



Single Sink with non-simultaneous Demand

Joël Clivaz*

January 2010

Abstract

We know that for the Steiner Tree Problem, there exist some properties that lead to a theorem proving the existence of a 2-approximation algorithm. We are interested in the Single Sink with non-simultaneous Demand problem. In this case, the Steiner Tree's properties do not hold anymore, but we do not know if the theorem is still true and if a 2-approximation algorithm is still possible. In this project, we will try to find a counter-example of the theorem which would imply that we cannot find a 2-approximation algorithm by simple extending the technique used for the Steiner Tree Problem.

*assisted by Laura Sanità

Contents

1	Introduction	3
2	Description of the problem	3
3	The Steiner Tree Problem	3
3.1	Definition of the Steiner Tree Problem	3
3.2	Link with our problem	4
3.3	Important properties of the Steiner Tree Problem	4
3.4	Weak supermodularity of the function f in the Steiner Tree Problem	5
3.5	First possibility: $f(A) = f(B) = 1$	5
3.5.1	First case	6
3.5.2	Second case	6
3.5.3	Third case	6
3.6	Second possibility: $f(A) = 1, f(B) = 0$	7
3.7	Last possibility: $f(A) = f(B) = 0$	8
4	Back to the Single Sink with non-simultaneous Demand	8
4.1	Weak supermodularity	8
4.2	Laminarity	9
5	Computational search for an interesting point	10
5.1	First step: Complete graph with 6 nodes	10
5.1.1	Possibilities of subsets of nodes	10
5.1.2	Test of the different possibilities	11
5.2	Results for the case with 6 nodes	11
5.3	Next step: Complete graph with 11 nodes	11
5.4	Results for the case with 11 nodes	12
A	C++ code	13
A.1	Program to find all the possibilities of subsets of 5 nodes	13
A.2	Program to create the input file for the program cdd+	21
A.3	Program to test if some points verify $x_e < 1/2, \forall e \in E$	31
A.4	Program to test if an interesting point is a vertex	32
B	Possibilities of subsets R^i with a graph with 6 nodes	32
C	References	34

1 Introduction

It is known, following some properties, that it exists a 2-approximation algorithm for the Steiner Tree Problem. We want to know here if the technique used in the case of the Steiner Tree Problem could be applied in the case of the Single Sink with non-simultaneous Demand to find a 2-approximation algorithm. More precisely, we will try to find a counter-example in the case of the Single Sink problem, which will prove that the technique used in the case of the Steiner Tree Problem does not work in this case.

First of all, we will study the link between the Steiner Tree Problem and the Single Sink with non-simultaneous Demand Problem. Then, we will study the properties that lead, in the case of the Steiner Tree Problem, to a theorem which proves the existence of a 2-approximation algorithm. After that, we will show that these properties do not hold in the case of the Single Sink with non-simultaneous Demand Problem. The last step is the explanation of the experimental way that we have followed to try to find some counter-example of the theorem which leads to the proof of the existence of a 2-approximation algorithm in the case of the Steiner Tree Problem.

2 Description of the problem

In the case of the Single Sink with non-simultaneous Demand Problem, we have given an undirected graph $G(V, E)$, a cost function $c : E \rightarrow \mathbb{R}_+$, a node $r \in V$ defined as the root node and k sets of terminal nodes, R^1, \dots, R^k with $R^i \subseteq V \setminus \{r\}$, $i = 1, \dots, k$. Each node in R^i has a unit demand. If two nodes belong to two different subsets R^i and R^j with $i \neq j$, their demand is not simultaneous.

We want to install a min cost-capacity x_e on each edge e of this graph, such that, for all $i = 1, \dots, k$, the capacity allows for routing the flow from R^i to the root node r .

The cut formulation for this problem is:

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e \\ & \sum_{e \in \delta(S)} x_e \geq f(S) \quad \forall S \subseteq V \\ & x_e \in \mathbb{Z}_+ \quad \forall e \in E, \end{aligned} \tag{1}$$

where $\delta(S)$ is the set of edges with exactly one endpoint in S , S is a cut and $f(S) = \max_{i=1, \dots, k} |S \cap R^i|$ if $r \notin S$, $f(S) = 0$ otherwise.

This problem is a generalization of the Steiner Tree Problem. For this latter, we know that for each extreme point of the polyhedron described by the linear system of the cut formulation (after relaxing integrality constraints), we have $x_e \geq 1/2$ for at least one $e \in E$.

The aim of this project is to find a counter example of this property for the Single Sink with non-simultaneous Demand Problem. In other words, we want to find an extreme point of the polyhedron described by the linear system (1) with non-negativity constraints, such that $x_e < 1/2$ for all $e \in E$.

3 The Steiner Tree Problem

3.1 Definition of the Steiner Tree Problem

Given an undirected graph $G(V, E)$, where V is the set of nodes and E is the set of edges, a cost function $c : E \rightarrow \mathbb{Q}_+$ and a set $T = \{r_1, \dots, r_k, r_{k+1}\} \in V$, which elements are called *terminals*, we want to find a tree with minimal cost which binds all the terminals.

This problem corresponds to the following linear system:

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e \\ & \sum_{e \in \delta(S)} x_e \geq f(S) \quad \forall S \subseteq V \\ & x_e \in \mathbb{Z}_+, \quad \forall e \in E, \end{aligned} \tag{2}$$

where

$$f(S) = \begin{cases} 1 & \text{if } (S \cap T) \neq \emptyset, (\bar{S} \cap T) \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

The second equation in the system (2) makes that all the terminals are connected.

3.2 Link with our problem

Given an undirected graph $G(V, E)$ and a set of terminals as in the definition of the Steiner Tree Problem, we can define the node r_{k+1} as the root node in the Single Sink with non-simultaneous Demand Problem. We also can build k subsets of V with only one element per subset as it follows: for each terminal r_i , $i = 1, \dots, k$, we define $R^i = \{r_i\}$.

In that case, the Single Sink with non-simultaneous Demand and the Steiner Tree Problems are equivalent. We can see that by comparing the linear systems corresponding to each problem. In that case, indeed, the value of the function f in the Single Sink Problem will be 1 if $(S \cap R^i) \neq \emptyset$ for at least one i and 0 otherwise, and we can see that the two linear systems will be the same.

We can also see the equivalence of the two problems without using the linear systems. Indeed, if we take the Single Sink with non-simultaneous Demand Problem with the function f always equal to 1 or 0, as defined above and if we find a minimal capacity on the edges of the graph, this one will correspond to the x_e in the Steiner Tree Problem. Conversely, if we find a minimal tree joining all the terminals for the Steiner Tree Problem, the x_e for the edges e in the tree will correspond to a minimal capacity on this edge in the Single Sink Problem, in the case where f is always equal to 1 or 0. In that case, the non-simultaneity comes from the fact that all the subsets R^i , $i = 1, \dots, k$ have cardinality one.

3.3 Important properties of the Steiner Tree Problem

We study now some important properties of the Steiner Tree Problem. We begin with the central theorem concerning the Steiner Tree Problem in our project.

Theorem 3.1. *For the Steiner Tree Problem, each extreme point of the polyhedron described by the constraints of the LP-relaxation (2) has at least one variable $x_e \geq 1/2$. (Jain, 1998)*

In fact, this result follows from some steps that we will explain here. We begin with the definition of the *weak supermodularity*.

Definition 3.1. *A function $f : V \rightarrow 2^V$ is said to be weakly supermodular if $f(V) = 0$ and for every two sets $A, B \subseteq V$, at least one of the following conditions holds:*

1. $f(A) + f(B) \leq f(A - B) + f(B - A)$,
2. $f(A) + f(B) \leq f(A \cap B) + f(A \cup B)$.

Then, we define the concept of *laminar family*.

Definition 3.2. A collection L of subsets of V forms a laminar family if no two sets in this collection cross.

We can see an example of a crossing family on the figure (1(a)) and an example of a laminar family on the figure (1(b)).

