A combinatorial optimization problem arising from text classification

Introduction

The amount of unclassified digital documents requires automatic tools for text classification.

Introduction

The amount of unclassified digital documents requires automatic tools for text classification.

A preliminary step for text classification is segmentation:

Introduction

The amount of unclassified digital documents requires automatic tools for text classification.

A preliminary step for text classification is segmentation:

- bag-of-word approach: simple and language independent, but may lead to a data sparseness problem.

Introduction

The amount of unclassified digital documents requires automatic tools for text classification.

A preliminary step for text classification is segmentation:

- bag-of-word approach: simple and language independent, but may lead to a data sparseness problem.
- segmentation based upon a morphological analisys: very refined, but introduces costs for any language used.

Introduction

The amount of unclassified digital documents requires automatic tools for text classification.

A preliminary step for text classification is segmentation:

- bag-of-word approach: simple and language independent, but may lead to a data sparseness problem.
- segmentation based upon a morphological analisys: very refined, but introduces costs for any language used.

A new approach is based upon an optimal reduction of the suffix tree of a training test.

Suffix tree

First a preprocessing on the text is made, reducing the alphabet to lower case letters plus a unique non alphabetic character.

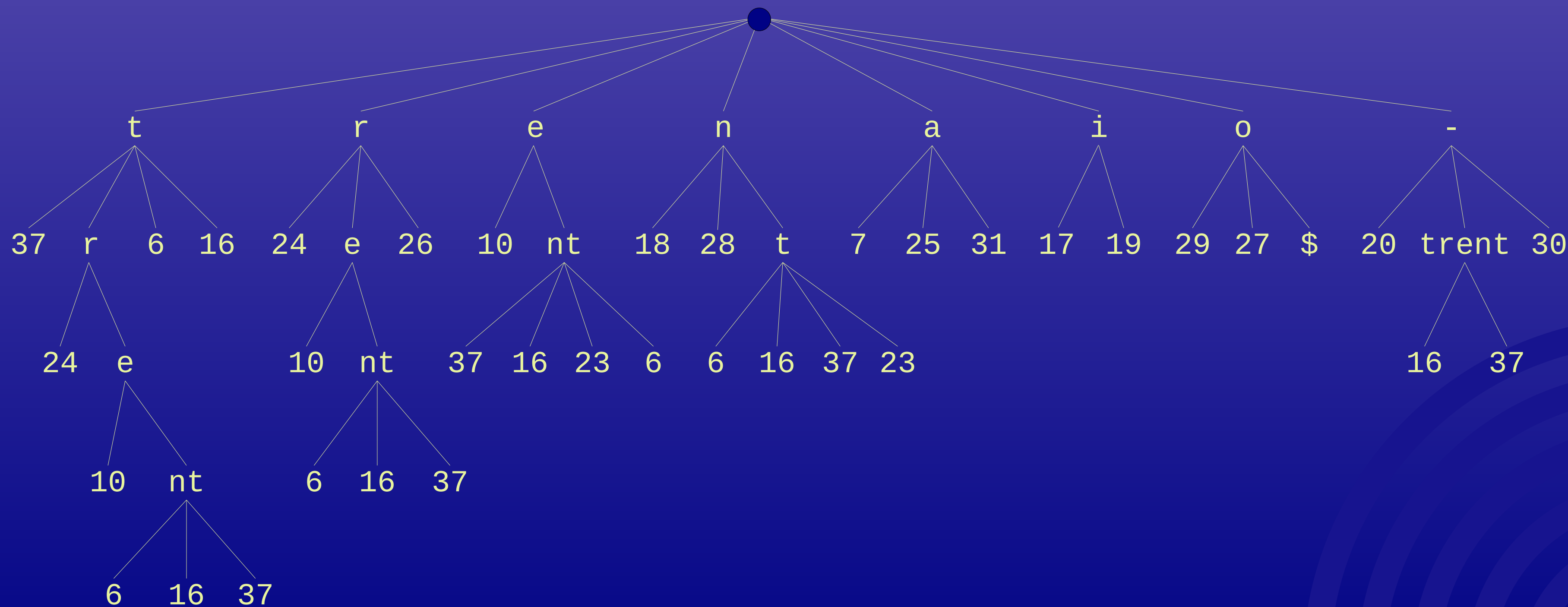
```
t r e n t a t r e - t r e n t i n i - e n t r a r o n o - a - t r e n t o $
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
```

Suffix tree

First a preprocessing on the text is made, reducing the alphabet to lower case letters plus a unique non alphabetic character.

```
t r e n t a t r e - t r e n t i n i - e n t r a r o n o - a - t r e n t o $
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
```

The suffix tree is then built:

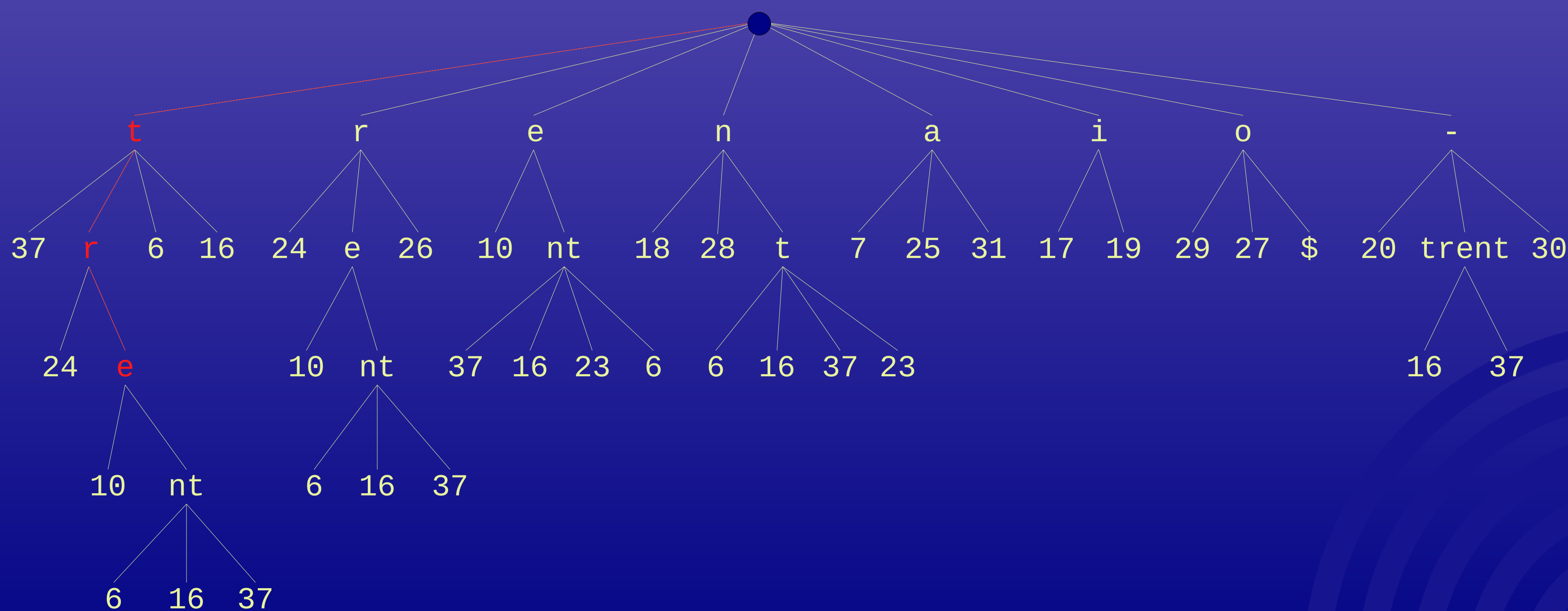


Suffix tree

First a preprocessing on the text is made, reducing the alphabet to lower case letters plus a unique non alphabetic character.

t r e n t a t r e - t r e n t i n i - e n t r a r o n o - a - t r e n t o $
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37

The suffix tree is then built:

