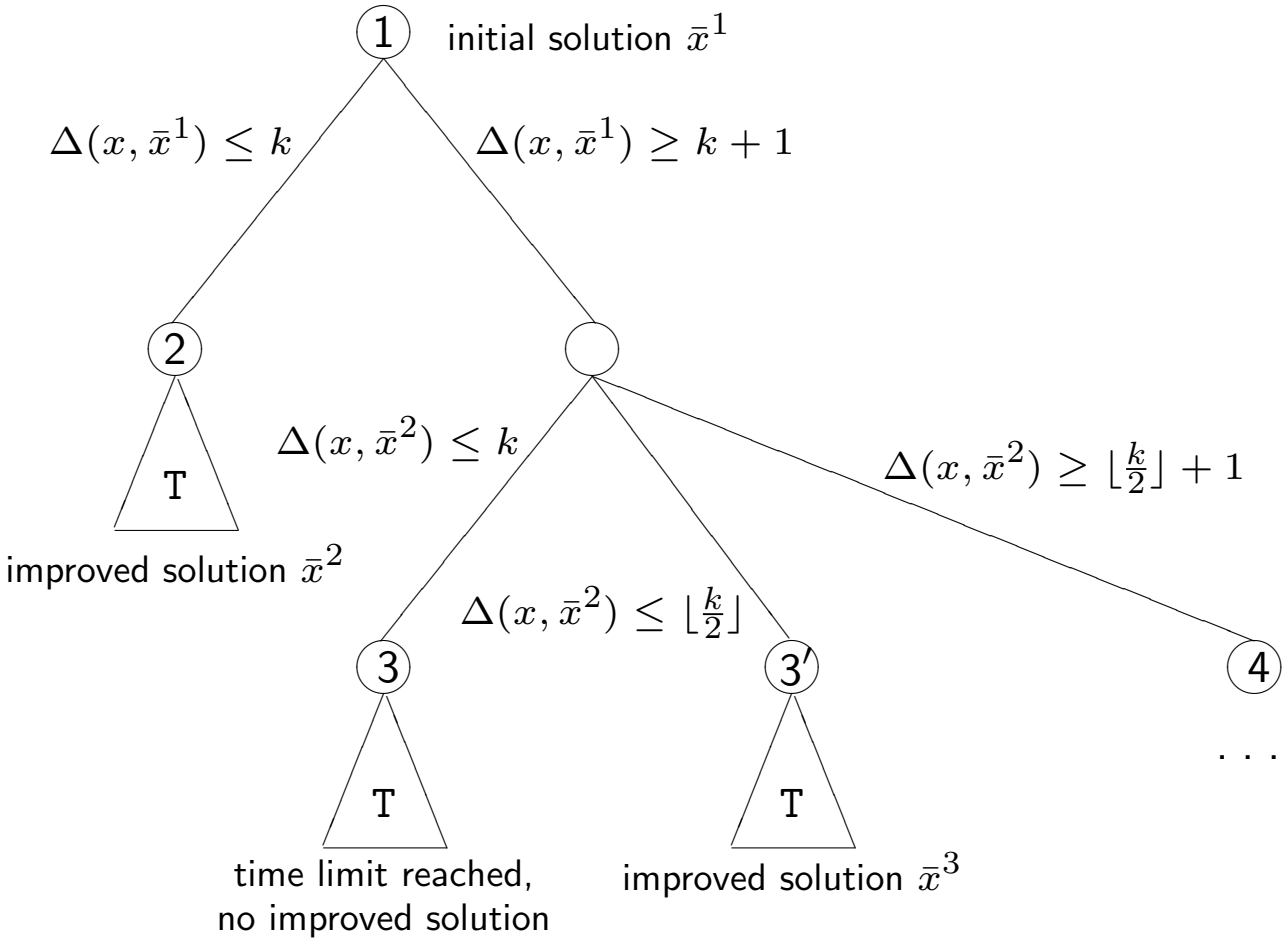
# Working with a node time limit: case (b)