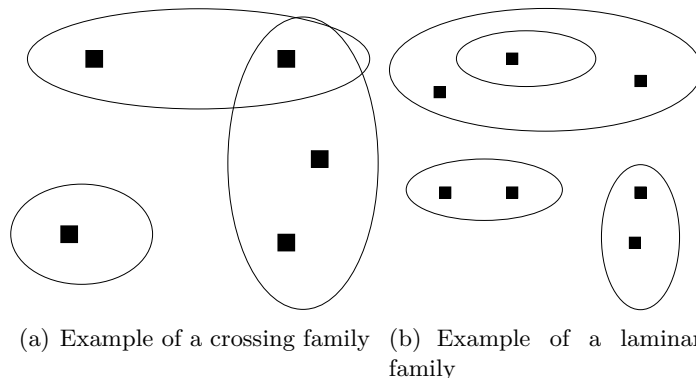


Figure 1: Definiton of a laminar family (examples).

Given a function f , Jain (1998) showed that:

f is weakly supermodular.

↓

The extreme points are defined by tight constraints in the corresponding linear system forming a laminar family.

↓

$x_e \geq 1/2$ for at least one $e \in E$

3.4 Weak supermodularity of the function f in the Steiner Tree Problem

In the case of the Steiner Tree problem, we have that the function f in the linear system (2) is weakly supermodular. In fact, we can take two subsets of nodes A and B . We have that

$$f(A) = \begin{cases} 1 & \text{if } (A \cap T) \neq \emptyset, (\bar{A} \cap T) \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

and

$$f(B) = \begin{cases} 1 & \text{if } (B \cap T) \neq \emptyset, (\bar{B} \cap T) \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

So for the subsets $A \cap B$ and $A \cup B$ we have different cases.

3.5 First possibility: $f(A) = f(B) = 1$

Suppose that $f(A) = f(B) = 1$, so we have

$$A \cap T \neq \emptyset, \overline{A} \cap T \neq \emptyset,$$

$$B \cap T \neq \emptyset, \overline{B} \cap T \neq \emptyset.$$

Then, we have that $(\overline{A \cap B}) = (\overline{A} \cup \overline{B}) \neq \emptyset$ and $(\overline{A} \cup \overline{B}) \cap T \neq \emptyset$.

3.5.1 First case

Suppose that $(A \cap B) \cap T \neq \emptyset$, so

$$(A \cap B) \cap T \neq \emptyset, (\overline{A \cap B}) \cap T \neq \emptyset,$$

and therefore, $f(A \cap B) = 1$. Then we have that $(A \cup B) \neq \emptyset$ and $(A \cup B) \cap T \neq \emptyset$ in this case and suppose that $(\overline{A \cap B}) \cap T = (\overline{A \cup B}) \cap T \neq \emptyset$. So we have that

$$(A \cup B) \cap T \neq \emptyset, (\overline{A \cup B}) \cap T \neq \emptyset,$$

and so $f(A \cup B) = 1$. So in this case, the second condition in the definition of weak supermodularity holds.

3.5.2 Second case

Suppose now that $(A \cap B) \cap T \neq \emptyset$ and that $(\overline{A \cup B}) \cap T = \emptyset$. In this case, we should have $T \subset (A \cup B)$. The only possibility is the one on the figure (2), because we have $\overline{A} \cap T \neq \emptyset$ and $\overline{B} \cap T \neq \emptyset$. Then in this case, we have

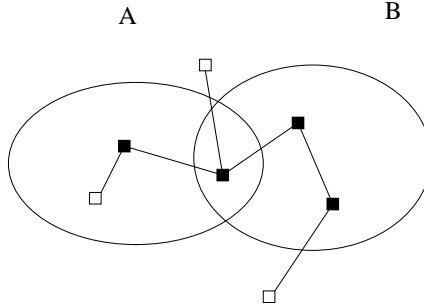


Figure 2: Situation in the second case

$$(A - B) \cap T = (A \cap \overline{B}) \cap T \neq \emptyset, (\overline{A - B}) \cap T = (\overline{A} \cup B) \cap T \neq \emptyset,$$

$$(B - A) \cap T = (B \cap \overline{A}) \cap T \neq \emptyset, (\overline{B - A}) \cap T = (\overline{B} \cup A) \cap T \neq \emptyset,$$

and therefore, $f(A - B) = f(B - A) = 1$ and the first condition in the definition of weak supermodularity holds.

3.5.3 Third case

Suppose now that $(A \cap B) \cap T = \emptyset$. We always have that $(\overline{A \cup B}) \cap T \neq \emptyset$. In this case, we have just the possibilities in the figure (3). We can easily see that the first condition of the definition of weak supermodularity holds here too.

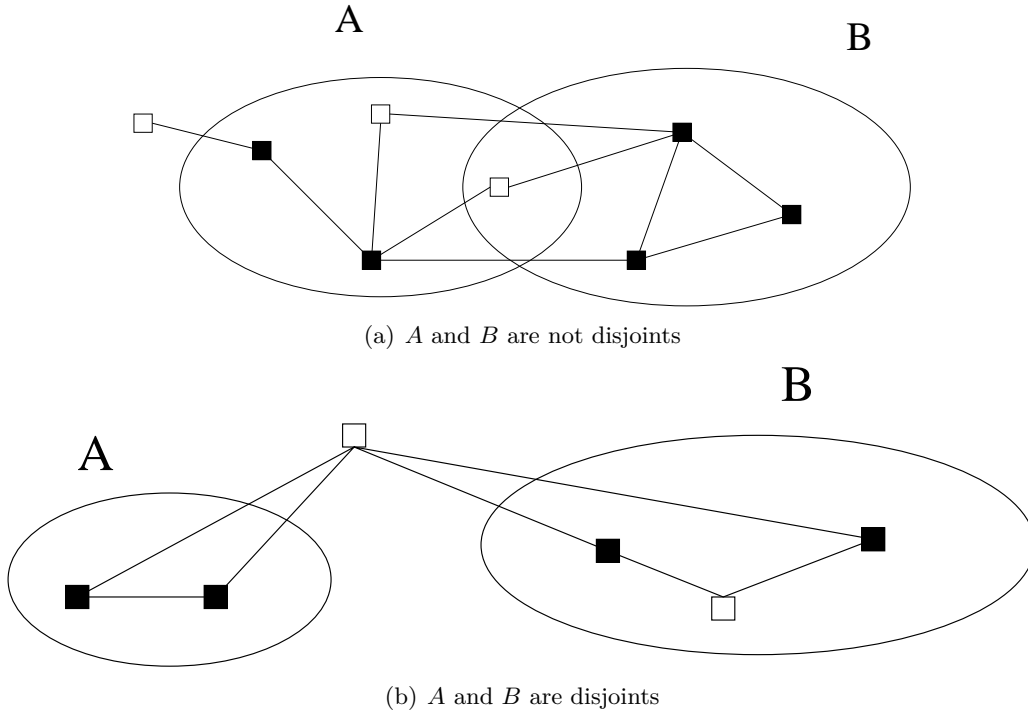


Figure 3: Possibilities in the third case

3.6 Second possibility: $f(A) = 1, f(B) = 0$

For the next cases, we suppose that $f(A) = 1$ and $f(B) = 0$. The cases where $f(B) = 1$ and $f(A) = 0$ is the same. In this case, we have

$$A \cap T \neq \emptyset, \bar{A} \cap T \neq \emptyset,$$

$$B \cap T \neq \emptyset, \bar{B} \cap T = \emptyset.$$

The case where $B' \cap T = \emptyset, \bar{B}' \cap T \neq \emptyset$ is the same (put $B' = \bar{B}$).

$B \cap T \neq \emptyset, \bar{B} \cap T = \emptyset$, so we have that $(A \cap T) \subset B$ and $T \subset B$. This case is represented in the figure (4). In this case, we have that

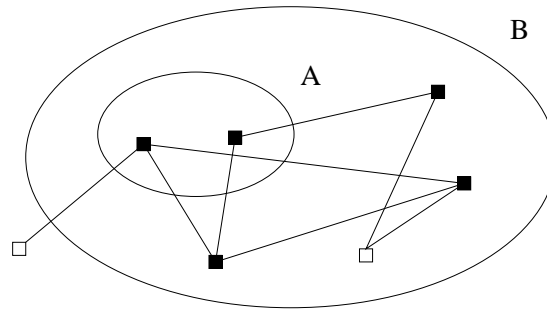


Figure 4: Situation in the fourth case

$$(A - B) \cap T = \emptyset, (\overline{A - B}) \cap T \neq \emptyset,$$

$$(B - A) \cap T \neq \emptyset, (\overline{B - A}) \cap T \neq \emptyset.$$

So, we have that $f(A - B) = 0$ and $f(B - A) = 1$ and the first condition in the definition of weak supermodularity holds.