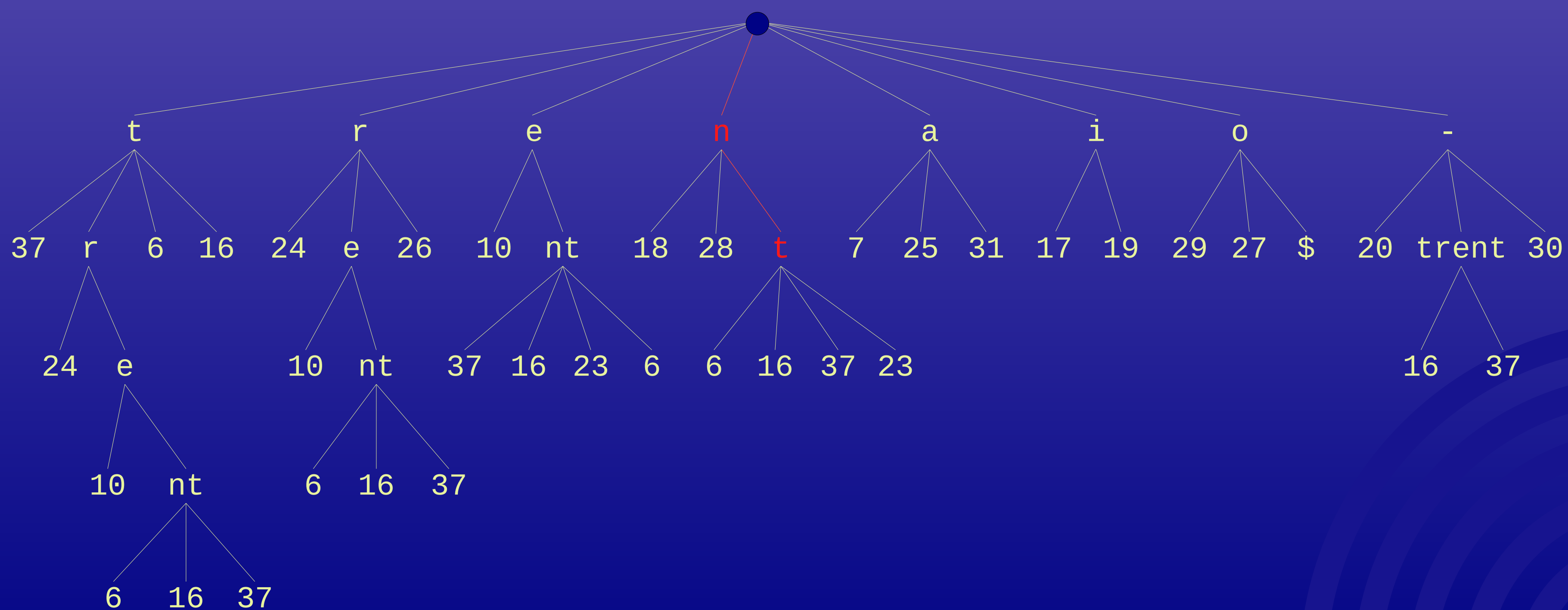


Suffix tree

First a preprocessing on the text is made, reducing the alphabet to lower case letters plus a unique non alphabetic character.

t r e n t a t r e - t r e n t i n i - e n t r a r o n o - a - t r e n t o $
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37

The suffix tree of this text is then built:



Reduction of the suffix tree

A first pruning is made, based upon a minimal and a maximal lenght of the suffix, and a minimal number of occurrences in the text.

Reduction of the suffix tree

A first pruning is made, based upon a minimal and a maximal lenght of the suffix, and a minimal number of occurrences in the text.

Such a reduction is not sufficient, there are still too many suffixes.

Reduction of the suffix tree

A first pruning is made, based upon a minimal and a maximal lenght of the suffix, and a minimal number of occurrences in the text.

Such a reduction is not sufficient, there are still too many suffixes.

In this work we have studied a further reduction of the suffix tree that tries to keep only the *important* suffixes, those that describe the training text in the best way.

TCSS

Given:

TCSS

Given:

- a text W (the training text)

TCSS

Given:

- a text W (the training text)
- a set of strings S (the units of the suffix tree)

TCSS

Given:

- a text W (the training text)
- a set of strings S (the units of the suffix tree)
- the set O of the occurrences of the strings over the text

TCSS

Given:

- a text W (the training text)
- a set of strings S (the units of the suffix tree)
- the set O of the occurrences of the strings over the text

Find $Y \subseteq S$ e $X \subseteq O$ such that:

TCSS

Given:

- a text W (the training text)
- a set of strings S (the units of the suffix tree)
- the set O of the occurrences of the strings over the text

Find $Y \subseteq S$ e $X \subseteq O$ such that:

- the occurrences in X are occurrences of the strings in Y

TCSS

Given:

- a text W (the training text)
- a set of strings S (the units of the suffix tree)
- the set O of the occurrences of the strings over the text

Find $Y \subseteq S$ e $X \subseteq O$ such that:

- the occurrences in X are occurrences of the strings in Y
- the occurrences in X do not overlap (not even partially)

TCSS

Given:

- a text W (the training text)
- a set of strings S (the units of the suffix tree)
- the set O of the occurrences of the strings over the text

Find $Y \subseteq S$ e $X \subseteq O$ such that:

- the occurrences in X are occurrences of the strings in Y
- the occurrences in X do not overlap (not even partially)

The name given to that problem is TCSS, that stands for
Text Covering with Strings Subset.

TCSS

The objectives are:

TCSS

The objectives are:

- maximize the covering of the text by the occurrences in X

TCSS

The objectives are:

- maximize the covering of the text by the occurrences in X
- minimize a cost function of the strings in Y

TCSS

The objectives are:

- maximize the covering of the text by the occurrences in X
- minimize a cost function of the strings in Y

These objectives are in contrast, because to cover a larger portion of text a larger number of strings is needed.

TCSS

The objectives are:

- maximize the covering of the text by the occurrences in X
- minimize a cost function of the strings in Y

These objectives are in contrast, because to cover a larger portion of text a larger number of strings is needed.

For classification purposes the use of long strings is preferable to that of short ones, so the cost coefficient used is the inverse of the string length.

Istances

The number of strings, occurrences and characters obtained for class instances is the following:

CLASS	$\|S\|$	$\|O\|$	$\|W\|$
A	170	2850	8870
B	2200	55700	49000
C	9950	375000	218000

ILP model for TCSS

A formulation for this problem is the following:

ILP model for TCSS

A formulation for this problem is the following:

TCSS) $$\max z = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i$$

s.t. $$\begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j - y_{u(j)} \leqslant 0 & j = 1 \dots |O| \\ x_j \in \{0,1\} & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

ILP model for TCSS

A formulation for this problem is the following:

$$\text{TCSS)} \qquad \max z = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i$$

$$\textit{s.t.} \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j - y_{u(j)} \leqslant 0 & j = 1 \dots |O| \\ x_j \in \{0,1\} & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

Binary variables y for the strings.

ILP model for TCSS

A formulation for this problem is the following:

$$\text{TCSS)} \qquad \max z = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i$$

$$\text{s.t.} \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \ldots |W| \\ x_j - y_{u(j)} \leqslant 0 & j = 1 \ldots |O| \\ x_j \in \{0,1\} & j = 1 \ldots |O| \\ y_i \in \{0,1\} & i = 1 \ldots |S| \end{cases}$$

Binary variables x for the occurrences.

ILP model for TCSS

A formulation for this problem is the following:

$$\text{TCSS)} \qquad \max z = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i$$

$$s.t. \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j - y_{u(j)} \leqslant 0 & j = 1 \dots |O| \\ x_j \in \{0,1\} & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

The objective 1 tries to maximize the number of covered characters.

ILP model for TCSS

A formulation for this problem is the following:

TCSS)
$$\max z = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i$$

$$s.t. \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \ldots |W| \\ x_j - y_{u(j)} \leqslant 0 & j = 1 \ldots |O| \\ x_j \in \{0,1\} & j = 1 \ldots |O| \\ y_i \in \{0,1\} & i = 1 \ldots |S| \end{cases}$$

The objective 2 tries to minimize the cost of the used strings.