# Working with a node time limit: case (b)

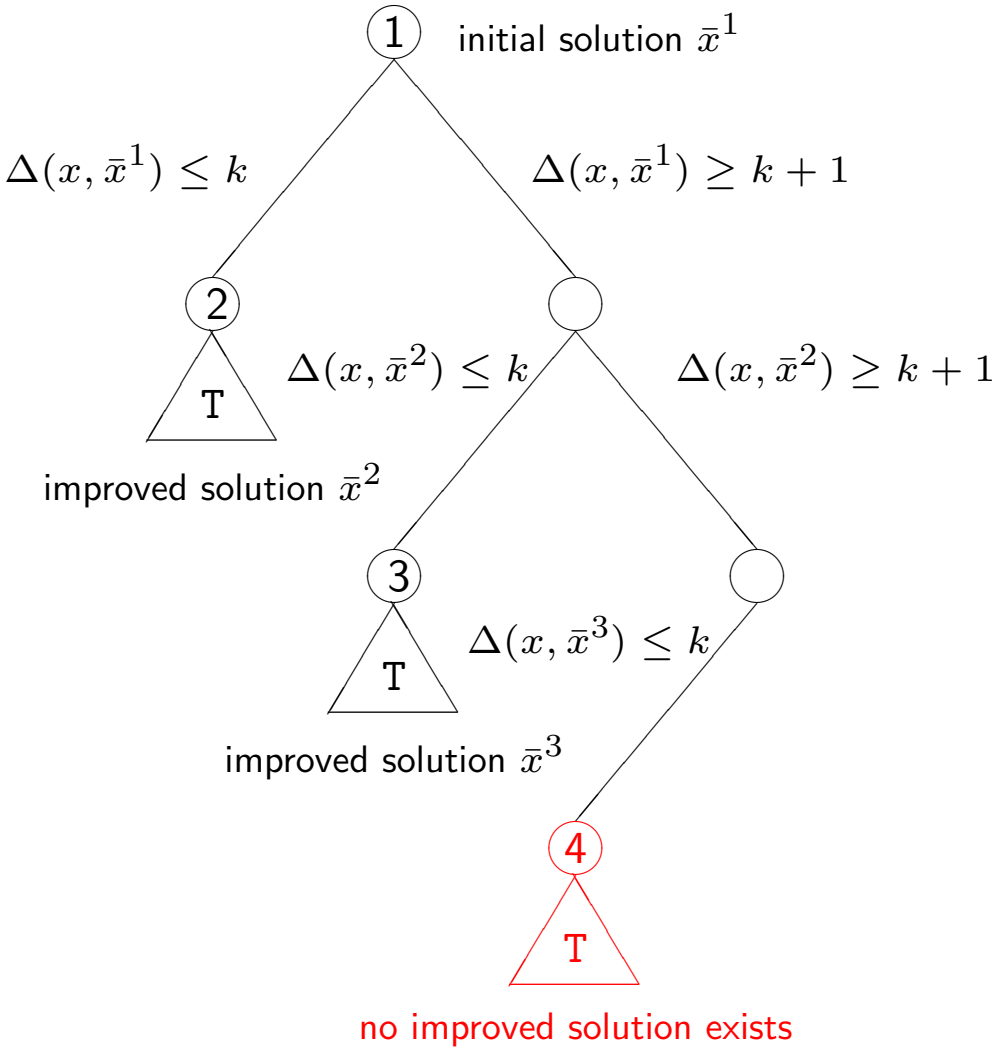


# Working with a node time limit: case (b)




go back

# Diversification



# Diversification (cont.d)

In order to keep a strategic control on the enumeration even in this situation, we use two different diversification mechanisms:

# Diversification (cont.d)

In order to keep a strategic control on the enumeration even in this situation, we use two different diversification mechanisms:

- **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]

# Diversification (cont.d)

In order to keep a strategic control on the enumeration even in this situation, we use two different diversification mechanisms:

- **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:

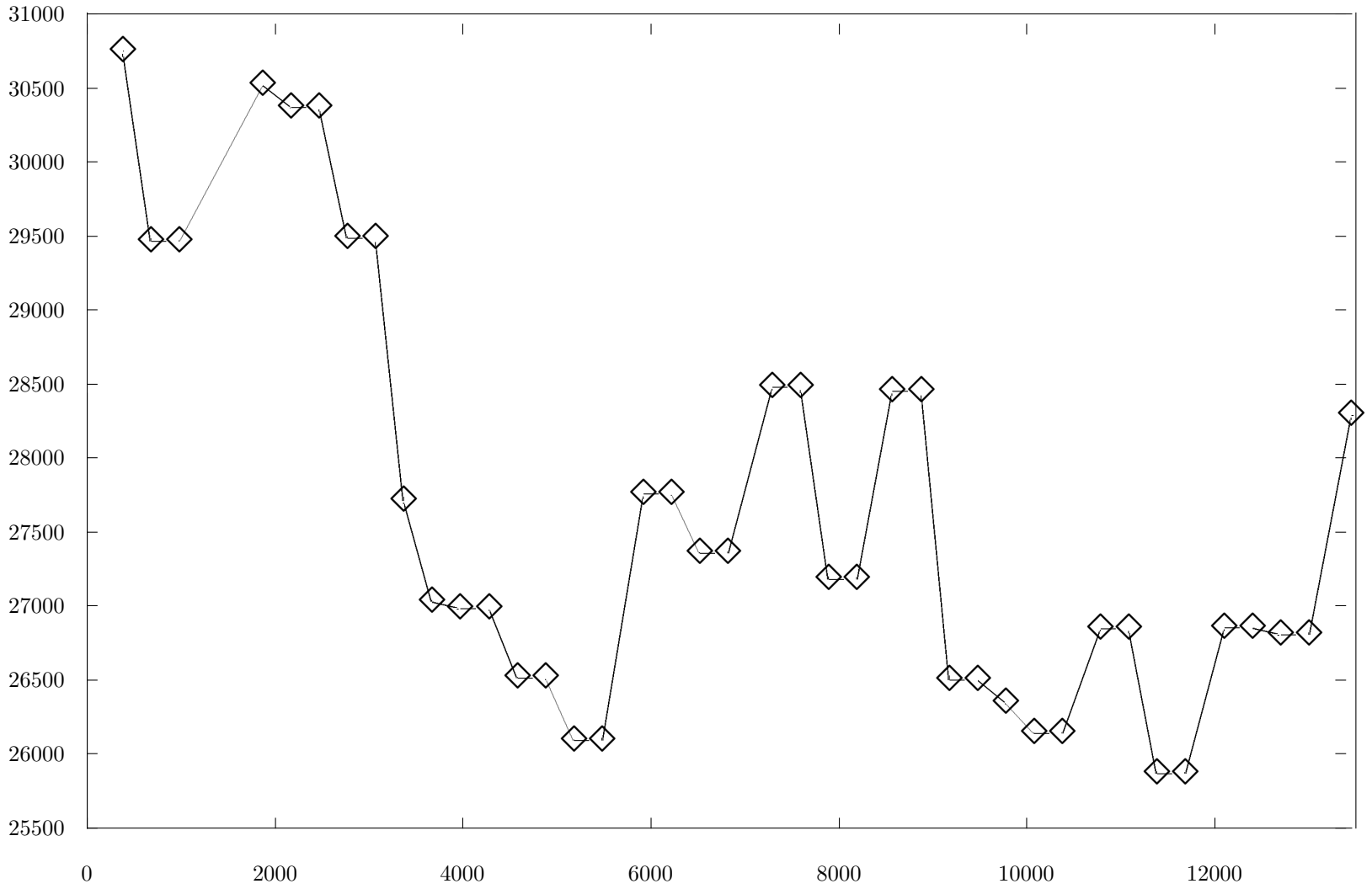
  We look for a solution (typically worse than the incumbent one) which is not "too far" from the current reference solution.

  We apply tactical branching to the current problem amended by $\Delta(x, \bar{x}^3) \leq k + 2\lceil k/2 \rceil$, 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 then used as the new reference solution.