We also have

$$\begin{aligned} (A \cap B) \cap T &\neq \emptyset, (\overline{A \cap B}) \cap T \neq \emptyset, \\ (A \cup B) \cap T &\neq \emptyset, (\overline{A \cup B}) \cap T = \emptyset. \end{aligned}$$

So, we have that $f(A \cap B) = 1$ and $f(A \cup B) = 0$ and the second condition in the definition of weak supermodularity holds.

3.7 Last possibility: $f(A) = f(B) = 0$

In this case, the two conditions of the definition of weak supermodularity hold directly, because the function f can only take the values 0 or 1.

So we have shown that the function f in the Steiner Tree problem is *weakly supermodular*. We could also prove that for the Steiner Tree problem, the tight constraints associated to extreme points form a laminar family, but we will not do that here. Then, the theorem (3.1) says that $x_e \geq 1/2$ for at least one $e \in E$. This theorem is important because this property leads to a 2-approximation algorithm for the Steiner Tree problem.

4 Back to the Single Sink with non-simultaneous Demand

We now want to see if the theorem (3.1) is still holding for the Single Sink with non-simultaneous Demand Problem.

4.1 Weak supermodularity

First of all, we will show a counter-example, which will prove that the function f in the linear system corresponding to the Single Sink Problem is not weakly supermodular. The figure (5) shows the situation. Let A and B be two cuts. We have here $f(A) = 2$ and $f(B) = 2$. Moreover, we have

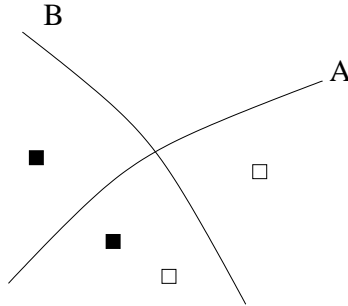


Figure 5: Counter-example for the weak supermodularity in the Single Sink with non-simultaneous Demand Problem.

$$f(A - B) = 1 \quad \text{and} \quad f(B - A) = 1.$$

We can see that the first condition in the definition of weak supermodularity does not hold. Let's study the second condition. We have

$$f(A \cup B) = 2 \quad \text{and} \quad f(A \cap B) = 1,$$

so we can see that the second condition in the definition (3.1) does not hold too, which proves that f is not weakly supermodular.

4.2 Laminarity

Now, we prove also with a counter-example that the tight constraints associated to extreme points do not form a laminar family. We have this situation on the figure (6). We have the root node r , two subsets of nodes $R^1 = \{s_1, s_2\}$ and $R^2 = \{t_1, t_2\}$ and the capacities on the four edges. We also have the different cuts for which the constraints are tight on the figure.

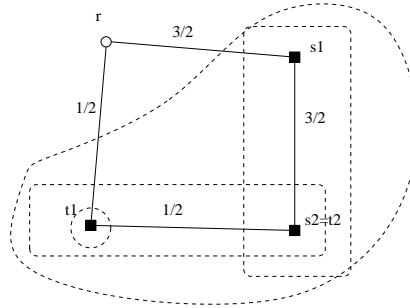


Figure 6: Counter-example for the laminarity in the Single Sink with non-simultaneous Demand Problem.

We have for the cut containing s_1 and s_2 :

$$\frac{3}{2} + \frac{1}{2} = 2,$$

for the cut containing t_1 and t_2 :

$$\frac{1}{2} + \frac{3}{2} = 2$$

for the cut containing t_1 , t_2 and s_1 :

$$\frac{1}{2} + \frac{3}{2} = 2,$$

and for the cut containing t_1 :

$$\frac{1}{2} + \frac{1}{2} = 1.$$

We can clearly see that this subsets do not form a laminar family and so it is for the tight constraints associated to extreme points.

So we have seen that the properties that hold in the case of the Steiner Tree Problem and that lead to the theorem (3.1) do not hold in the case of the Single Sink with non-simultaneous Demand Problem. But it is maybe possible that, even without these properties, we have that $x_e \geq 1/2$ for at least one $e \in E$ in the system (1), related to the Single Sink Problem.

5 Computational search for an interesting point

After this theoretical part, we arrive at the central point of this project. The principal aim was to find an extreme point of the polyhedron described by the linear system (1) with $x_e < 1/2$ for all $e \in E$. The existence of such a point would prove that the theorem (3.1) does not hold in the case of the Single Sink with non-simultaneous Demand Problem.

5.1 First step: Complete graph with 6 nodes

For this research, we have to choose a graph, choose subsets of nodes and find the extreme points of the corresponding linear system.

The first graph that we have chosen is the complete graph with 6 nodes. We did not want to begin with a too bigger graph because the number of constraints is exponential in this problem. In fact, if we had a graph with n nodes, we would have $2^{n-1} - 1$ constraints. We can see this as follows: for each node except the root node, we have the choice to let it enter in a cut or not, that is why we have 2^{n-1} . We also do not have to count the possibility where no node is in the cut, corresponding to the possibility without cut.

The figure (7) shows the complete graph with 6 nodes.

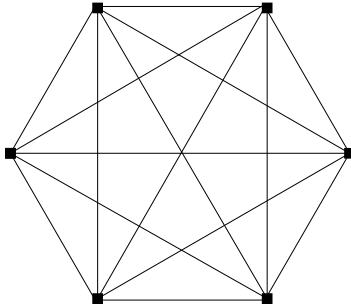


Figure 7: Complete graph with 6 nodes

We also take the complete graph because studying this graph, we would be sure to study all the graphs with 6 nodes. In fact, we can find all the capacities on graphs with 6 nodes on the complete graph by setting high costs on the edges which are not in the interesting graph.

5.1.1 Possibilities of subsets of nodes

After the choice of a graph, we have to choose some subsets of nodes, that we will call $R^i, i = \{1, \dots, k\}$ to build the function f in the linear system (1). Our aim here is to be exhaustive, so we need to find all the possibilities of subsets of nodes in the case of 5 different nodes (the sixth node is the root node and belongs to none of the subsets R^i). We are interested only in the case where $|R^i| < 3, \forall i \in \{1, \dots, k\}$. The figure (8) explains us why. In the case represented in this figure, we have three subsets of nodes in blue and a cut in red. We have that $f(S) = 3$ and so the corresponding equation in the linear system (1) is $x_1 + x_2 + x_3 + x_4 + x_5 \geq 3$. But we know that $3/5 = 0.6$. So the mean of the capacity on these edges should be greater than 0.5. This implies that at least one of the capacity on these edges should be greater than 0.5 and so we cannot have a point with $x_e < 1/2, \forall e \in E$.

We have made a program in C++ to find these different possibilities and to avoid isomorphic possibilities, which code is in the appendix. We show the different possibilities

on the figure (10) in the appendix. The round node is the root node and we have not put the edges on the figure (10).

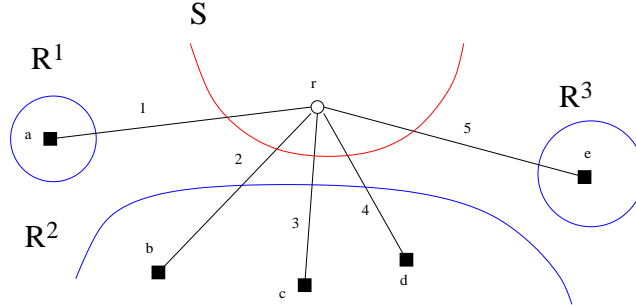


Figure 8: Problem with a subset of 3 nodes

We can observe on the figure (10) that on some graphs, it exists nodes which are in no subset. For each of these nodes, we have always tested two possibilities, one with the node in no subset and one with a supplementary subset R^i containing only this node.

5.1.2 Test of the different possibilities

First of all we build the matrix A which represents, for each cut S , the edges with exactly one endpoint in S . In this matrix, the element $a_{i,j}$ will be equal to 1 if the edge j has exactly one endpoint in the i^{th} cut. This matrix will be the same for all the possibilities of different subsets of nodes R^i . The next step is for each possibility of subsets R^i , to build the function f , to find the extreme points of the polyhedron described by the system (1) and to test if one of these vertices has all its coordinates $x_e < 1/2$.