ILP model for TCSS

A formulation for this problem is the following:

TCSS)

$$\max z = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i$$

$$\text{s.t.} \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j - y_{u(j)} \leqslant 0 & j = 1 \dots |O| \\ x_j \in \{0,1\} & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

The two objectives are combined with a parameter α.

ILP model for TCSS

A formulation for this problem is the following:

$$\text{TCSS)} \qquad \max z = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i$$

$$\text{s.t.} \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \ldots |W| \\ x_j - y_{u(j)} \leqslant 0 & j = 1 \ldots |O| \\ x_j \in \{0,1\} & j = 1 \ldots |O| \\ y_i \in \{0,1\} & i = 1 \ldots |S| \end{cases}$$

Packing constraints.

ILP model for TCSS

A formulation for this problem is the following:

$$\text{TCSS)} \qquad \max z = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i$$

$$\text{s.t.} \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j - y_{u(j)} \leqslant 0 & j = 1 \dots |O| \\ x_j \in \{0,1\} & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

Variable upper bound constraints.

Subproblem MPP

When Y is fixed, the problem reduces to the maximization of the covering with the occurrences O_Y of these strings.

Subproblem MPP

When Y is fixed, the problem reduces to the maximization of the covering with the occurrences O_Y of these strings.

MPP) $$\max z_{\texttt{MPP}} = \alpha \sum_{j \in O_Y} l_{u(j)} x_j$$

$$\textit{s.t.} \begin{cases} \sum_{j \in O_Y} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j \in \{0, 1\} & j \in O_Y \end{cases}$$

Subproblem MPP

When Y is fixed, the problem reduces to the maximization of the covering with the occurrences O_Y of these strings.

MPP) $$\max z_{\text{MPP}} = \alpha \sum_{j \in O_Y} l_{u(j)} x_j$$

$$\text{s.t.} \begin{cases} \sum_{j \in O_Y} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j \in \{0, 1\} & j \in O_Y \end{cases}$$

For the structure of the matrix $|a_{tj}|$, this problem can be solved in polynomial time.

Subproblem MPP

The text can be seen as a graph:

Subproblem MPP

The text can be seen as a graph:

Every character is a node. Number the nodes in the same order as the characters.

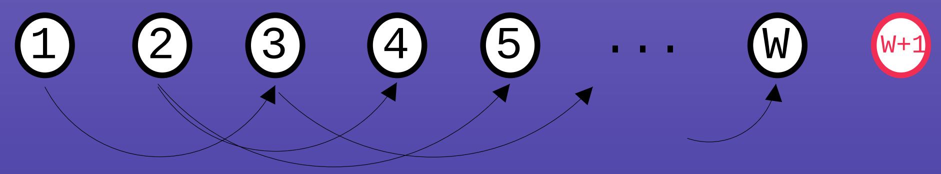
Subproblem MPP

The text can be seen as a graph:

Add one node as last node.

Subproblem MPP

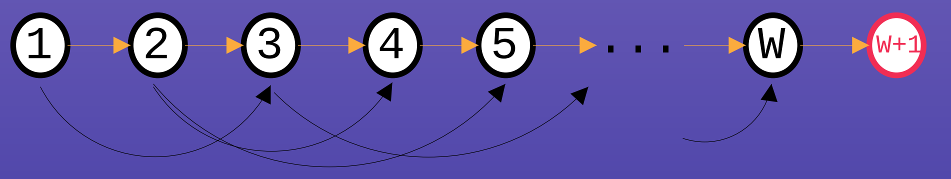
The text can be seen as a graph:



Every occurrence is an arc, outgoing from the node correspondent to its first covered character and entering the node correspondent to the character after the last covered character, and having as weight the number of covered characters.

Subproblem MPP

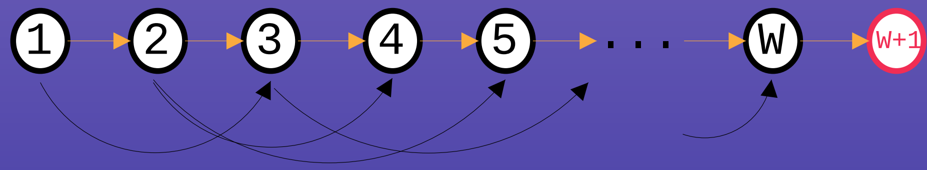
The text can be seen as a graph:



Add one arc from every node the next one (except the last one).

Subproblem MPP

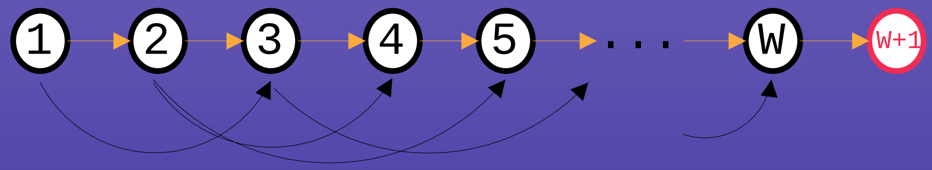
The text can be seen as a graph:



The solution of a Max Path Problem, from the node 1 to the node $|W| + 1$, corresponds to the maximum cover of the text.

Subproblem MPP

The text can be seen as a graph:



The solution of a Max Path Problem, from the node 1 to the node $|W| + 1$, corresponds to the maximum cover of the text.

This graph is acyclic and directed and the optimal solution can be found in polynomial time.

CPLEX 6.5

Three equivalent formulations have been solved with CPLEX:

CPLEX 6.5

Three equivalent formulations have been solved with CPLEX:

Relaxing 1:

$$\text{TCSS}_1) \qquad \max z = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i$$

$$\text{s.t.} \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j - y_{u(j)} \leqslant 0 & j = 1 \dots |O| \\ x_j \in \{0,1\} & j = 1 \dots |O| \\ y_i \in [0,1] & i = 1 \dots |S| \end{cases}$$

CPLEX 6.5

Three equivalent formulations have been solved with CPLEX:

Relaxing 2:

$$\text{TCSS}_2) \qquad \max z = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i$$

$$\text{s.t.} \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j - y_{u(j)} \leqslant 0 & j = 1 \dots |O| \\ x_j \in [0,1] & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

CPLEX 6.5

Three equivalent formulations have been solved with CPLEX:

Relaxing 3:

$$\text{TCSS}_3) \qquad \max z = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i$$

$$\textit{s.t.} \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \ldots |W| \\ \sum_{j \in O_S(i)} x_j - y_i |O_S(i)| \leqslant 0 & i = 1 \ldots |S| \\ x_j \in [0,1] & j = 1 \ldots |O| \\ y_i \in \{0,1\} & i = 1 \ldots |S| \end{cases}$$

Lagrangean relaxation

The TCSS problem has two distinct sets of constraints that can be dualized, and therefore we studied two different Lagrangean relaxations:

Lagrangean relaxation

The TCSS problem has two distinct sets of constraints that can be dualized, and therefore we studied two different Lagrangean relaxations:

- relaxation of packing constraints

Lagrangean relaxation

The TCSS problem has two distinct sets of constraints that can be dualized, and therefore we studied two different Lagrangean relaxations:

- relaxation of packing constraints
- relaxation of variable upper bound constraints

Lagrangean relaxation

The TCSS problem has two distinct sets of constraints that can be dualized, and therefore we studied two different Lagrangean relaxations:

- relaxation of packing constraints
- relaxation of variable upper bound constraints

Both of them were used to compute dual bounds in a branch and bound framework, solving in an approximate way the Lagrangean dual with Subgradient Optimization.

