University of Nice Sophia Antipolis
Department of Computer Science

Master of Science in
Ubiquitous Networking and Computing

Research Internship Report

The Subset Interconnection Design problem with Subset Uncertainty

Participant
Ioannis Mantas

Supervisors
Christelle Caillouet
David Coudert

31/8/2016

Abstract

Subset Interconnection Design is an $\mathcal{NP}-hard$ problem which draws its motivation from various applications, such as the design of scalable overlay networks, the inference of inter-molecular protein connectivity, the design of vacuum systems and several other problems in the industry. In that problem, we are given a set of vertices V and a collection S of subsets of V and we want to find an edge set E of minimum cardinality such that every subset S_i of the collection S induces a connected subgraph of $G = (V, E)$. In this work, we concern ourselves with a generalization of the problem, which arises mainly from computational structural biology and where our data is covered by Subset Uncertainty. Under this scope, V is partitioned into a set of types T, and we are not aware of the exact composition of each subset S_i. We only know the number of vertices of each type that it is composed by. So, apart from having to find the edge set E of minimum cardinality respecting the constraints, we also have to find a collection S with such composition, that will lead us to the optimal edge set. In the context of Subset Uncertainty we start by formally defining the problem and consider its hardness. Following, we analyse the problem and the structure of the solution to give bounds for the solution. Later on, we model the problem using an *MILP* approach. Moreover, we present and analyse an *approximation algorithm*. Finally, we devise heuristics based on *LP-relaxation* and conclude by making a series of experiments on different datasets comparing all the aforementioned methods, and presenting the results.

Contents

1 Subset Interconnection Design

This report is divided into eight main sections. In this first section we introduce the original SID problem, formally define it and illustrate it with a simple example for its further understanding. Following, we briefly consider the related work, by making a small reference to its complexity, which is important when considering the problem, and we present a few other notable results.

1.1 Description of the problem

Introduction

The problem we consider, Subset Interconnection Design, draws its motivation from various research areas. It has multifarious applications, and for that reason, it has been studied by different research communities, usually independently and under several different names. It has been studied as the Subset Interconnection Design (SID) problem, for the design of vacuum systems by the respective community [8,9,10] but also for the establishment of communication networks where there is a need for internal connectivity of the subsets [19]. Moreover, it has been studied for the design of scalable overlay networks, referred as Minimum Topic-Connected Overlay (min-TCO), where the goal is to support decentralized topic-based publish/subscribe communication [5,6,7,13]. More lately, research has been also made regarding the design of reconfigurable interconnection systems, referred as the Interconnection Graph Problem (IGP) [11,12], as well as with respect to inference of underlying social networks under the general name Network Inference [4]. Finally, and more recently, it has been studied, associating with structural biology, with the aim to infer the pairwise contacts between the constituting proteins of macro-molecular assemblies [2,3]. We will focus more on this last application of SID, as it is the scientific area from which we draw the motivation to study the generalization of the SID where we consider Subset Uncertainty.

Problem statement

As mentioned above, many different names and definitions have been given to the problem. We formally define the problem as follows:

Definition 1 (Subset Interconnection Design problem, SID)
Input: *i) A vertex set* **V**.

ii) A collections of subsets $S = \{S_i \mid S_i \subseteq V_i$ *and* $i \in I \subseteq \mathbb{N}\}$

Objective: *Find a graph* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *such that:*

i) Each induced subgraph $G[S_i]$ *is connected.*

ii) Edge set E *is minimized.*

Prosaically, our goal is to find a graph $G = (V, E)$ whose edge set E both minimizes the number of edges and complies to the set of connectivity constraints. Each of these connectivity constraints is actually represented by a subset in the collection S.

An example

Figure 1 is an illustration of a small instance of the SID problem for the better understanding of the problem.