We build f using a C++ program which is in the appendix. Then, to find the extreme points of the polyhedron, we use the program cdd+ (version cdd+-077a). This program takes as an input a file containing the matrix $-A$ and the vector f for the linear system $Ax \leq f$. Finally, we have made another program to test if there exist some vertices with $x_e < 1/2$ for all $e \in E$. The code of this program is also in the appendix.

5.2 Results for the case with 6 nodes

For the case of graph with 6 nodes, we have tested all the possibilities of subsets R^i , but we have find no vertex of the polyhedron described by the linear system (1) with the property that $x_e < 1/2, \forall e \in E$. We have also tested if it exists some point with $x_e < 1/2$ for some $e \in E$ and have found that some kind of point exists.

5.3 Next step: Complete graph with 11 nodes

After these first tests, we want to test bigger graphs. We are interested in the complete graph with 11 nodes. This graph allows us to test if some interesting points appear in the case of the Single Sink Problem on the Petersen graph, that we can see on figure (9).

In this case, it is more difficult to find all the possibilities of subsets of nodes R^i , because their number becomes really big. This is why we decided to follow our instinct and to test only a few possibilities of subsets R^i . In the case of graphs with 11 nodes, we are interested only in cases of subsets with $|R^i| < 5, \forall i$ for the same reason explained in the case with 6 nodes. We have chosen the following possibilities.

- 5 disjoint subsets containing 2 nodes each,

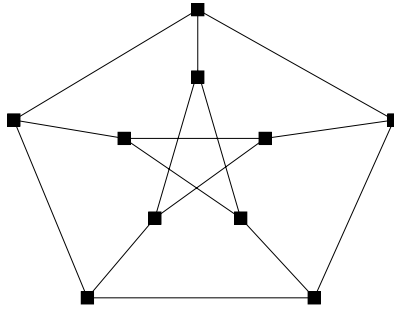


Figure 9: Petersen graph

- 3 disjoint subsets containing 3 nodes each and one node remaining,
- 3 disjoint subsets containing 3 nodes each and 1 subset containing the remaining node,
- 2 disjoint subsets containing 4 nodes each and two nodes remaining,
- 2 disjoint subsets containing 4 nodes each and 1 subset containing the remaining nodes,
- 2 disjoint subsets containing 4 nodes each, 1 subset with one node and one node remaining,
- 2 disjoint subsets containing 4 nodes each and 2 subsets with 1 node each,
- 1 subset per edge of the Petersen graph (15 subsets and 2 nodes per subset),
- 1 subset per edge of the Petersen graph with another node ($15 \cdot 8$ subsets),
- 1 subset per edge of the Petersen graph with another non adjacent node ($15 \cdot 4$ subsets).

The linear system corresponding to the complete graph with 11 nodes was too big for the program cdd+. It takes too much time to find the vertices of the polyhedron described by this linear system. But an extension of this program, the library cddlib-094f allows us to minimize the LP, using a revised Simplex method that updates $(d+1) \times (d+1)$ matrix for a pivot operation, where d is the dimension of x in the LP. This method could also provide us a vertex. We have done that with a cost function with high costs on the edges which are not in the Petersen graph, to reduce the problem to the Petersen graph.

5.4 Results for the case with 11 nodes

We have minimized the LP (1) with a cost function which allows capacity only on the edges of the Petersen graph. We have done that for the different possibilities of subset R^i that we have enumerated higher. For the possibility with 1 subset per edge of the Petersen graph with another node ($15 \cdot 8$ subsets), we find that the point minimizing the LP is

(0,0,0,0,0.42,0,0.42,0,0.42,0,0.3,0.42,0,0,0.42,0.42,0,0,0,0.3,0.42,0,0,0,0.42,0,0,0.3,0.42,0,0,0,0.42,0,0,0.3,0.42,0,0,0.42,0.3,0,0,0.42,0.3,0,0,0.3,0,0.42,0.42,0.3,0,0.42,0.3,0,0.3,0.3).

As we can see, this point verifies $x_e < 1/2, \forall e \in E$. The library cddlib-094f guarantees that this point is a vertex. So that point is the counter-example that we were searching for. It proves that it exists a vertex of the polyhedron corresponding to (1) with $x_e < 1/2$ for all $e \in E$, and, therefore, that Jain's Theorem (3.1) does not work for the Single Sink with non-simultaneous Demand Problem and that we cannot follow the same way as in the case of the Steiner Tree Problem to find a two-approximation algorithm for the Single Sink with non-simultaneous Demand Problem.

A C++ code

In this appendix, we put all the C++ code used to do this project.

A.1 Program to find all the possibilities of subsets of 5 nodes

First of all, we have a program to find all the possibilities of subsets of nodes R^i for the case with a graph with 6 nodes.

We have here the makefile.

```
CC = c++
CXX = c++

all: ssens1

ssens1 :ssens1.o tab1.o tab4.o tab3.o tab2.o tab5.o

tab1.o: tab1.cc tab1.h
tab2.o: tab2.cc tab2.h
tab3.o: tab3.cc tab3.h
tab4.o: tab4.cc tab4.h
tab5.o: tab5.cc tab5.h
ssens1.o: ssens1.cc tab1.h tab2.h tab4.h tab3.h tab5.h
```

We have then a code containing a function which finds the possibilities of subsets of cardinality one, another function writes the subsets and the third function creates a matrix containing all the permutations. The nodes of the graph are represented by integers and so a subset R^i is just a matrix of integer (vector < vector <int> >), each vector of vector<int> representing a subset. The permutation matrix is also a dynamic matrix. Each line represents a permutation as follow: if the position i is j , the permutation will replace i by j . For example, the vector (2,3,1,0) represents the following permutation:

$$0 \rightarrow 2, \quad 1 \rightarrow 3, \quad 2 \rightarrow 1 \quad \text{and} \quad 3 \rightarrow 0.$$

Here we have the code of tab1.h.

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

void cree_tab1(vector <vector<int> >& tab1);
```

```
void ecriture(vector <vector<int> > tab, int taille);

void permutations(vector < vector <int> >& permut);
```

The following code is the corresponding tab1.cc file.

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

//creates tab1 containing all the possibilities of subset of 1 element
void cree_tab1(vector <vector<int> >& tab1)
{
    for(int i(0); i < 5 ; ++i){
        int rajout(tab1.size());
        for(int j(i+1) ; j < 5 ; ++j){
            tab1.push_back(vector <int> (2));
            tab1[rajout][0]=i;
            tab1[rajout][1]=j;
            rajout=rajout+1;}}

//creates permut containing all the possible permutations
void permutations(vector < vector <int> >& permut)
{
    int col(0);
    int b1,c1,d1,e1;
    for(int a(0); a<5 ; ++a){
        for(int b(0); (b<5); ++b){
            if(b!=a){b1=b;}
            else{b1=6;}
            for(int c(0); (c<5); ++c){
                if((c!=a) && (c!=b)){c1=c;}
                else{c1=6;}
                for(int d(0); (d<5) ; ++d){
                    if((d!=a) && (d!=b) && (d!=c)){d1=d;}
                    else{d1=6;}
                    for(int e(0); (e<5) ;++e){
                        if((e!=a) && (e!=b) && (e!=c) && (e!=d)){e1=e;}
                        else{e1=6;}
                        if((b1!=6)&&(c1!=6)&&(d1!=6)&&(e1!=6)){
                            permut.push_back(vector <int> (5));
                            permut[col][0]=a;
                            permut[col][1]=b1;
                            permut[col][2]=c1;
                            permut[col][3]=d1;
                            permut[col][4]=e1;
                            col=col+1;}
                    }
                }
            }
        }
    }
}
```

```

    }
  }
}}

//writes the subsets that we have found
void ecriture(vector < vector <int> > tab,int taille)
{int count(0);
  for(int w(0);w<tab.size();++w){
    if(tab[w].size()==taille){++count;
      for (int r2(0);r2<taille;+r2){
        cout << tab[w][r2]<<" ";}
      cout<<endl;}}
  cout<<count<<endl;}

```

The next code finds all the possible subsets of 2,3,4 and 5 nodes and deletes the possibilities that are isomorphic.