# LocBra as a heuristic for instance B1C1S1 (solution value vs. CPU seconds)



# Computational results (1)

| | | 1 hour | | | 3 hours | | | 5 hours | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Name | Gap | cpx-O | cpx-F | LocBra | cpx-O | cpx-F | LocBra | cpx-O | cpx-F | LocBra |
| mkc | % | 3.765 | 2.399 | 2.281 | 3.765 | 1.294 | 0.404 | 3.765 | 1.294 | **0** |
| swath | % | 94.504 | 2.507 | 1.599 | 26.374 | 1.153 | **0** | 26.374 | 1.153 | **0** |
| danoint | % | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| markshare1 | Abs. | 1 | 5 | 10 | 1 | **0** | 2 | 1 | **0** | 2 |
| markshare2 | Abs. | 9 | 1 | 34 | 9 | **0** | 11 | 9 | **0** | 11 |
| arki001 | % | 0.024 | 0.028 | **0** | 0.017 | 0.016 | **0** | 0.013 | 0.013 | **0** |
| seymour | Abs. | 11 | 5 | 2 | 11 | 3 | **0** | 11 | 3 | **0** |
| net12 | Abs. | – | – | – | 41 | – | 41 | 41 | **0** | 41 |
| biella1 | % | 0.256 | 31.313 | 0.241 | 0.256 | 18.272 | 0.172 | 0.256 | 11.475 | **0** |
| NSR8K* | % | – | – | – | 1843.8 | 1526.4 | 109.3 | 1426.8 | 1526.4 | **0** |
| rail507 | Abs. | 2 | 6 | 1 | 2 | 5 | **0** | 1 | 5 | **0** |
| rail2536c | Abs. | 5 | 3 | **0** | 1 | 3 | **0** | **0** | 3 | **0** |
| rail2586c | Abs. | 54 | 34 | 7 | 14 | 34 | 2 | 14 | 34 | **0** |
| rail4284c | Abs. | 51 | 40 | 10 | 42 | 40 | 2 | 38 | 40 | **0** |
| rail4872c | Abs. | 73 | 69 | 27 | 70 | 49 | 9 | 45 | 49 | **0** |

*Gaps for NSR8K refer to 1 hour, 5 hours, and 10 hours of CPU time, respectively.

# Computational results (2)

| | | 1 hour | | | 3 hours | | | 5 hours | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Name | Gap | cpx-O | cpx-F | LocBra | cpx-O | cpx-F | LocBra | cpx-O | cpx-F | LocBra |
| glass4 | % | 22.835 | 7.087 | 19.685 | 14.226 | 7.087 | 10.236 | 13.386 | 7.087 | **0** |
| UMTS | % | 6.403 | **0** | 2.413 | 6.403 | **0** | 1.216 | 6.403 | **0** | 0.126 |
| van | % | – | – | – | 30.845 | **0** | **0** | 5.108 | **0** | **0** |
| roll3000 | % | 2.763 | 3.804 | 3.131 | 2.763 | 3.804 | **0** | 2.763 | 3.375 | **0** |
| nsrand_ipx | % | 0.932 | 0.932 | 0.621 | 0.932 | 0.932 | **0** | 0.932 | 0.932 | **0** |
| A1C1S1 | % | 7.297 | 5.438 | 4.464 | 5.569 | 5.438 | 2.361 | 5.429 | 3.423 | **0** |
| A2C1S1 | % | 7.615 | 7.261 | 0.995 | 6.379 | 5.123 | **0** | 5.846 | 5.079 | **0** |
| B1C1S1 | % | 11.672 | 13.689 | 4.495 | 11.672 | 7.749 | 0.863 | 11.672 | 7.132 | **0** |
| B2C1S1 | % | 18.196 | 0.268 | 11.642 | 18.196 | **0** | 5.037 | 14.734 | **0** | 5.037 |
| tr12-30 | % | 0.036 | 0.573 | 0.622 | 0.007 | 0.410 | 0.332 | **0** | 0.389 | 0.332 |
| sp97ar | % | 2.494 | 0.842 | 1.171 | 2.494 | 0.428 | **0** | 2.376 | 0.124 | **0** |
| sp97ic | % | 5.453 | 0.622 | 3.675 | 3.834 | 0.622 | 0.761 | 3.834 | 0.622 | **0** |
| sp98ar | % | 1.724 | 2.715 | 0.602 | 1.724 | 1.409 | **0** | 1.724 | 0.282 | **0** |
| sp98ic | % | 1.350 | 0.872 | 0.247 | 1.350 | 0.872 | **0** | 1.350 | 0.872 | **0** |

# Computational results (3), flexibility

Improved solution values for set covering instances.