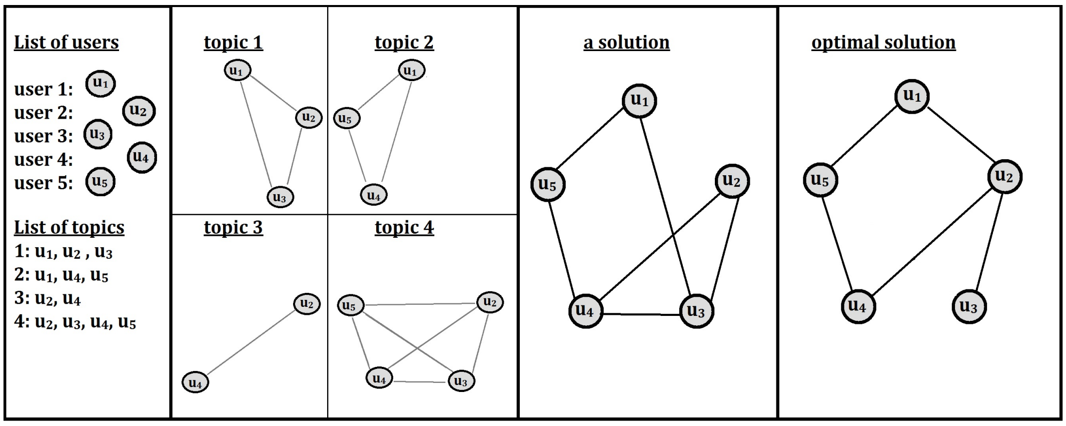


Figure 1: A SID instance with two feasible solutions.

Suppose we are in a *publish/subscribe topic-based* context, as in [5,6,7,13], and we have a set of users, V, who are interested in a set of topics S. We want *topic-connectivity* to exist which means that all users interested in a topic should be connected only through users interested in that topic. So, as we can see in Figure 1, with $|V| = 5$ and $|S| = 4$, we observe that each topic/subset imposes a connectivity constraint. Also, the figure illustrates that in the optimal solution each $G[S_i]$ must induce a spanning tree. But, as we can observe, it is not ensured that graph G will be a tree itself.

1.2 Related work

Complexity

To get a notion of the problem's complexity we can consider the well-known $\mathcal{NP} - hard$ problem, Set Cover, where, given a set of elements $U = \{1, 2, ..., m\}$, called the *universe*, and a collection S of n sets whose union equals the universe, the aim is to identify the smallest sub-collection of S whose union equals the universe. From one perspective we can observe the similarities between SID and Set Cover, although in SID, there is a fundamental difference, which is the extra topological constraint imposed by the connectivity of the subsets.
Observing this relation between the two problems, we can get a intuition of the complexity of SID. Nevertheless, it has been proved in several different papers [2,7,8], that the optimization version of SID is an $\mathcal{NP} - hard$ problem, where most of the proofs rely on a reduction from the Set Cover. Moreover, it has been proved that the SID problem cannot be approximated within a constant factor in polynomial time [7]. Finally, the actual lower bound of the inapproximability is $\Omega(log(|V|))$ [4], where $|V|$ is the number of vertices of the input graph G.

Other results

We summarize some important results and approaches that have been made for the Subset Interconnection Design problem so far.
First of all, there is a greedy algorithm which yields a $O(log|S|)- approximation$, where $|S|$ is the number of subsets [2,4,7,19]. It is very similar to the well-known greedy algorithm of the Set Cover problem and follows the same rule by picking at each step the edge which belongs to the largest number of subsets. This algorithm actually almost meets the lower bound of inapproximability, which is, as we mentioned, $\Omega(log(|V|))$. This algorithm will be used as a part of the algorithms we later devise.
Another approach that has been done, is concerned with the divide & conquer design paradigm. An algorithm has been devised with respect to the design of overlay networks for topic-based publish/subscribe systems [5]. In that algorithm, random partitioning is used for the division of the problem, and later, the partial solutions are combined in a greedy manner, similar to the aforementioned algorithm. It is reported to be in position to output solutions significantly faster than any other algorithm. However, it comes with a slight increase in the number of edges, compared to the greedy algorithm. It is expected to be of interest in applications of SID where the execution time is more important than a potentially less optimal solution.