For 2 nodes we have the file tab2.h,

```

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

void cree_tab2(vector < vector<int> >& tab2, vector < vector<int> > tab1);

void test_isomorphie2(vector < vector <int> > permut,vector < vector
<int> >& tab2);

```

and the file tab2.cc.

```

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

//creates tab2 containing all the possibilities of subsets of 2 nodes
void cree_tab2(vector < vector <int> >&tab2, vector < vector <int> >tab1)
{
  for(int l(0); l<tab1.size(); ++l){
    int raj(tab2.size());
    for(int m(0); (m<tab1.size()) && (m!=l); ++m){
      if((tab1[l][0] != tab1[m][1]) || (tab1[l][1] != tab1[m][0])){
        tab2.push_back(vector <int> (4));
        tab2[raj][0]=tab1[l][0];
        tab2[raj][1]=tab1[l][1];
        tab2[raj][2]=tab1[m][0];
        tab2[raj][3]=tab1[m][1];
        raj=raj+1;}
      }
    }
  }
}

void test_isomorphie2(vector < vector <int> > permut,vector < vector

```

```

    <int> >& tab2)
{vector <int> temp;
//we take every vector of tab2
for(int vect(0);vect<tab2.size();++vect){
    //we test if this vector was not eliminated
    if(tab2[vect].size()==4){
    //we apply the permutations
    for(int p(0);p<permut.size();++p){
        if((permut[p][0]!=0)|| (permut[p][1]!=1)|| (permut[p][2]!=2)|| (permut[p][3]!=3)
        || (permut[p][4]!=4)){
            for(int ind(0);ind<4;++ind){
                if(tab2[vect][ind]==0){temp.push_back(permut[p][0]);}
            else if(tab2[vect][ind]==1){temp.push_back(permut[p][1]);}
            else if(tab2[vect][ind]==2){temp.push_back(permut[p][2]);}
            else if(tab2[vect][ind]==3){temp.push_back(permut[p][3]);}
            else{temp.push_back(permut[p][4]);}
        }
    }
//tests if the permuted vector is equal to another vector
for(int tst(vect+1);tst<tab2.size();++tst){
    int n(0);
    for(int tst_iso(0);tst_iso<4;tst_iso=tst_iso+2){
        if(((temp[0]==tab2[tst][tst_iso])&&(temp[1]==tab2[tst][tst_iso+1]))||
        ((temp[0]==tab2[tst][tst_iso+1])&&(temp[1]==tab2[tst][tst_iso]))){
            n=n+2;}
        else if(((temp[2]==tab2[tst][tst_iso])&&(temp[3]==tab2[tst][tst_iso+1]))||
        ((temp[2]==tab2[tst][tst_iso+1])&&(temp[3]==tab2[tst][tst_iso]))){
            n=n+2;}}

//eliminates the vector if it is isomorphic to another one
    if(n==4){tab2[tst].push_back(0);}
}
temp.clear();}}}}

```

For subsets of 3 nodes, we have the code tab3.h,

```

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

void cree_tab3(vector < vector<int> >& tab3, vector < vector<int> > tab1);

void test_isomorphie3(vector < vector <int> > permut,vector < vector
<int> >& tab3);

```

and the code tab3.cc.

```

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

```

```

void cree_tab3(vector < vector <int> >&tab3, vector < vector <int> >tab1)
{
    for(int l(0); l<tab1.size(); ++l){
        int raj(tab3.size());
        for(int m(l+1); m<tab1.size(); ++m){
            for(int r(m+1);r<tab1.size(); ++r){
tab3.push_back(vector <int> (6));
tab3[raj] [0]=tab1[l] [0];
tab3[raj] [1]=tab1[l] [1];
tab3[raj] [2]=tab1[m] [0];
tab3[raj] [3]=tab1[m] [1];
tab3[raj] [4]=tab1[r] [0];
tab3[raj] [5]=tab1[r] [1];
raj=raj+1;}
            }
        }
    }

void test_isomorphie3(vector < vector <int> > permut,vector < vector
<int> >& tab3)
{vector <int> temp;
for(int vect(0);vect<tab3.size();++vect){
    if(tab3[vect].size()==6){
        for(int p(0);p<permut.size();++p){
            if((permut [p] [0] !=0) || (permut [p] [1] !=1) || (permut [p] [2] !=2) ||
(permut [p] [3] !=3) || (permut [p] [4] !=4)){
                for(int ind(0);ind<6;++ind){
                    if(tab3[vect] [ind]==0){temp.push_back(permut [p] [0]);}
else if(tab3[vect] [ind]==1){temp.push_back(permut [p] [1]);}
else if(tab3[vect] [ind]==2){temp.push_back(permut [p] [2]);}
else if(tab3[vect] [ind]==3){temp.push_back(permut [p] [3]);}
else{temp.push_back(permut [p] [4]);}
                }

for(int tst(vect+1);tst<tab3.size();++tst){
    int n(0);
for(int tst_iso(0);tst_iso<6;tst_iso=tst_iso+2){
    if(((temp[0]==tab3[tst] [tst_iso])&&(temp[1]==tab3[tst] [tst_iso+1])) ||
((temp[0]==tab3[tst] [tst_iso+1])&&(temp[1]==tab3[tst] [tst_iso]))){
        n=n+2;}
else if(((temp[2]==tab3[tst] [tst_iso])&&(temp[3]==tab3[tst] [tst_iso+1])) ||
((temp[2]==tab3[tst] [tst_iso+1])&&(temp[3]==tab3[tst] [tst_iso]))){
        n=n+2;}
else if(((temp[4]==tab3[tst] [tst_iso])&&(temp[5]==tab3[tst] [tst_iso+1])) ||
((temp[4]==tab3[tst] [tst_iso+1])&&(temp[5]==tab3[tst] [tst_iso]))){
        n=n+2;}}

    if(n==6){tab3[tst].push_back(0);}
}

temp.clear();}}}}

```

For the subsets of 4 nodes, we have the file tab4.h,

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

void cree_tab4(vector < vector<int> >& tab4, vector < vector<int> > tab1);

void test_isomorphie4(vector < vector <int> > permut,vector < vector
<int> >& tab4);
```

and the file tab4.cc.

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

void cree_tab4(vector < vector<int> >& tab4, vector < vector<int> > tab1)
{ for(int l(0); l<tab1.size(); ++l){
    int raj(tab4.size());
    for(int m(l+1); m<tab1.size(); ++m){
        for(int k(m+1);k<tab1.size(); ++k){
            for(int u(k+1);u<tab1.size(); ++u){
                tab4.push_back(vector <int> (8));
                tab4[raj] [0]=tab1[l] [0];
                tab4[raj] [1]=tab1[l] [1];
                tab4[raj] [2]=tab1[m] [0];
                tab4[raj] [3]=tab1[m] [1];
                tab4[raj] [4]=tab1[k] [0];
                tab4[raj] [5]=tab1[k] [1];
                tab4[raj] [6]=tab1[u] [0];
                tab4[raj] [7]=tab1[u] [1];
                raj=raj+1;}
            }
        }
    }
}

void test_isomorphie4(vector < vector <int> > permut,vector < vector
<int> >& tab4)
{vector <int> temp;
for(int vect(0);vect<tab4.size();++vect){
    if(tab4[vect].size()==8){
        for(int p(0);p<permut.size();++p){
            if((permut[p] [0]!=0)|| (permut[p] [1]!=1)|| (permut[p] [2]!=2)||
(permut[p] [3]!=3)|| (permut[p] [4]!=4)){
                for(int ind(0);ind<8;++ind){
                    if(tab4[vect] [ind]==0){temp.push_back(permut[p] [0]);}
                }
            }
        }
    }
}
```

```

else if(tab4[vect][ind]==1){temp.push_back(permut[p][1]);}
else if(tab4[vect][ind]==2){temp.push_back(permut[p][2]);}
else if(tab4[vect][ind]==3){temp.push_back(permut[p][3]);}
else{temp.push_back(permut[p][4]);}
}

for(int tst(vect+1);tst<tab4.size();++tst){
int n(0);
for(int tst_iso(0);tst_iso<8;tst_iso=tst_iso+2){
if(((temp[0]==tab4[tst][tst_iso])&&(temp[1]==tab4[tst][tst_iso+1]))||
((temp[0]==tab4[tst][tst_iso+1])&&(temp[1]==tab4[tst][tst_iso])))){
n=n+2;}
else if(((temp[2]==tab4[tst][tst_iso])&&(temp[3]==tab4[tst][tst_iso+1]))||
((temp[2]==tab4[tst][tst_iso+1])&&(temp[3]==tab4[tst][tst_iso])))){
n=n+2;}
else if(((temp[4]==tab4[tst][tst_iso])&&(temp[5]==tab4[tst][tst_iso+1]))||
((temp[4]==tab4[tst][tst_iso+1])&&(temp[5]==tab4[tst][tst_iso])))){
n=n+2;}
else if(((temp[6]==tab4[tst][tst_iso])&&(temp[7]==tab4[tst][tst_iso+1]))||
((temp[6]==tab4[tst][tst_iso+1])&&(temp[7]==tab4[tst][tst_iso])))){
n=n+2;}}
if(n==8){tab4[tst].push_back(0);}
}

temp.clear();}}}}