Relaxation of Packing constraints

Relaxing the Packing constraints we obtain:

Relaxation of Packing constraints

Relaxing the Packing constraints we obtain:

LR1)
$$\max z_{LR1}(\lambda) = \sum_{j=1}^{|O|} \left(\alpha l_{u(j)} - \sum_{t=1}^{|W|} \lambda_t a_{tj} \right) x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i + \sum_{t=1}^{|W|} \lambda_t$$

$$\text{s.t.} \begin{cases} x_j - y_{u(j)} \leqslant 0 & j = 1 \dots |O| \\ x_j \in \{0,1\} & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

Relaxation of Packing constraints

Relaxing the Packing constraints we obtain:

LR1)
$$\max z_{LR1}(\lambda) = \sum_{j=1}^{|O|} \left(\alpha l_{u(j)} - \sum_{t=1}^{|W|} \lambda_t a_{tj} \right) x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i + \sum_{t=1}^{|W|} \lambda_t$$

$$\text{s.t.} \begin{cases} x_j - y_{u(j)} \leqslant 0 & j = 1 \dots |O| \\ x_j \in \{0,1\} & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

For any choice of λ the term $\sum_{t=1}^{|W|} \lambda_t$ is constant, and this problem can be decomposed into $|S|$ problems.

Relaxation of Packing constraints

Relaxing the Packing constraints we obtain:

LR1) $$\max z_{LR1}(\lambda) = \sum_{j=1}^{|O|} \left(\alpha l_{u(j)} - \sum_{t=1}^{|W|} \lambda_t a_{tj} \right) x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i + \sum_{t=1}^{|W|} \lambda_t$$

$$\text{s.t.} \begin{cases} x_j - y_{u(j)} \leqslant 0 & j = 1 \dots |O| \\ x_j \in \{0,1\} & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

Let c_j be the coefficient of x_j. Every problem is solved fixing:

$$y_i = \begin{cases} 1 & \text{if } \sum_{j \in O_S(i) | c_j > 0} c_j - \frac{1-\alpha}{l_i} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Branching strategy

Given the optimal solution (x^*, y^*), define:

Branching strategy

Given the optimal solution (x^*, y^*), define:

$$q_t = \sum_{j=1}^{|O|} a_{tj} x_j^* \quad \text{and} \quad V_t = \left\{ j | a_{tj} x_j^* = 1 \right\} = \{v(t)_1, \dots, v(t)_{q_t}\}$$

Branching strategy

Given the optimal solution (x^*, y^*), define:

$$q_t = \sum_{j=1}^{|O|} a_{tj} x_j^* \quad \text{and} \quad V_t = \left\{ j | a_{tj} x_j^* = 1 \right\} = \{v(t)_1, \ldots, v(t)_{q_t}\}$$

Let be $k = \arg\max_{t \in W} \{q_t\}$. The tested branching strategies are:

Branching strategy

Given the optimal solution (x^*, y^*), define:

$$q_t = \sum_{j=1}^{|O|} a_{tj} x_j^* \quad \text{and} \quad V_t = \left\{ j | a_{tj} x_j^* = 1 \right\} = \{v(t)_1, \ldots, v(t)_{q_t}\}$$

Let be $k = \arg\max_{t \in W} \{q_t\}$. The tested branching strategies are:

- *on characters*. Generate $q_k + 1$ nodes; in node $r = 1, \ldots, k$ fix $v_r \leftarrow 1$, and in node $q_k + 1$ fix to 0 all occurrences in V_k.

Branching strategy

Given the optimal solution (x^*, y^*), define:

$$q_t = \sum_{j=1}^{|O|} a_{tj} x_j^* \quad \text{and} \quad V_t = \left\{ j | a_{tj} x_j^* = 1 \right\} = \{v(t)_1, \ldots, v(t)_{q_t}\}$$

Let be $k = \arg\max_{t \in W} \{q_t\}$. The tested branching strategies are:

- *on characters*. Generate $q_k + 1$ nodes; in node $r = 1, \ldots, k$ fix $v_r \leftarrow 1$, and in node $q_k + 1$ fix to 0 all occurrences in V_k.
- *on occurences*. Select $\bar{\jmath} = \arg\max_{j \in V_k} \{c_j\}$. Generate two nodes, fixing respectively $x_{\bar{\jmath}} = 1$ and $x_{\bar{\jmath}} = 0$.

Branching strategy

Given the optimal solution (x^*, y^*), define:

$$q_t = \sum_{j=1}^{|O|} a_{tj} x_j^* \quad \text{and} \quad V_t = \left\{ j | a_{tj} x_j^* = 1 \right\} = \{v(t)_1, \ldots, v(t)_{q_t}\}$$

Let be $k = \arg\max_{t \in W} \{q_t\}$. The tested branching strategies are:

- *on characters*. Generate $q_k + 1$ nodes; in node $r = 1, \ldots, k$ fix $v_r \leftarrow 1$, and in node $q_k + 1$ fix to 0 all occurrences in V_k.
- *on occurences*. Select $\bar{j} = \arg\max_{j \in V_k} \{c_j\}$. Generate two nodes, fixing respectively $x_{\bar{j}} = 1$ and $x_{\bar{j}} = 0$.
- *on strings*. Select the string $\bar{\imath} = u(\bar{j})$ and generate two nodes fixing respectively $y_{\bar{\imath}} = 1$ and $y_{\bar{\imath}} = 0$.

Lagrangean heuristic

The optimal solution (x^*, y^*) of LR1 can violate some of the packing constraints constraints.

Lagrangean heuristic

The optimal solution (x^*, y^*) of LR1 can violate some of the packing constraints constraints.

It is possible to obtain a feasible solution by solving an MPP subproblem keeping the values of the y^* fixed:

$$\text{MPP)} \qquad \max z = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j$$

$$\textit{s.t.} \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j \leqslant y^*_{u(j)} & j = 1 \dots |O| \\ x_j \in \{0,1\} & j = 1 \dots |O| \end{cases}$$

Relaxation of vub constraints

Relaxing the variable upper bound constraints we obtain:

LR2) $$\max z_{LR2}(\mu) = \sum_{j=1}^{|O|} \left(\alpha l_{u(j)} - \mu_j\right) x_j + \sum_{i=1}^{|S|} \left(-\frac{1-\alpha}{l_i} + \sum_{j \in O_S(i)} \mu_j \right) y_i$$

$$s.t. \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j \in \{0,1\} & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

Relaxation of vub constraints

Relaxing the variable upper bound constraints we obtain:

LR2) $$\max z_{LR2}(\mu) = \sum_{j=1}^{|O|} \left(\alpha l_{u(j)} - \mu_j\right) x_j + \sum_{i=1}^{|S|} \left(-\frac{1-\alpha}{l_i} + \sum_{j \in O_S(i)} \mu_j \right) y_i$$

$$\text{s.t.} \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \ldots |W| \\ x_j \in \{0,1\} & j = 1 \ldots |O| \\ y_i \in \{0,1\} & i = 1 \ldots |S| \end{cases}$$

that can be decomposed into two independent problems:

Relaxation of vub constraints

Relaxing the variable upper bound constraints we obtain:

LR2) $$\max z_{LR2}(\mu) = \sum_{j=1}^{|O|} \left(\alpha l_{u(j)} - \mu_j\right) x_j + \sum_{i=1}^{|S|} \left(-\frac{1-\alpha}{l_i} + \sum_{j \in O_S(i)} \mu_j \right) y_i$$

$$s.t. \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \ldots |W| \\ x_j \in \{0,1\} & j = 1 \ldots |O| \\ y_i \in \{0,1\} & i = 1 \ldots |S| \end{cases}$$

that can be decomposed into two independent problems:

- LR2y is a trivial problem

Relaxation of vub constraints

Relaxing the variable upper bound constraints we obtain:

LR2)

$$\max z_{LR2}(\mu) = \sum_{j=1}^{|O|} (\alpha l_{u(j)} - \mu_j) x_j + \sum_{i=1}^{|S|} \left(-\frac{1-\alpha}{l_i} + \sum_{j \in O_S(i)} \mu_j \right) y_i$$

$$s.t. \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \ldots |W| \\ x_j \in \{0,1\} & j = 1 \ldots |O| \\ y_i \in \{0,1\} & i = 1 \ldots |S| \end{cases}$$

that can be decomposed into two independent problems:

- LR2y is a trivial problem
- LR2x is an MPP instance whose coefficients depend on μ values

Branching strategy

Let (x^*, y^*) be the solution of LR2(μ) and

$$V(i) = \left\{ j \mid x_j^* > y_i^*, \quad u(j) = i \right\}.$$

Branching strategy

Let (x^*, y^*) be the solution of LR2(μ) and $V(i) = \left\{ j \mid x^*_j > y^*_i, \quad u(j) = i \right\}$. Fixing a variable y_i:

Branching strategy

Let (x^*, y^*) be the solution of LR2(μ) and $V(i) = \left\{ j \mid x_j^* > y_i^*, \quad u(j) = i \right\}$. Fixing a variable y_i:

- if y_i would be fixed to 0 then the occurrences should be fixed to 0 and $z_{LR2}(\mu)$ would decrease by $\sigma_i^0 = \sum_{j \in V(i)} (\alpha l_i - \mu_j)$

Branching strategy

Let (x^*, y^*) be the solution of LR2(μ) and $V(i) = \left\{ j \mid x^*_j > y^*_i, \quad u(j) = i \right\}$. Fixing a variable y_i:

- if y_i would be fixed to 0 then the occurrences should be fixed to 0 and $z_{LR2}(\mu)$ would decrease by $\sigma^0_i = \sum_{j \in V(i)} (\alpha l_i - \mu_j)$
- if y_i would be fixed to 1 then the string should be paid and $z_{LR2}(\mu)$ would decrease by $\sigma^1_i = \frac{1-\alpha}{l_i} - \sum_{j \in O_S(i)} (\mu_j)$

Branching strategy

Let (x^*, y^*) be the solution of LR2(μ) and $V(i) = \left\{ j \mid x_j^* > y_i^*, \quad u(j) = i \right\}$. Fixing a variable y_i:

- if y_i would be fixed to 0 then the occurrences should be fixed to 0 and $z_{LR2}(\mu)$ would decrease by $\sigma_i^0 = \sum_{j \in V(i)} (\alpha l_i - \mu_j)$
- if y_i would be fixed to 1 then the string should be paid and $z_{LR2}(\mu)$ would decrease by $\sigma_i^1 = \frac{1-\alpha}{l_i} - \sum_{j \in O_S(i)} (\mu_j)$

The branching variable is the one that maximizes the minimum decrement of $z_{LR2}^*(\mu)$:

$$i = \arg\max_k \left\{ \min \left\{ \sigma_k^0, \sigma_k^1 \right\} \right\}$$

Lagrangean heuristic

The optimal solution (x^*, y^*) of LR2 can violate some variable upper bound constraints. To obtain a feasible solution it is necessary to flip some variables.

Lagrangean heuristic

The optimal solution (x^*, y^*) of LR2 can violate some variable upper bound constraints. To obtain a feasible solution it is necessary to flip some variables.

One way is to evaluate the costs of strings and occurrences and fix:

$$y_i = \begin{cases} 1 & \text{if } \sum_{j \in O_S(i)} \alpha l_i x_j^* > \frac{(1-\alpha)}{l_i} \\ 0 & \text{otherwise, fixing to } 0 \text{ also the occurrences} \end{cases}$$

Lagrangean heuristic

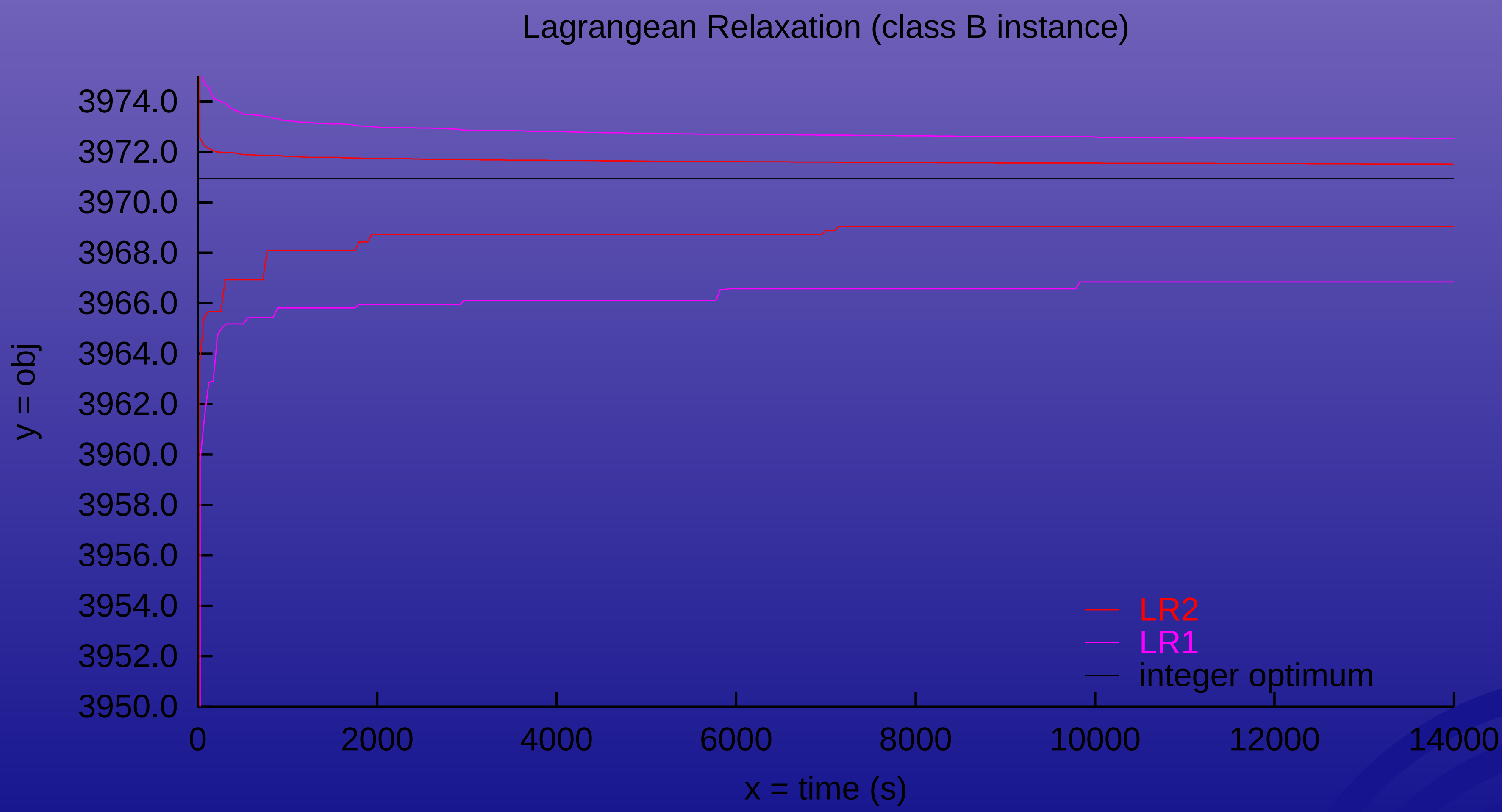
The optimal solution (x^*, y^*) of LR2 can violate some variable upper bound constraints. To obtain a feasible solution it is necessary to flip some variables.

One way is to evaluate the costs of strings and occurrences and fix:

$$y_i = \begin{cases} 1 & \text{if } \sum_{j \in O_S(i)} \alpha l_i x_j^* > \frac{(1-\alpha)}{l_i} \\ 0 & \text{otherwise, fixing to } 0 \text{ also the occurrences} \end{cases}$$

A more sophisticated heuristic consists of solving an MPP problem on a graph containing only the arcs corresponding to the occurrences of the strings such that $y_i^* = 1$.

Comparison of the LR1 and LR2



Local search heuristics

A local search algorithm moves from s_t to $s_{t+1} \in N(s_t)$.