Moreover, the third approach that has been done is based on a Mixed Integer Linear Programming formulation [2]. It was devised to deal with applications of SID regarding structural biology and for that reason it is promising for our problem. It has the advantage that it not only solves the problem to optimality but makes it also possible to output the set of all feasible solutions. This is of high importance for the structural biology community, as it is of high importance to analyze the solutions in order to evaluate the effectiveness of the methods and the approaches, as well as the ways in which such algorithms should be used. This linear programming modeling is done with a flow – based formulation, the basis of which we use and expand accordingly for the Subset Uncertainty - SID in a later section. It has also been used to solve the weighted version of the SID problem where the weight function on the edges is a probability that represents the likelihood of appearance of a potential edge given prior beliefs [3].
It must also be noted that other modifications of the general problem have been studied. In particular there has been a study for a few alterations which introduce topological constraints to the problem. Under that scope either each vertex induced subgraph $G[S_i]$, or the whole graph G has to be of specific graph classes [14,15,16]. The three main graph classes that have been taken into consideration so far have been *stars*, *paths* and *trees*. Under specific conditions there exist polynomial algorithms to solve SID for these classes to optimality. At first place, and surely not in the modelling phase of the SU-SID, these do not seem to have any relevance. But it is not unlikely, that when we will be attempting to find ways to exploit the behavior of the proteins and the oligomers, we are lead into one of these topological graph classes, or others not yet studied.
Finally, some research has been done on the online version of the problem aiming to infer social networks from outbreaks [4], which does not seem to be of interest in our case, since our input is constant and not changing eventually. Several genetic algorithms have been devised and compared to the aforementioned greedy algorithms [1]. And, there has also been interest towards a polynomial-time data reduction [6] as well as in the approximability and hardness of the problem for special instances [13].

2 Introducing Subset Uncertainty

In this second section of the report the notion of Subset Uncertainty is introduced, which is the main subject of this research. Initially, we explain the motivation of the problem and its potential applications and illustrate it with a simple example. Following we make some important observations regarding the definition of the problem. Finally, we consider the complexity of the problem by showing it is an $\mathcal{NP} - hard$ optimization problem.

2.1 Description of the problem

Motivation

As mentioned above, the motivation for the notion of Subset Uncertainty arises in structural biology. In order to understand this motivation clearly we must first explain in detail how the initial problem, SID, is related with structural biology.
Macro-molecular assemblies are massive chemical structures, many of which involve from tens to hundreds of macro-molecules. Unfortunately, there is very little information regarding their structure. Acquiring information for their structure, by building corresponding models, is of high interest by the respective research communities.
In that direction, there exist methods which involve biophysical experiments that are in place to provide us information on the composing structures of the macro-molecular assemblies, which are called oligomers. So, there arises the need to infer the connectivity between these overlapping oligomers which have been produced by several types of experiments.
We consider the whole macro-molecular assembly as a graph, the vertices as proteins or molecules and the oligomers as subsets. By requiring connectivity among proteins of each oligomer and looking to find the minimum number of pair-wise contacts the application Subset Interconnection Design in Structural Biology becomes obvious. And only under this context, has the inference of such protein contacts been studied until now [2,3].

If we want to look at the aforementioned problem more realistically, we will have to make things more complicated though. Let us consider two well known and widely used methods in structural biology.

• A solution to the *Stoichiometry Determination* problem gives us only knowledge of the type of a molecule and partially of the structure of each

complex [20]. Therefore, by observing the relation, we are in position to know which type every vertex belongs to.
• The outcome of the *Native Mass Spectrometry* method gives us the number of molecules of each type contained in an oligomer but not which specific instances they are [21]. So, in the graph context, we now know how many vertices of each type each subset must have, but not which particular ones.
These two combined, bring us to a situation where we are given an assembly together with the list of constituting protein types as well as a list of associated complexes. So, in order to be in place to deal with this problem and efficiently provide information about the elucidation of the connectivity between macro-molecular assemblies, it is necessary to find ways to solve efficiently a generalization of SID, which we define as the Subset Interconnection Design with Subset Uncertainty.