```

Finally for the subsets with 5 nodes, we also have the file tab5.h,

```

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

void cree_tab5(vector < vector<int> >& tab5, vector < vector<int> > tab1);

void test_isomorphie5(vector < vector <int> > permut,vector < vector
<int> >& tab5);

```

and the file tab5.cc.

```

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

void cree_tab5(vector < vector <int> >&tab5, vector < vector <int> >tab1)
{
for(int l(0); l<tab1.size(); ++l){
int raj(tab5.size());
for(int m(l+1); m<tab1.size(); ++m){

```



```

        ((temp[8]==tab5[tst][tst_iso+1])&&(temp[9]==tab5[tst][tst_iso]))){
            n=n+2;}}
    if(n==10){tab5[tst].push_back(0);}
}

temp.clear();}}}}

```

Then we have the file containing the function *main*.

```

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include "tab1.h"
#include "tab2.h"
#include "tab3.h"
#include "tab4.h"
#include "tab5.h"
using namespace std;

vector <vector <int> > tab1,tab2,tab3,tab4,tab5,permut;

int main(){

    cree_tab1(tab1);
    cree_tab2(tab2,tab1);

    cree_tab3(tab3,tab1);
    cree_tab4(tab4,tab1);
    cree_tab5(tab5,tab1);
    permutations(permut);
    test_isomorphie2(permut,tab2);
    test_isomorphie3(permut,tab3);
    test_isomorphie4(permut,tab4);
    test_isomorphie5(permut,tab5);
    ecriture(tab2,4);
    ecriture(tab3,6);
    ecriture(tab4,8);
    ecriture(tab5,10;;

    return 0;}

```

A.2 Program to create the input file for the program cdd+

The second program that we have made is a program taking in a file (*R.txt*) the subsets R^i and giving in the file *six.in* the input for the program cdd+. The format *.in* is the format asked by cdd+. In the file *R.txt*, the subsets R^i are represented as follow: each subset is represented by n number, where n is the number of nodes of our graph minus one. The number in the place i is equal to 0 if the node i is not in the subset R^i and to 1 if it is in the subset. For example, 0 0 1 0 1 represents a subset containing the nodes 3 and 5.

We put here the makefile for this program.

```
CC = c++
CXX = c++
```

```
all: creation
```

```
creation: creation.o cutsfin.o matrice.o f.o input.o
```

```
cutsfin.o: cutsfin.cc cutsfin.h
```

```
matrice.o: matrice.cc matrice.h
```

```
f.o: f.cc f.h
```

```
input.o: input.cc input.h
```

```
creation.o: creation.cc cutsfin.h matrice.h f.h input.h
```

We have then the files *cutsfin.h* and *cutsfin.cc* which creates all the possibilities of cut and write them in the file *cuts.txt*. A cut is represented by some integer, one for each node in the cut.

cutsfin.h:

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
using namespace std;
```

```
void fonc_sortie(vector < vector <int> > cut);
```

```
void cree_cut1(vector < vector <int> >& cut1,int nbre_somm);
```

```
void cree_cut(vector < vector <int> >& cut2,vector < vector <int> >
cut,int nbre_somm);
```

cutsfin.cc:

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include "cutsfin.h"
using namespace std;
```

```
//writes the cuts in a file
```

```
void fonc_sortie(vector < vector <int> > cut)
```

```
{string nom_fichier("cuts.txt");
```

```
ofstream sortie(nom_fichier.c_str(),ios::out|ios::app);
```

```
if (sortie.fail()){
```

```
cerr<<"Erreur: impossible d' crire dans le fichier"<<nom_fichier<<endl;}
```

```
else{
```

```
for(int k(0); k<cut.size(); ++k){
```

```
for (int r(0);r<cut[k].size();++r){
```

```
sortie<<cut[k][r]<<" ";
```

```
sortie<<endl;}}
```

```

sortie.close();}

//creates the cuts with 1 node
void cree_cut1(vector < vector <int> >& cut1,int nbre_somm)
{
    for (int i(0);i<(nbre_somm-1);++i){
        cut1.push_back(vector <int> (1));
        cut1[i][0]=i;}}

//creates the different cuts
void cree_cut(vector < vector <int> >& cut2,vector < vector <int> >
cut,int nbre_somm)
{vector <int> temp1;
    int taille(cut[0].size()-1);
    for (int j(0);j<cut.size();++j){
        for(int i(cut[j][taille]);i<(nbre_somm-1);++i){
            if(i!=cut[j][taille]){
for (int u(0);u<(taille+1);++u){
                temp1.push_back(cut[j][u]);}
                temp1.push_back(i);
                cut2.push_back(temp1);
                temp1.clear();}}}}

```

The next code creates the matrix *A*.
matrice.h:

```

#include <iostream>
#include <vector>
#include <string>
#include <fstream>
using namespace std;

void cree_aretes(vector < vector <int> >& aretes,int nbre_somm);

void cree_matrice6(vector <vector <int> > aretes,int tab[][15], int taille[],
int nbre_const,int nbre_var,int nbre_somm);

void cree_matrice11(vector <vector <int> > aretes,int tab[][55], int taille[],
int nbre_const,int nbre_var,int nbre_somm);

```

matrice.cc

```

#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include "matrice.h"
using namespace std;

//creates an array representing the edges
void cree_aretes(vector < vector <int> >& aretes,int nbre_somm)
{vector <vector <int> > ar;
    vector <int> temp;
    for (int i(0);i<nbre_somm;++i){

```

```

    ar.push_back(vector<int>(1));
    ar[i][0]=i;}

    for (int j(0);j<ar.size();++j){
        for(int i(ar[j][0]);i<nbre_somm;++i){
            if(i!=ar[j][0]){
temp.push_back(ar[j][0]);
temp.push_back(i);
aretes.push_back(temp);
temp.clear();}}}}

//creates the matrix A in the case of a graph with 6 nodes
void cree_matrice6(vector<vector<int>> aretes,int tab[][15],
    int taille[],int nbre_const,int nbre_var,int nbre_somm)
{string nom_fichier("cuts.txt");
    ifstream entree(nom_fichier.c_str());
    string nom_fichier2("matricecdd.txt");
    ofstream sortie(nom_fichier2.c_str());
    vector<int> temp;
    int p,t;

    if (entree.fail()){
        cerr<<"Erreur : impossible de lire le fichier " <<nom_fichier
        << endl;
    } else {
        //reads cuts from the file 'cuts.txt'
        int cut(0);int q(0);
        for(int i(0);i<(nbre_somm-1);++i){
            p=taille[i];cut=cut+1;
            for(int v(0);v<p;++v){
for(int u(0);u<cut;++u){
                entree>>t;
                temp.push_back(t);}//here we have one cut in temp

            //tests if each edge has exactly one endpoint in the cut in temp
for(int i(0);i<nbre_var;++i){
                int n(0);
                for(int j(0);j<2;++j){
                    for(int k(0);k<temp.size();++k){
                        if(aretes[i][j]==temp[k]){
n=n+1;}}
                }
                if(n==1){
                    tab[q][i]=1;}
                else{
                    tab[q][i]=0;}}
            q=q+1;temp.clear();}
        }

        entree.close();}

    if (sortie.fail()){