Local search heuristics

A local search algorithm moves from s_t to $s_{t+1} \in N(s_t)$.

Let $z(s)$ be the value of a solution s.

Local search heuristics

A local search algorithm moves from s_t to $s_{t+1} \in N(s_t)$.

Let $z(s)$ be the value of a solution s.

There are at least three strategies for the choice of $s_{t+1} \in N(s_t)$:

Local search heuristics

A local search algorithm moves from s_t to $s_{t+1} \in N(s_t)$.

Let $z(s)$ be the value of a solution s.

There are at least three strategies for the choice of $s_{t+1} \in N(s_t)$:

- *best improve*. Find $n^* = \arg \max_{n \in N(s_t)} z(n)$.

Local search heuristics

A local search algorithm moves from s_t to $s_{t+1} \in N(s_t)$.

Let $z(s)$ be the value of a solution s.

There are at least three strategies for the choice of $s_{t+1} \in N(s_t)$:

- *best improve*. Find $n^* = \arg \max\limits_{n \in N(s_t)} z(n)$.
- *first improve*. Select $n_i \in N(s_t)$. If not improving select another neighbor n_{i+1}.

Local search heuristics

A local search algorithm moves from s_t to $s_{t+1} \in N(s_t)$.

Let $z(s)$ be the value of a solution s.

There are at least three strategies for the choice of $s_{t+1} \in N(s_t)$:

- *best improve*. Find $n^* = \arg \max_{n \in N(s_t)} z(n)$.
- *first improve*. Select $n_i \in N(s_t)$. If not improving select another neighbor n_{i+1}.
- hybrid strategies. Find $n_i^* = \arg \max_{n \in \tilde{N}_i(s_t) \subseteq N(s_t)} z(n)$.

  If not improving consider another subset $\tilde{N}_{i+1} \subseteq N(s_t)$.

Application to TCSS

To apply a local search method to TCSS we defined:

Application to TCSS

To apply a local search method to TCSS we defined:

- *Solutions space.* A solution s consists of the values y^s of all string variables. A complete solution is obtained solving a correspondent MPP problem over these values.

Application to TCSS

To apply a local search method to TCSS we defined:

- *Solutions space.* A solution s consists of the values y^s of all string variables. A complete solution is obtained solving a correspondent MPP problem over these values.
- *Neighborhood.* $N(s)$ is the set of solutions that differ from s by the value of exactly one variable.

Application to TCSS

To apply a local search method to TCSS we defined:

- *Solutions space*. A solution s consists of the values y^s of all string variables. A complete solution is obtained solving a correspondent MPP problem over these values.
- *Neighborhood*. $N(s)$ is the set of solutions that differ from s by the value of exactly one variable.
- *Subneighborhoods*. $|N(s)| = |S|$ and the evaluation is heavy, so the search strategy is a hybrid one. $N(s)$ is explored by sets of NBR elements in two ways:
  - by a ciclyc predefined order.
  - extracting NBR elements randomly.

Application to TCSS

To apply a local search method to TCSS we defined:

- *Solutions space*. A solution s consists of the values y^s of all string variables. A complete solution is obtained solving a correspondent MPP problem over these values.
- *Neighborhood*. $N(s)$ is the set of solutions that differ from s by the value of exactly one variable.
- *Subneighborhoods*. $|N(s)| = |S|$ and the evaluation is heavy, so the search strategy is a hybrid one. $N(s)$ is explored by sets of NBR elements in two ways:
  - by a ciclyc predefined order.
  - extracting NBR elements randomly.
- *Starting solution*. The starting solution is $y_i^{s_0} = 1 \quad \forall i$. Also an initialization to zero and a random initialization have been tested.

Threshold accepting

In this method also some worsening solution, those under a certain threshold, can be accepted. Let:

Threshold accepting

In this method also some worsening solution, those under a certain threshold, can be accepted. Let:

- k_t be the threshold at iteration t

Threshold accepting

In this method also some worsening solution, those under a certain threshold, can be accepted. Let:

- k_t be the threshold at iteration t
- n be the neighbor selected by the chosen strategy

Threshold accepting

In this method also some worsening solution, those under a certain threshold, can be accepted. Let:

- k_t be the threshold at iteration t
- n be the neighbor selected by the chosen strategy
- c_t be the benchmark solution at time t

Threshold accepting

In this method also some worsening solution, those under a certain threshold, can be accepted. Let:

- k_t be the threshold at iteration t
- n be the neighbor selected by the chosen strategy
- c_t be the benchmark solution at time t

If $z(n) + k_t > z(c_t)$ then $s_{t+1} := n$.

Threshold accepting

In this method also some worsening solution, those under a certain threshold, can be accepted. Let:

- k_t be the threshold at iteration t
- n be the neighbor selected by the chosen strategy
- c_t be the benchmark solution at time t

If $z(n) + k_t > z(c_t)$ then $s_{t+1} := n$.

The threshold value at iteration t is set to $k_t = \frac{(1-\alpha)}{1+t \textbf{ div } |S|}$

Threshold accepting

In this method also some worsening solution, those under a certain threshold, can be accepted. Let:

- k_t be the threshold at iteration t
- n be the neighbor selected by the chosen strategy
- c_t be the benchmark solution at time t

If $z(n) + k_t > z(c_t)$ then $s_{t+1} := n$.

The threshold value at iteration t is set to $k_t = \frac{(1-\alpha)}{1+t \textbf{ div } |S|}$

Two versions of the acceptance test have been tested:

Threshold accepting

In this method also some worsening solution, those under a certain threshold, can be accepted. Let:

- k_t be the threshold at iteration t
- n be the neighbor selected by the chosen strategy
- c_t be the benchmark solution at time t

If $z(n) + k_t > z(c_t)$ then $s_{t+1} := n$.

The threshold value at iteration t is set to $k_t = \frac{(1-\alpha)}{1+t \textbf{ div } |S|}$

Two versions of the acceptance test have been tested:

- compare n with the current solution ($c_t = s_t$).

Threshold accepting

In this method also some worsening solution, those under a certain threshold, can be accepted. Let:

- k_t be the threshold at iteration t
- n be the neighbor selected by the chosen strategy
- c_t be the benchmark solution at time t

If $z(n) + k_t > z(c_t)$ then $s_{t+1} := n$.

The threshold value at iteration t is set to $k_t = \frac{(1-\alpha)}{1+t \textbf{ div } |S|}$

Two versions of the acceptance test have been tested:

- compare n with the current solution ($c_t = s_t$).
- compare n with the best solution found ($c_t = \max\limits_{\tau \leqslant t}(s_\tau)$).

Simulated annealing

The acceptance of a worsening solution is probabilistic:

$p(n, s, T) = e^{\frac{z(n)-z(s)}{T}}$, where T is a parameter, the *temperature*.

Simulated annealing

The acceptance of a worsening solution is probabilistic:
$p(n, s, T) = e^{\frac{z(n)-z(s)}{T}}$, where T is a parameter, the *temperature*.

The cooling schedule used is $T_t = T_{start} \cdot (T_{cool})^t \quad (T_{cool}<1)$.

Simulated annealing

The acceptance of a worsening solution is probabilistic:
$p(n, s, T) = e^{\frac{z(n)-z(s)}{T}}$, where T is a parameter, the *temperature*.

The cooling schedule used is $T_t = T_{start} \cdot (T_{cool})^t \quad (T_{cool}<1)$.

These two parameters have been made input dependent:

$$T_{start} = \frac{-L\alpha}{\ln(0.5)} \qquad T_{cool} = \sqrt[|S|]{\frac{\ln(0.5)}{L \ln(0.01)}}$$

Simulated annealing

The acceptance of a worsening solution is probabilistic:
$p(n, s, T) = e^{\frac{z(n)-z(s)}{T}}$, where T is a parameter, the *temperature*.