Problem statement

We start by giving an initial definition to the problem. After explaining it, we complement it with additional constraints to fully represent our problem and the desirable solutions.

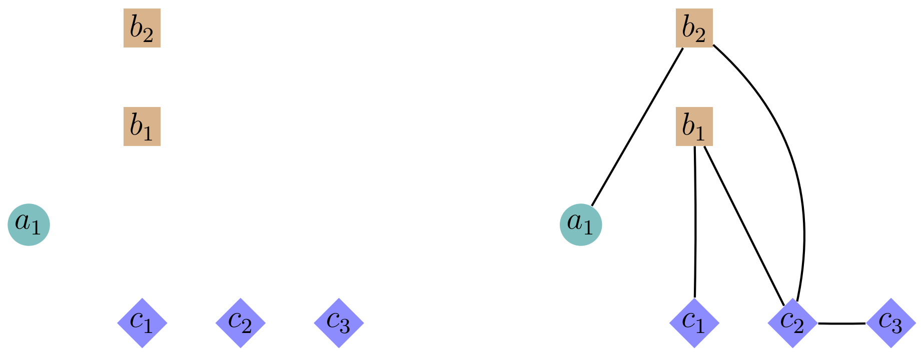
Definition 2.1 (Initial definition for SU-SID)
Input: *i) A vertex set* $\mathbf{V}$.
ii) A partition $\mathbf{T}$ *of the vertex* V.
iii) A matrix, of subset specifications, $\mathbf{M}$, *where element* m_{ij} *represents the number of vertices of type* T_j *subset* S_i *must have.*
Objective: *Find a graph* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *and a collection of subsets* $\mathbf{S}$ *such that: i) Each subset* S_i *satisfies the corresponding vector* M_i.
ii) Each induced subgraph $G[S_i]$ *is connected.*
iii) Edge set E *is minimized.*

◦ *Note 1:* We say that a vertex u is *chosen* by a subset S_i, or respectively a subset *chooses* a vertex, if $u \in S_i$.
◦ *Note 2:* We say that a subset S_i *satisfies* a vector M_i if the exact number of vertices of each type T_j is *chosen*. Respectively, a matrix of subset specifications M is *satisfied* if all vectors are *satisfied*, $m_{ij} = |S_i \cap T_j|$.

In simple terms, we have a set of types of vertices T. Each vertex is of a certain type and we are not aware of the exact composition of each subset, but rather the number of vertices of each type that it is composed by. So apart from having to find the edge set E of minimum cardinality we also have to ensure that the vertex specifications for each subset are respected.

An example

For a thorough understanding of the problem, a SU-SID instance, with $|V| = 6$, $|T| = 3$ and $|S| = 6$, and an optimal solution is illustrated.



$M_1 = [1,1,0]$	$V = T_1 \cup T_2 \cup T_3$
$M_2 = [0,1,1]$	
$M_3 = [0,2,1]$	$T_1 = \{a_1\}$
$M_4 = [1,1,1]$	$T_2 = \{b_1, b_2\}$
$M_5 = [0,1,3]$	$T_3 = \{c_1, c_2, c_3\}$

$S_1 = \{a_1, b_2\}$
$S_2 = \{b_2, c_2\}$
$S_2 = \{b_1, b_2, c_2\}$
$S_4 = \{a_1, b_2, c_2\}$
$S_5 = \{b_1, c_1, c_2, c_3\}$

Figure 2: A SU-SID instance and an optimal solution.