```

```

        cerr<<"Erreur : impossible d'ecrire dans le fichier " <<nom_fichier2
        << endl;
    } else { //writes the matrix in a file
        for(int i(0);i<nbre_const;++i){
            for(int j(0);j<nbre_var;++j){
sortie<<tab[i][j]<<" ";}
            sortie<<" "<<endl;}
        sortie.close();}}

//create the matrix A for the case of a graph with 11 nodes
void cree_matrice11(vector <vector <int> > aretes,int tab[][55],
    int taille[],int nbre_const,int nbre_var,int nbre_somm)
{string nom_fichier("cuts.txt");
    ifstream entree(nom_fichier.c_str());
    string nom_fichier2("matricecdd.txt");
    ofstream sortie(nom_fichier2.c_str());
    vector <int> temp;
    int p,t;

    if (entree.fail()){
        cerr<<"Erreur : impossible de lire le fichier " <<nom_fichier
        << endl;
    } else {
        int cut(0);int q(0);
        for(int i(0);i<(nbre_somm-1);++i){
            p=taille[i];cut=cut+1;
            for(int v(0);v<p;++v){
for(int u(0);u<cut;++u){
                entree>>t;
                temp.push_back(t);} //hier we have one cut in temp

for(int i(0);i<nbre_var;++i){
                int n(0);
                for(int j(0);j<2;++j){
                    for(int k(0);k<temp.size();++k){
                        if(aretes[i][j]==temp[k]){
n=n+1;}}
                }
                if(n==1){
                    tab[q][i]=1;}
                else{
                    tab[q][i]=0;}}
q=q+1;temp.clear();}
            }

            entree.close();}

    if (sortie.fail()){
        cerr<<"Erreur : impossible d'ecrire dans le fichier " <<nom_fichier2
        << endl;
    } else {

```

```

    for(int i(0);i<nbre_const;++i){
        for(int j(0);j<nbre_var;++j){
sortie<<tab[i][j]<<" ";}
        sortie<<" "<<endl;}
    sortie.close();}}

```

The next code creates the value of the function f .
f.h:

```

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

```

```

void creer_f(int taille [], int m,int nbre_somm);

```

f.cc:

```

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include "f.h"
using namespace std;

```

```

void creer_f(int taille [], int m, int nbre_somm)
{ int p,t;
  int Ri[m][nbre_somm-1];
  string nom_fichier("R.txt");
  ifstream entree(nom_fichier.c_str());
  string nom_fichier2("cuts.txt");
  ifstream entree2(nom_fichier2.c_str());
  string nom_fichier3("fonc.txt");
  ofstream sortie(nom_fichier3.c_str());
  vector <int> temp;
  if (entree.fail()){
    cerr<<"Erreur : impossible de lire le fichier " <<nom_fichier
    << endl;
  } else {\\reads the subsets of nodes R^i from the file 'R.txt'
    for(int j(0);j<m;++j){
      for(int i(0);i<(nbre_somm-1);++i){
entree >> Ri[j][i];}}
    entree.close();}

  if (entree2.fail()){
    cerr<<"Erreur : impossible de lire le fichier " <<nom_fichier2
    << endl;
  } else {//reads the cuts from the file 'cuts.txt'
    int cut(0);int q(0);
    for(int i(0);i<(nbre_somm-1);++i){
      p=taille[i];cut=cut+1;

```

```

        for(int v(0);v<p;++v){int w(0);
for(int u(0);u<cut;++u){
    entree2>>t;
    temp.push_back(t);}//here we have only one cut in temp
    //creates the value of the function f
for (int j(0);j<m;++j){
    int n(0);
    for(int k(0);k<temp.size();++k){
        if(Ri[j][temp[k]]==1){
            n=n+1;}}

    if(n>w){w=n;}
}

if (sortie.fail()){
    cerr<<"Erreur : impossible d'ecrire dans le fichier " <<nom_fichier3
    << endl;
} else {//writes the value of f in a file ('fonc.txt')
    sortie<<w<<" "<<endl;}

q=q+1;temp.clear();}
}

entree2.close();}}

```

The next code really creates the file *six.in*.
input.h:

```

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

```

```
void creer_input(int ligne, int col, int nbre_const);
```

input.cc

```

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include "input.h"
using namespace std;

```

```

void creer_input(int ligne, int col, int nbre_const)
{
    string nom_fichier("fonc.txt");
    ifstream entree(nom_fichier.c_str());
    string nom_fichier2("matricecdd.txt");

```

```

ifstream entree2(nom_fichier2.c_str());
string nom_fichier3("six.in");
ofstream sortie(nom_fichier3.c_str());
int tab[col];
int t;

if (sortie.fail()){
    cerr<<"Erreur : impossible d'ecrire dans le fichier " <<nom_fichier3
    << endl;
} else { //writes the beginning of the input file 'six.in'
    sortie<<"H-representation"<<endl;
    sortie<<"begin"<<endl;
    sortie<<" "<<ligne<<" "<<col<<" "<<"real"<<endl;

    for(int r(0);r<nbre_const;++r){

        if (entree.fail()){
            cerr<<"Erreur : impossible de lire le fichier " <<nom_fichier << endl;
            } else { //reads the rth value of f
            entree >> t;
            tab[0]=-t;}

        if (entree2.fail()){
            cerr<<"Erreur : impossible de lire le fichier " <<nom_fichier2
            << endl;
            } else { //reads the rth line of the matrix A
            for(int i(1);i<col;++i){
                entree2 >> tab[i];}}

//writes in 'six.in' the matrix and the value of f
    for(int j(0);j<col;++j){
        sortie <<tab[j]<<" ";}
        sortie<<endl;}

//writes in 'six.in' the constraint on each capacity (xe >= 0)
    for(int z(1);z<col;++z){
        for(int s(0);s<(col);++s){
            if(s==z){sortie<<1<<" ";}
            else if(s != z){sortie<<0<<" ";}}
        sortie<<endl;}
        sortie<<"end";}

entree.close();
entree2.close();
sortie.close();
}

```

Finally, we have the file *creation.cc* containing the function *main*. Here we maybe have to change the value of the variable *m*, representing the number of subsets R^i . We also have to delete the content of the file *cuts.txt* before every new test, because the writing in this file is in append mode.

```
#include <iostream>
```

```

#include <vector>
#include <string>
#include <fstream>
#include "cutsfin.h"
#include "matrice.h"
#include "f.h"
#include "input.h"
using namespace std;

vector <vector <int> > cut1,cut2,cut3,cut4,cut5,cut6,cut7,cut8,
cut9,cut10;
vector <vector <int> > aretes;
vector <int> temp;
int m(5);
int nbre, nbre_somm,nbre_const, nbre_var, col, ligne;

int main(){

    cout<<"Combien de sommets a le graphe:6 ou 11?"<<endl;
    cin>>nbre;
    if(nbre==6){//case with 6 nodes
        nbre_somm=6;
        nbre_const=31;
        nbre_var=15;
        col=16;
        ligne=46;

        cree_cut1(cut1,nbre_somm);

        cout<<cut1.size()<<endl;
        fonc_sortie(cut1);

        cree_cut(cut2,cut1,nbre_somm);
        fonc_sortie(cut2);

        cree_cut(cut3,cut2,nbre_somm);
        fonc_sortie(cut3);

        cree_cut(cut4,cut3,nbre_somm);
        fonc_sortie(cut4);

        cree_cut(cut5,cut4,nbre_somm);
        fonc_sortie(cut5);

        int tab[nbre_const][15];
        int taille[5]={5,10,10,5,1};
        cree_aretes(aretes,nbre_somm);
        cree_matrice6(aretes,tab, taille, nbre_const, nbre_var,nbre_somm);
        creer_f(taille,m,nbre_somm);
    }
}

```

```

creer_input(ligne,col,nbre_const);}

else if (nbre==11){//case with 11 nodes
    nbre_somm=11;
    nbre_const=1023;
    nbre_var=55;
    col=56;
    ligne=1078;

    cree_cut1(cut1,nbre_somm);

    cout<<cut1.size()<<endl;
    fonc_sortie(cut1);

    cree_cut(cut2,cut1,nbre_somm);
    fonc_sortie(cut2);

    cree_cut(cut3,cut2,nbre_somm);
    fonc_sortie(cut3);

    cree_cut(cut4,cut3,nbre_somm);
    fonc_sortie(cut4);

    cree_cut(cut5,cut4,nbre_somm);
    fonc_sortie(cut5);

    cree_cut(cut6,cut5,nbre_somm);
    fonc_sortie(cut6);

    cree_cut(cut7,cut6,nbre_somm);
    fonc_sortie(cut7);

    cree_cut(cut8,cut7,nbre_somm);
    fonc_sortie(cut8);

    cree_cut(cut9,cut8,nbre_somm);
    fonc_sortie(cut9);

    cree_cut(cut10,cut9,nbre_somm);
    fonc_sortie(cut10);

    int tab11[nbre_const][55];
    int taille11[10]={10,45,120,210,252,210,120,45,10,1};

    cree_aretes(aretes,nbre_somm);
    cree_matrice11(aretes,tab11, taille11, nbre_const, nbre_var,nbre_somm);
    creer_f(taille11,m,nbre_somm);

    creer_input(ligne,col,nbre_const);}

return 0;}