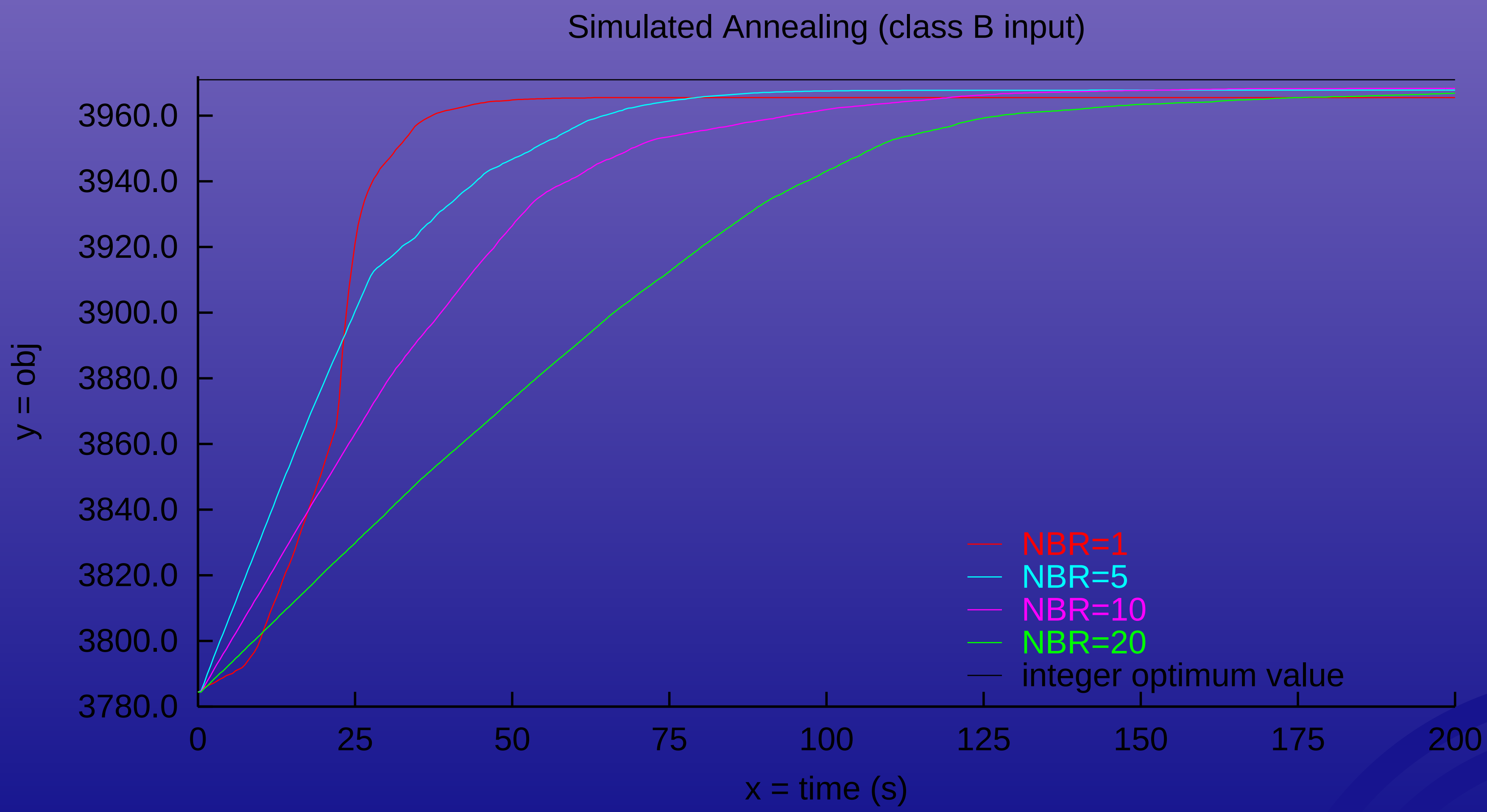
The cooling schedule used is $T_t = T_{start} \cdot (T_{cool})^t \quad (T_{cool} < 1)$.

These two parameters have been made input dependent:

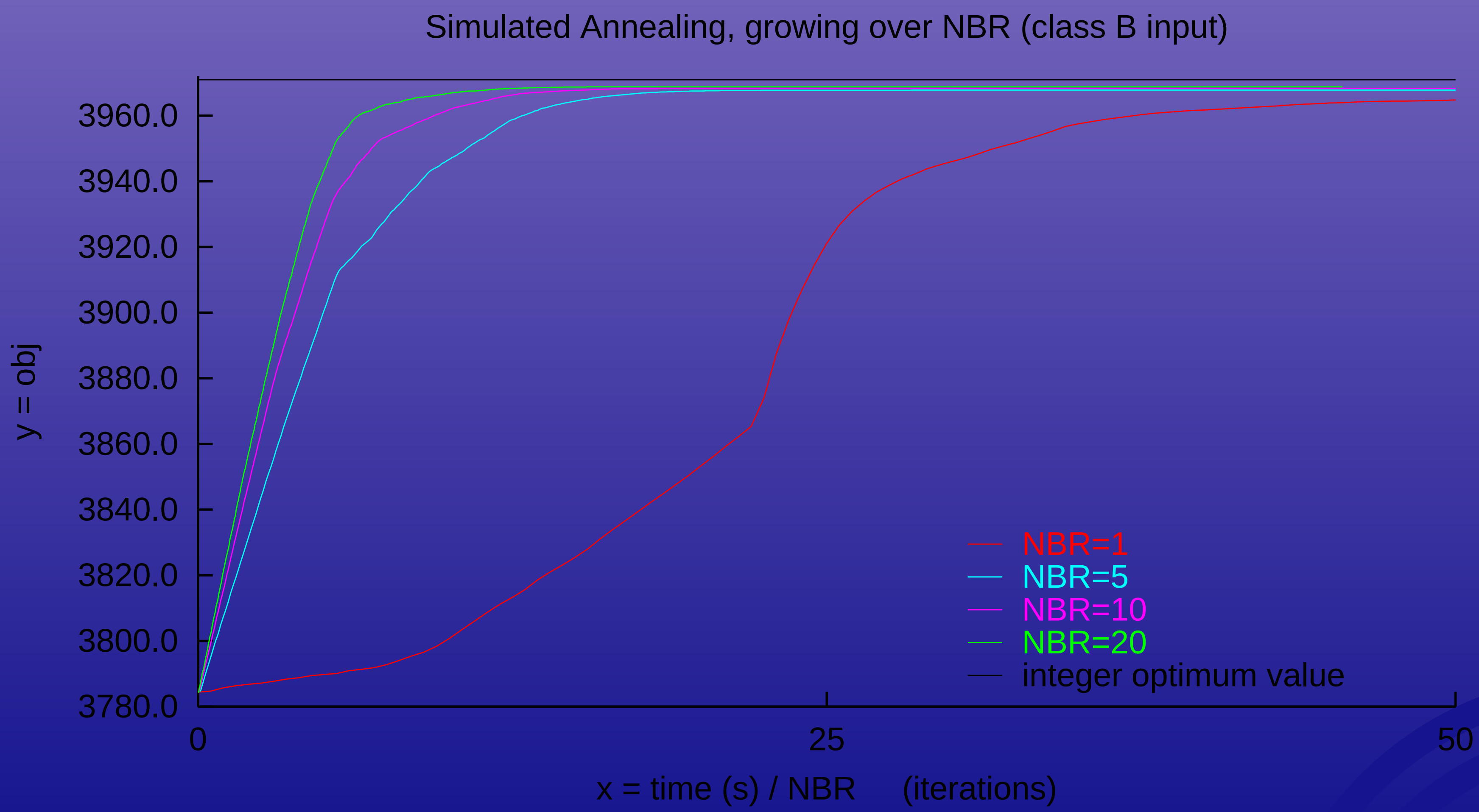
$$T_{start} = \frac{-L\alpha}{\ln(0.5)} \qquad T_{cool} = \sqrt[|S|]{\frac{\ln(0.5)}{L\ln(0.01)}}$$

When the selected neighbor is not accepted the subneighborhood enlarges itself. In that way every time the test is made against the most probable neighbor.