On the left side of Figure 2, is the input of the problem for the given instance. That is, a vertex set V partitioned into 3 types and the specifications matrix M. On the right side, an optimal solution of the instance is illustrated. That is an edge set E and a collection of subsets S satisfying M.
It is easy to observe that all the constraints posed in the definition of SU-SID are satisfied. Also, we know that this is an optimal solution as G is a tree, but we must note that this is not always the case.

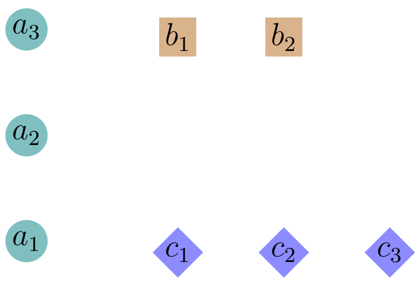
Notes on the definition

The SU-SID problem, as we have defined it, is a generalization of the Subset Interconnection Design problem. In SID we have only one constraint. That is, that each induced subgraph $G[S_i]$ should be connected, which remains as it is, in Definition 2.1. In our problem we have to add another constraint to ensure the correctness of our proposed generalization which has to do with the correctness of the specifications matrix M.
But, we are studying the problem for a very specific application, that of Structural Biology, and as mentioned above, each element of the problem has a physical representation. Therefore, we have to add additional constraints to

the definition in order to address the problem with respect to the application.

In [2], where SID was studied for the same motivation, the connectivity of the total graph G was not required in the constraints. Nevertheless, it was mentioned that the subsets in the collection S were overlapping. Indeed, since we want to infer the connectivity in a macro-molecular assembly, it would not make sense if a solution would include different connected components. So, it should necessarily address the connectivity of graph G.

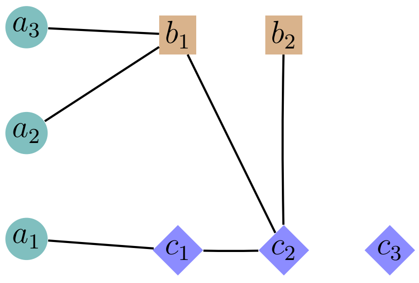
Following, we look into some solutions to a SU-SID instance, which will guide us to impose the necessary extra constraints defining the problem with respect to its applications. Suppose we have the following instance, illustrated in Figure 3.



$M_1 = [2,1,0]$	$V = T_1 \cup T_2 \cup T_3$
$M_2 = [1,1,1]$	
$M_3 = [1,2,1]$	$T_1 = \{a_1, a_2, a_3\}$
$M_4 = [0,1,1]$	$T_2 = \{b_1, b_2\}$
$M_5 = [1,0,2]$	$T_3 = \{c_1, c_2, c_3\}$

Figure 3: A SU-SID instance with $|V| = 8$, $|T| = 3$ and $|S| = 5$.

Figure 4 illustrates an optimal solution to the given instance according to Definition 2.1. We observe that the graph G is not connected.



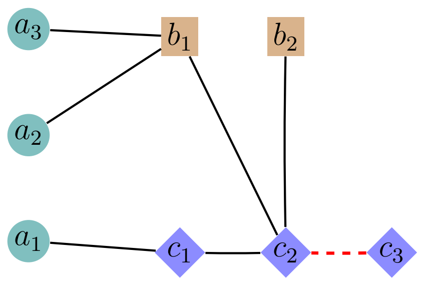
$S_1 = \{a_2, a_3, b_1\}$
$S_2 = \{a_2, b_1, c_2\}$
$S_2 = \{a_2, b_1, b_2, c_2\}$
$S_4 = \{b_2, c_2\}$
$S_5 = \{a_1, c_1, c_2\}$

Figure 4: An optimal solution with the initial definition.

We actually have a connected component and a set of disjoint vertices, which is only vertex c_3 in our case. To tackle this situation, the first idea would be to add a constraint requiring that the total graph G is connected. So we add an extra constraint to Definition 2.1.

iv) Graph G is connected.