| elapsed | LocBra with local branching constraint $\sum_{j \in \overline{S}} (1 - x_j) \leq k' = 10$ | | | | | |
|---|---|---|---|---|---|---|
| Time | seymour | rail507 | rail2536c | rail2586c | rail4284c | rail4872c |
| 1:00 | * 423 | * 174 | 691 | 964 | 1081 | 1588 |
| 3:00 | * 423 | * 174 | 690 | * 954 | * 1076 | 1561 |
| 5:00 | * 423 | * 174 | * 690 | * 954 | * 1071 | * 1552 |

# Computational results (3), flexibility

Improved solution values for set covering instances.

| elapsed | LocBra with local branching constraint $\sum_{j \in \overline{S}} (1 - x_j) \leq k' = 10$ | | | | | |
|---|---|---|---|---|---|---|
| Time | seymour | rail507 | rail2536c | rail2586c | rail4284c | rail4872c |
| 1:00 | * 423 | * 174 | 691 | 964 | 1081 | 1588 |
| 3:00 | * 423 | * 174 | 690 | * 954 | * 1076 | 1561 |
| 5:00 | * 423 | * 174 | * 690 | * 954 | * 1071 | * 1552 |

Alternative LocBra runs.

| | LocBra emphasizing feasibility at the tactical level | | | | | |
|---|---|---|---|---|---|---|
| elapsed | Abs. Gap | Abs. Gap | Abs. Gap | % Gap | % Gap | % Gap |
| Time | markshare1 | markshare2 | net12 | UMTS° | B2C1S1 | tr12-30 |
| 1:00 | 4 | * 5 | – | * -0.048 | 8.317 | 2.597 |
| 3:00 | 3 | * 5 | – | * -0.066 | 8.317 | 2.129 |
| 5:00 | 2 | * 3 | * **0** | * -0.069 | 7.299 | 2.018 |

°The negative gaps for instance UMTS indicate an improvement of the best known

# Local branching extensions

The main **simple** idea discussed opens many interesting fields of application in which the basic local branching framework can extend its range of use.

# Local branching extensions

The main **simple** idea discussed opens many interesting fields of application in which the basic local branching framework can extend its range of use.

- **Tighter integration within the MIP solver.**

# Local branching extensions

The main **simple** idea discussed opens many interesting fields of application in which the basic local branching framework can extend its range of use.

- **Tighter integration within the MIP solver.**

  The described approach uses the MIP solver as a black-box for performing the tactical branchings.

  This is remarkably simple to implement, but has the disadvantage of wasting part of the computational effort devoted, e.g., to the exploration the nodes where no improved solution could be found within the node time limit.

  Therefore, a more integrated (and flexible) framework where the two branching rules work in tight cooperation is expected to produce an enhanced performance.

  [Andreello, Fischetti & Lodi, Work in progress]

# Local branching extensions (cont.d)

- **Local search by branch-and-cut.**

# Local branching extensions (cont.d)

- **Local search by branch-and-cut.**

  All the main ingredients of metaheuristics (defining the current solution neighborhood, dealing with tabu solutions or moves, imposing a proper diversification, etc.) can easily be modeled in terms of linear cuts to be dynamically inserted and removed from the model.

  This naturally leads to the possibility of implementing a full general "new" metaheuristic algorithm possibly taking into account the problem structure.

  Very promising results in this direction. [Fischetti, Polo & Scantamburlo, 2003]

# Local branching extensions (cont.d)

- **Local search by branch-and-cut.**

  All the main ingredients of metaheuristics (defining the current solution neighborhood, dealing with tabu solutions or moves, imposing a proper diversification, etc.) can easily be modeled in terms of linear cuts to be dynamically inserted and removed from the model.

  This naturally leads to the possibility of implementing a full general "new" metaheuristic algorithm possibly taking into account the problem structure.

  Very promising results in this direction. [Fischetti, Polo & Scantamburlo, 2003]

- **Use of local branching constraints within special-purpose codes.**

# Local branching extensions (cont.d)

- **Local search by branch-and-cut.**

  All the main ingredients of metaheuristics (defining the current solution neighborhood, dealing with tabu solutions or moves, imposing a proper diversification, etc.) can easily be modeled in terms of linear cuts to be dynamically inserted and removed from the model.