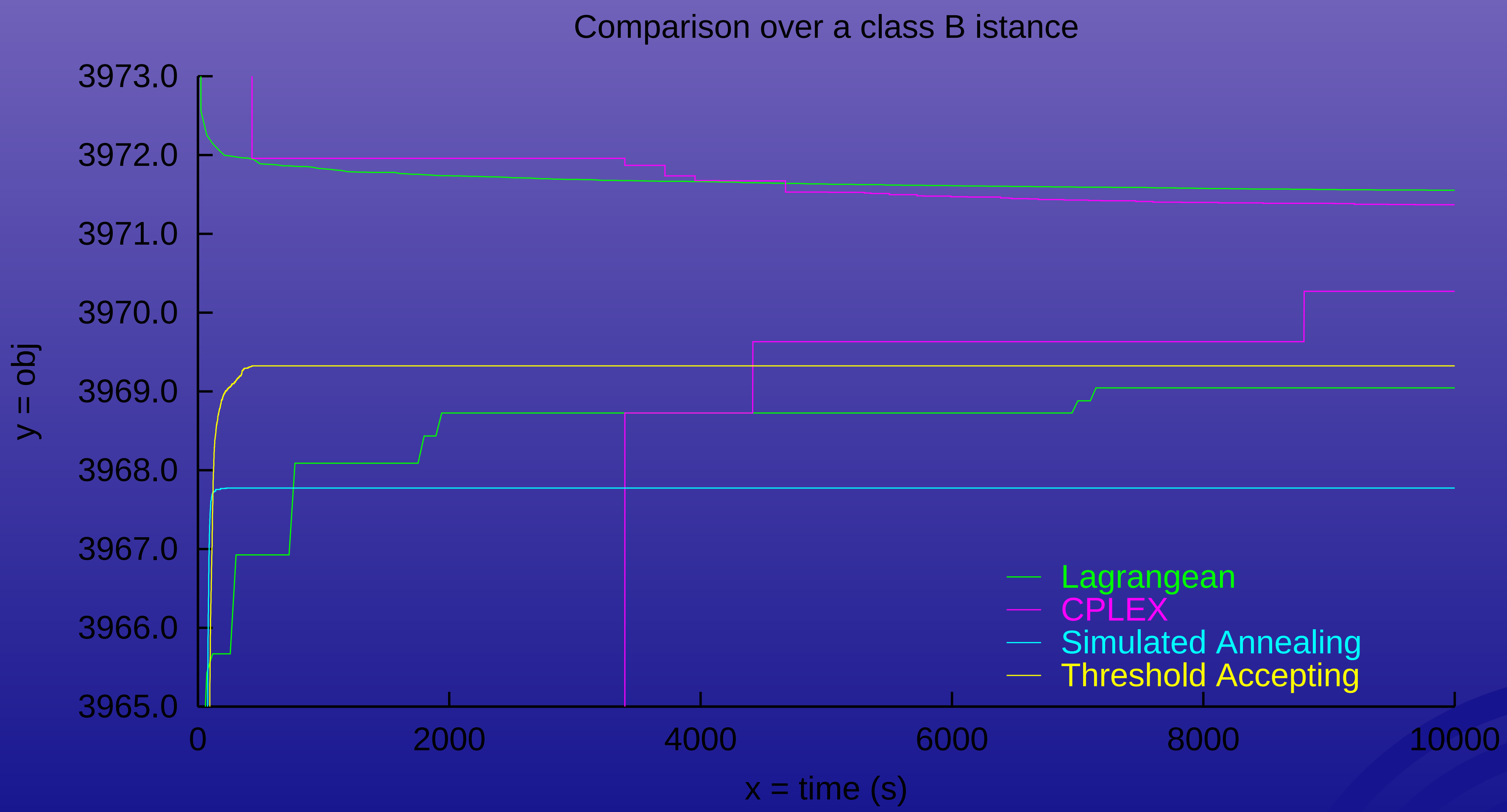
Comparisons



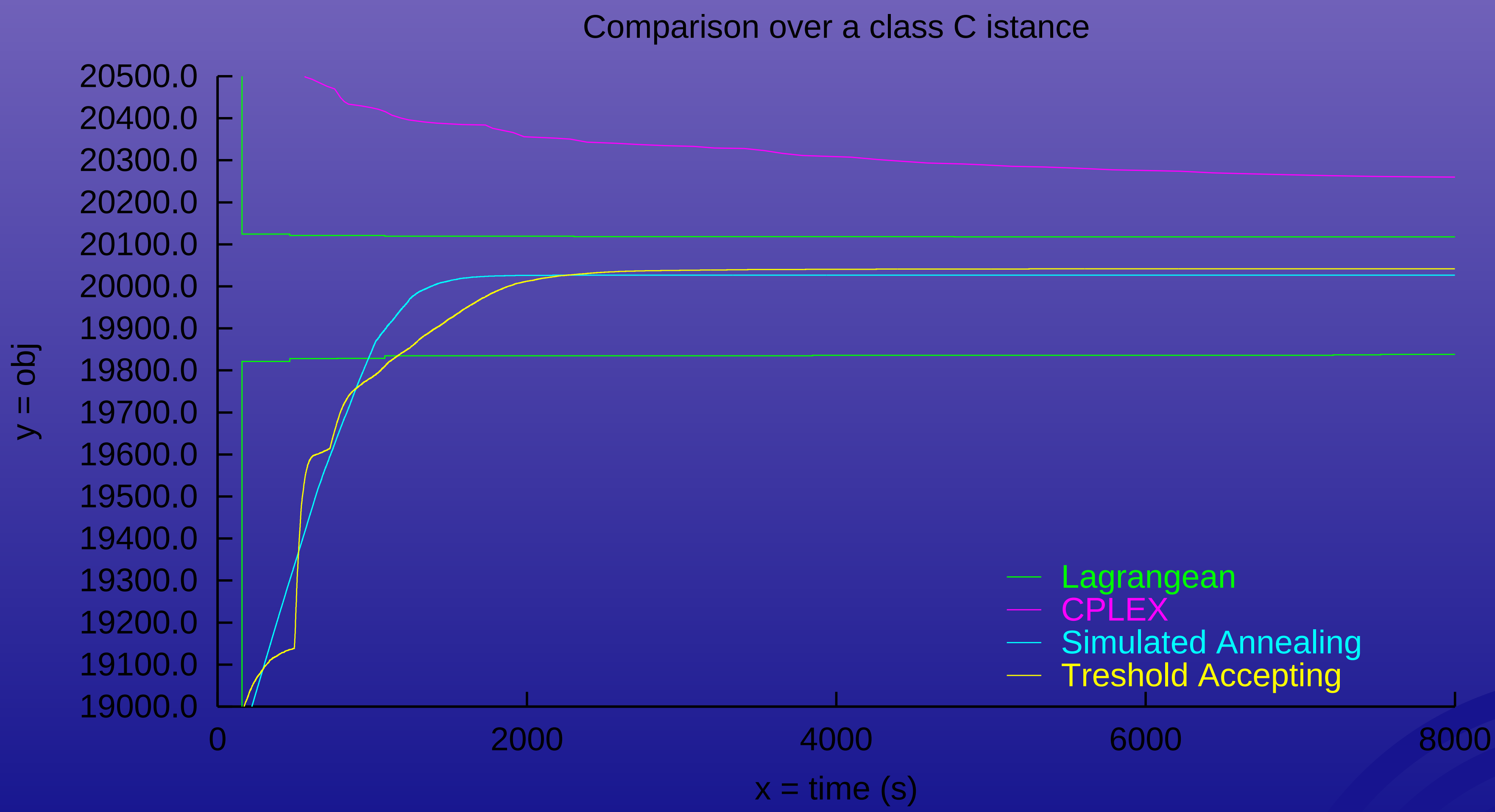
Comparisons



Comparisons of all the methods



Comparisons of all the methods



Comparisons of all the methods