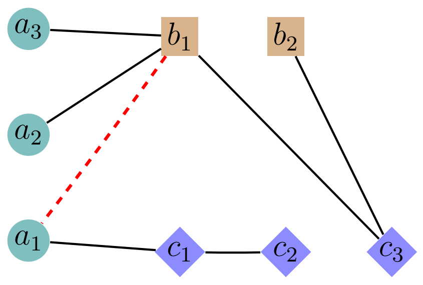
Some possible solutions could be as follows.



$S_1 = \{a_2, a_3, b_1\}$
$S_2 = \{a_2, b_1, c_2\}$
$S_2 = \{a_2, b_1, b_2, c_2\}$
$S_4 = \{b_2, c_2\}$
$S_5 = \{a_1, c_1, c_2\}$

Figure 5: An optimal solution with the constraint that G is connected.

As seen in Figure 5, a feasible solution would be just to arbitrarily add edges, between each disjoint vertex to the connected component, (c_2c_3) in our case. This solution would actually be optimal as well. But for our problem, this is not sufficient because although G would be connected, c_3 would not be *chosen* by any subset and (c_2c_3) could possibly be a *false positive*. In Structural Biology, each vector $M_i \in M$ comes from a series of experiments conducted [20,21]. Therefore, connecting the graph G with edges which have no physical meaning, and neglecting in that way input information, is in the wrong direction. To avoid such solutions, we could add that *each vertex $u \in V$ is chosen by at least 1 subset.*



$S_1 = \{a_2, a_3, b_1\}$
$S_2 = \{a_2, b_1, c_3\}$
$S_2 = \{a_2, b_1, b_2, c_3\}$
$S_4 = \{b_2, c_3\}$
$S_5 = \{a_1, c_1, c_2\}$

Figure 6: An optimal solution adding the constraint that G is connected and that each vertex is chosen by at least 1 subset.

In Figure 6 we observe a situation similar with the aforementioned, but with respect to the edges. There can be edges which are in the solution only to ensure connectivity of G, in our case (a_1, b_1). But again, it is probable that such edges can be *false positives* and must be avoided. So, we conclude that *(iv)* is insufficient, and we have to add one more constraint to Definition 2.1:

v) Each edge $e \in E$ contributes to at least 1 subset.

∘*Note:* We will say that an edge (uv) *contributes* to a subset S_i if both vertices u and v are *chosen* by S_i.

Remark 2.1
By adding constraints *(iv)* and *(v)* in Definition 2.1, it follows that each vertex will be *chosen* by at least 1 subset as well. The opposite is not true, as shown in Figure 6.

Summarizing the aforementioned, we formally state our problem below, extending Definition 2.1. All work presented later, considers this definition.

Definition 2.2 (SUBSET INTERCONNECTION DESIGN with SUBSET UNCERTAINTY problem, SU-SID)
Input: *i) A vertex set* $\mathbf{V}$.
ii) A partition $\mathbf{T}$ *of the vertex* V.
iii) A matrix, of subset specifications, $\mathbf{M}$, *where element* m_{ij} *represents the number of vertices of type* T_j *subset* S_i *must have.*
Objective: *Find a graph* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *and a collection of subsets* $\mathbf{S}$ *such that:* *i) Each subset* S_i *satisfies the corresponding vector* M_i.
ii) Each induced subgraph $G[S_i]$ *is connected.*
iii) Edge set E *is minimized.*
iv) Graph G *is connected.*
v) Each edge $e \in E$ *contributes to at least 1 subset.*

2.2 Complexity

In this section we analyse the complexity of our problem. We do not go in depth to prove the hardness of SUBSET UNCERTAINTY - SID by reduction but we rather explain how it is a generalization of SID. We also give a notion of the high complexity of the problem caused by a *combinatorial explosion.*