  This naturally leads to the possibility of implementing a full general "new" metaheuristic algorithm possibly taking into account the problem structure.

  Very promising results in this direction. [Fischetti, Polo & Scantamburlo, 2003]

- **Use of local branching constraints within special-purpose codes.**

  Despite the overall discussion, there is no need of using local branching constraints within a general-purpose MIP solvers.

  These constraints can be integrated within special-purpose codes, both exacts and heuristics, (black-boxes) designed for specific problems so as to enhance their heuristic capability.

  Obviously, the only requirement is that the code is able to cope with linear inequalities.

  Good results in this context. [Hernández-Pérez & Salazar-González, 2003]

# Local branching extensions (cont.d)

- **Dealing with general-integer variables.**

# Local branching extensions (cont.d)

- **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 of relevant importance.

  According to our 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 likely to be largely responsible for the difficulty of the model.

  However, in the general case of integer variables $x_j \mid l_j \leq x_j \leq u_j$, 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 $\mu_j$ 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:

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

# Local branching extensions (cont.d)

- **Working with infeasible solutions.**

# Local branching extensions (cont.d)

- **Working with infeasible solutions.**

  As stated, the local branching framework requires a starting (feasible) reference solution $\bar{x}^1$.

  However, for difficult MIPs (such as, e.g., hard set partitioning models) even the definition of this solution may require an excessive computing time.

  In this case, one should consider the possibility of working with infeasible reference solutions by adding slack variables to (some of) the constraints, while penalizing them in the objective function.

  Preliminary results. [Balas, 2003; Fischetti & Lodi, Work in progress]

# Local branching extensions (cont.d)

- **Working with infeasible solutions.**

  As stated, the local branching framework requires a starting (feasible) reference solution $\bar{x}^1$.

  However, for difficult MIPs (such as, e.g., hard set partitioning models) even the definition of this solution may require an excessive computing time.

  In this case, one should consider the possibility of working with infeasible reference solutions by adding slack variables to (some of) the constraints, while penalizing them in the objective function.

  Preliminary results. [Balas, 2003; Fischetti & Lodi, Work in progress]

  Moreover, another interesting point is using local branching ideas to devise a method to converge to an initial feasible solution without using the branch-and-bound framework.

  This means using the concept of neighborhood to define a distance between a **feasible continuous** solution and an **infeasible integer** one, and then solve a sequence of LPs by minimizing this distance.

  [Fischetti, Glover & Lodi, Work in progress]

# Local branching dissemination

- **Relaxation induced neighborhood search.**

[Danna, Le Pape & Rothberg, 2003]

# Local branching dissemination

- **Relaxation induced neighborhood search.**

[Danna, Le Pape & Rothberg, 2003]

A similar concept of neighborhood was recently introduced by taking into account simultaneously 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.

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.

This method turns out to give very competitive results on general MIPs and it is particularly suitable in the scheduling context where sometimes the problem is very constrained and a non-trivial value of $k$ would be too large (thus defining too large neighborhoods).

# Conclusions

- Integration of local search and metaheuristic within Mixed Integer Programming.

# Conclusions

- Integration of local search and metaheuristic within Mixed Integer Programming.
- Very competitive results with respect to general-purpose heuristic methods.

# Conclusions

- Integration of local search and metaheuristic within Mixed Integer Programming.
- Very competitive results with respect to general-purpose heuristic methods.
- Implemented within the next release of ILOG-Cplex (9.0).

# Conclusions

- Integration of local search and metaheuristic within Mixed Integer Programming.
- Very competitive results with respect to general-purpose heuristic methods.
- Implemented within the next release of ILOG-Cplex (9.0).
- Big advantage: flexibility!

# Conclusions

- Integration of local search and metaheuristic within Mixed Integer Programming.
- Very competitive results with respect to general-purpose heuristic methods.
- Implemented within the next release of ILOG-Cplex (9.0).
- Big advantage: flexibility!
- Tighter integration is needed and

# Conclusions

- Integration of local search and metaheuristic within Mixed Integer Programming.
- Very competitive results with respect to general-purpose heuristic methods.
- Implemented within the next release of ILOG-Cplex (9.0).
- Big advantage: flexibility!
- Tighter integration is needed and

this is a big challenge for this community