```

A.3 Program to test if some points verify $x_e < 1/2, \forall e \in E$

The next code takes an input file containing points and tests if it exists a point with $x_e < 1/2, \forall e \in E$ and if it exists some points with $x_e < 1/2$ for some $e \in E$. Here we have to change the value of the variable *nbre_var* if the size of the graph changes.

```
#include <iostream>
#include <vector>
#include <fstream>
#include <string>
using namespace std;

int main(){

vector <double> tab;
string nom_fichier("test.txt");
ifstream entree(nom_fichier.c_str());
int h;
int i;
int un;
double num;
int k;
int m(0);
int nbre_var(15);

if (entree.fail()){
    cerr<<"Erreur : impossible de lire le fichier " <<nom_fichier << endl;
} else {

    while(!entree.eof()){m=m+1;
        entree >> un;
        for(i=0; i<=(nbre_var-1) ; ++i){
            entree >> num;
            tab.push_back(num);}
        h=0;
        for(k=0; k<tab.size(); ++k){
            if(tab[k] < 0.5){
if(tab[k] != 0){//tests if some points have x_e < 1/2 for some e in E
cout<<tab[k]<<"ligne " <<m<<endl;};
h=h+1;}

            }//if h=nbre_var, we have found a counter-example
            if(h==nbre_var){cout<<"Contre-exemple trouve ligne " <<m<<endl;
                for(int z(0);z<=(nbre_var-1); ++z){
cout <<tab[z]<<" ";}
                tab.clear();
                cout<<endl;
            }
            else
                {tab.clear();}

        }
        entree.close();}
return 0;}
```

A.4 Program to test if an interesting point is a vertex

The last code tests if a point is a vertex, so for a given point, it gives in which lines the inequality in the LP are verified with equality. Here, the point is represented by an array.

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
using namespace std;

int taille(55);

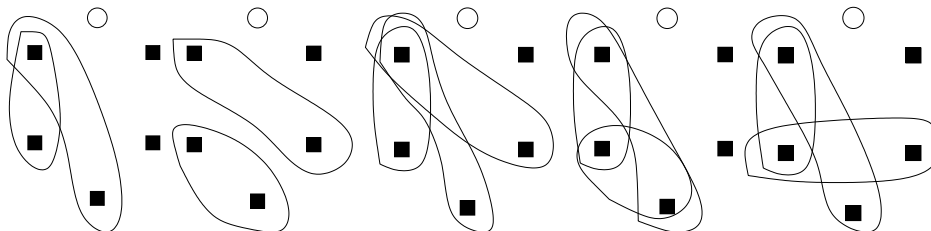
double sol[55] = {0,0,0,0,0.42,0,0.42,0.42,0,0.3,0.42,0,0,0.42,0.42,0,0,
0,0.3,0.42,0,0,0,0.42,0,0,0.3,0.42,0,0,0,0.42,0,0.3,0.42,0,0,0,0.42,0.3,
0,0,0,0,0.3,0,0.42,0.42,0.3,0,0.42,0.3,0,0.3,0.3};

double tab[55];
string nom_fichier("testeg.txt");//this file contains the matrix A and
the values of -f
ifstream entree(nom_fichier.c_str());
double x,temp;

int main(){
    if(entree.fail()){
        cerr << "Erreur: impossible de lire le fichier "<<nom_fichier<< endl;}
    else{int count(0);
        while(!entree.eof()){
            count = count+1;
            entree>>x;
            for(int i(0);i<taille;++i){
                entree>>tab[i];}
            temp=0;
            for(int j(0);j<taille;++j){
                temp = temp + (sol[j]*tab[j]);}
            if(temp == -x){
                cout<<"Egalite ligne "<<count<<endl;}}
            entree.close();}

    return 0;}
```

B Possibilities of subsets R^i with a graph with 6 nodes



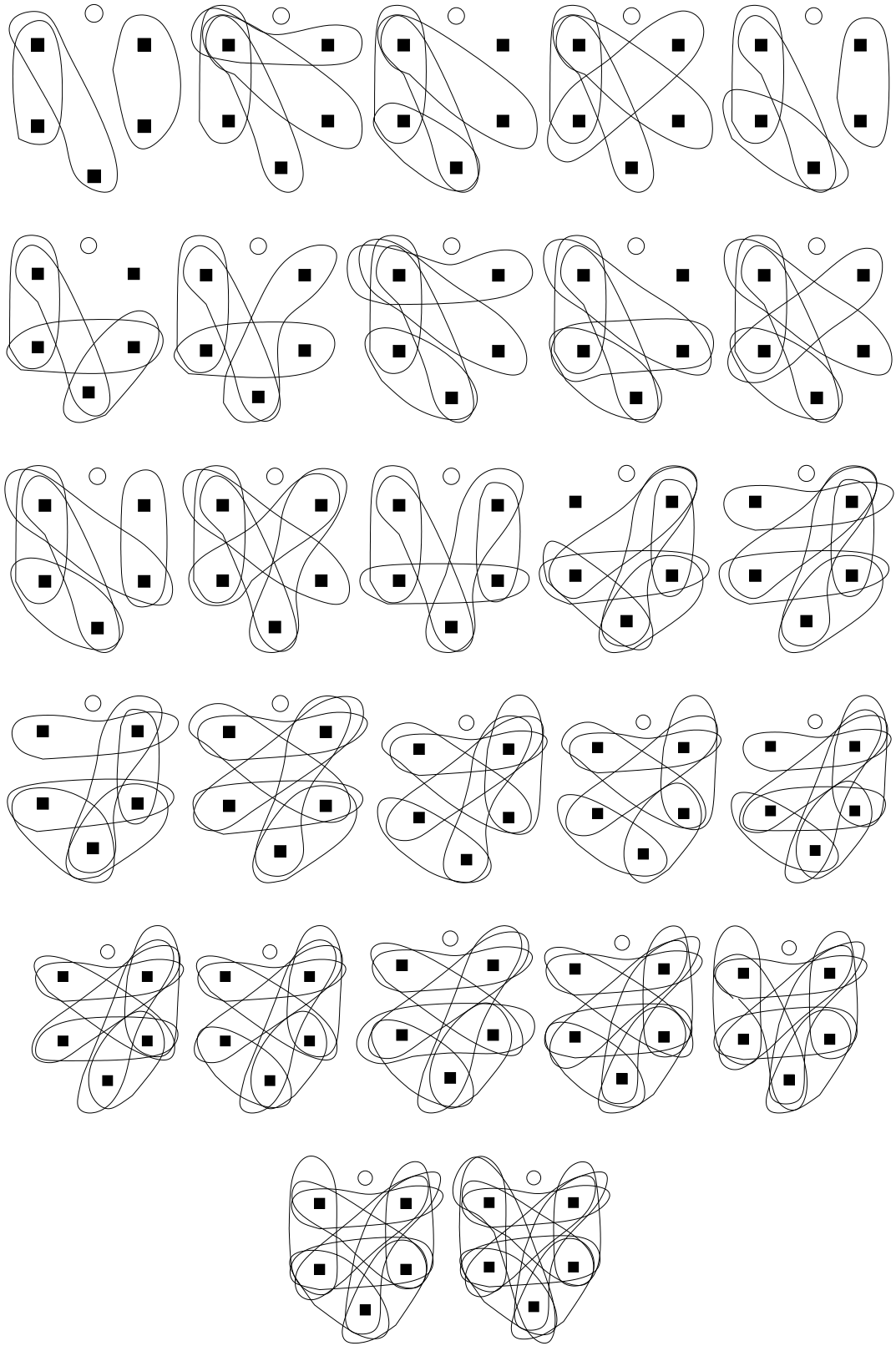


Figure 10: Possibilities of subsets R^i with a graph with 6 nodes

C References

cdd+ (version cdd+-077a) and cddlib (version cddlib-094f)

Programmer: Komei Fukuda, ETHZ - EPFL, McGill University.

On Internet: www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html