Remark 2.2 (SU-SID hardness)
SU-SID *is an* $\mathcal{NP}-$ *hard optimization problem*
As mentioned, SID has been proved to be an $\mathcal{NP}-$*hard* problem for $|V| > 2$ [12,13] We can observe that SU-SID coincides with SID when each type of vertices is a singleton. Since the vertices can only be of one type, there will be a one-to-one correspondence between each $u \in V$ and each $t \in T$, and thus $|V| = |T|$ will hold.
Suppose we now have a smaller number of types with $2 < |T| < |V|$. Even if we had one vertex of each type, connecting those vertices would

be $\mathcal{NP}-hard$, since it would be an instance of SID. But in SU-SID we also have to consider all the possible combinations that could lead to that problem. So, it becomes obvious, SU-SID is a generalization of SID. Therefore, SU-SID is $\mathcal{NP}-hard$. So, in any case, we have a problem which is at least as hard as SID to solve.

We say it is at least as hard, because not knowing the exact composition of the subsets means that all the possible combinations of vertices satisfying the specifications of each subset should be considered. The product of all the possible combinations of each set will lead to a, so called, *combinatorial explosion* as the number of sets and vertices increases.

If we have $|T| = r$ and $|S| = k$ and if m_{ij} denotes the vertices of type T_j subset S_i has, we will have a number of possible combinations which is of the order, given by the formula below:

$$\binom{|T_1|}{m_{11}} \times \binom{|T_2|}{m_{12}} \cdots \times \binom{|T_r|}{m_{1r}} \cdots \times \binom{|T_j|}{m_{ij}} \cdots \times \binom{|T_1|}{m_{k1}} \times \binom{|T_2|}{m_{k2}} \cdots \times \binom{|T_r|}{m_{kr}}$$

$$\#\text{combinations} = \mathcal{O}\left(\prod_{i=1}^{k}\prod_{j=1}^{r}\binom{|T_j|}{m_{ij}}\right)$$

Summing up, we showed that SU-SID is a generalization of SID which is an $\mathcal{NP}-hard$ problem. Therefore, SU-SID is also $\mathcal{NP}-hard$. Moreover, we explained that there exists a *combinatorial explosion* on the possible solutions. So, we conclude that SU-SID is a highly complex problem. To address this complexity and solve the problem efficiently will not be trivial.

3 Problem analysis

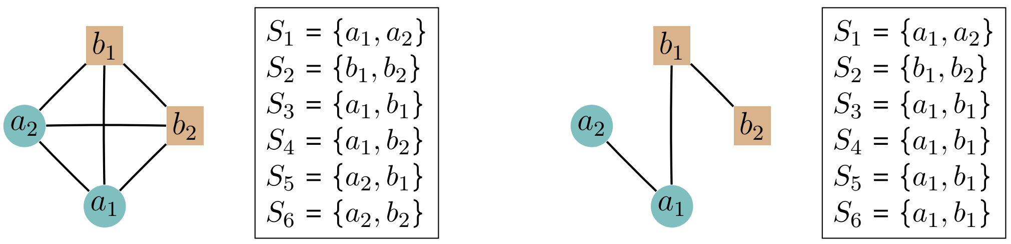
Analysing the problem and its solutions is fundamental to efficiently approach it with any potential way. In this section we start by studying in depth the structure of the solutions and providing a tight upper bound for the size of any optimal solution. This is an important result on which we later base the approximation algorithm that we devise. Following, we give some feasibility criteria for the instances and try to identify cycles in the solutions. In this way, it becomes possible to improve the lower bound on specific instances. Finally, we present a few other various results.

Some assumptions, about the instances and their solutions, which are not obvious but are crucial in the analysis, are the following two.
∘*Assumption 1:* A vector $M_i \in M$ is possible to be the same with one or more vectors $M_j \in M$. Such a vector M_i is considered different and cannot be, in general, eliminated.
∘*Assumption 2:* Suppose we have two or more vectors M_i, M_j, with $i \neq j$ which are are the same. Then it is possible to have $S_i = S_j$, meaning that they can be satisfied by the same set of vertices.

The importance of these assumptions and how they can affect the solution is illustrated in Figure 7. We are given an instance with:

$$V = T_1 \cup T_2, \quad T_1 = \{a_1, a_2\}, \quad T_2 = \{b_1, b_2\}$$

$M_1 = [2,0]$, $M_2 = [0,2]$, $M_3 = [1,1]$, $M_4 = [1,1]$, $M_5 = [1,1]$, $M_6 = [1,1]$



$S_1 = \{a_1, a_2\}$
$S_2 = \{b_1, b_2\}$
$S_3 = \{a_1, b_1\}$
$S_4 = \{a_1, b_2\}$
$S_5 = \{a_2, b_1\}$
$S_6 = \{a_2, b_2\}$

$S_1 = \{a_1, a_2\}$
$S_2 = \{b_1, b_2\}$
$S_3 = \{a_1, b_1\}$
$S_4 = \{a_1, b_1\}$
$S_5 = \{a_1, b_1\}$
$S_6 = \{a_1, b_1\}$

Figure 7: An instance of SU-SID and two different optimal solution, according to different assumptions.

It is easy to observe that when the vectors have to be satisfied by different vertex sets it is possible to have an optimal solution of K_V, as on the *left*. Therefore, the size of the solution has the same upper bound as SID. On the contrary, on the right side, we observe a completely different optimal solution. The difference is structural, as we prove right after.

3.1 Bounding solutions from above

An improved upper bound

Theorem 3.1 (Upper bound for optimal solutions)
Given an instance of SU-SID *with a set of vertices V partitioned into a set of types T, the size of an optimal solution $|E_{OPT}|$ is bounded by:*

$$|E_{OPT}| \leq \frac{|T| \times (|T| - 1)}{2} + |V| - |T| \tag{1}$$

Proof.
▷*Case* $|T| = 1$. As we have mentioned before, for $|T| = 1$, the problem is actually different. There is no uncertainty involved and by finding a spanning tree on the vertex set all the conditions will be satisfied and the problem will be solved. In any case (1) holds even for $|T| = 1$ since the solution will be a tree, the upper (and lower) bound for a solution will be $|V| - 1$.

▷*Case* $|T| = |V|$. In the case where $|T| = |V|$ every type of vertices is a singleton and it becomes obvious that no uncertainty is involved. In this case our problem becomes equivalent to SID. We know that in SID the upper bound is the complete graph K_V. Thus, replacing (1) with $|T| = |V|$, we have $|E_{OPT}| \leq \frac{|V| \times |V| - 1}{2}$, which holds.
∘*Note:* Such an instance for SID can be easily created if there exists a subset $S_i\ \forall\ (uv)$, where $u, v \in V$ and $u \neq v$.

▷*Case* $1 < |T| < |V|$. We give a constructive proof by a providing an algorithm which takes an instance of SU-SID and yields, in polynomial time, a feasible solution of size $|E_{SU-SID}| = \frac{|T| \times (|T| - 1)}{2} + |V| - |T|$. Therefore, since we give a feasible solution of that size, the optimal solution $|E_{OPT}|$ will always be smaller or equal, $|E_{OPT}| \leq |E_{SU-SID}|$, and thus (1) holds. □

A constructive proof

The algorithm that yields in polynomial time a feasible solution of that size can be describes as follows:

Algorithm 1: Solving SU-SID with $|E_{sol}| = \frac{|T|\times(|T|-1)}{2} + |V| - |T|$.

```
Result: Given a SU-SID instance with vertex set $V$ a partition $T$ and
        a matrix $M$, returns a graph $G = (V, E)$ and a collection $S$.
1  for each $T_j \in T$ do
2      select a vertex $u \in T_j$ as representative $r_j$
3      $l_j \leftarrow \max_{S_i \in S} m_{ij}$
4  $E_{sol} \leftarrow |K_T|$ edges between representatives.
5  for each $T_j \in T$ do
6      $E_{sol} \leftarrow E_{sol} \cup |l_j| - 1$ edges between $r_j$ and $u \in T_j$
7  for each $M_i \in M$ do
8      satisfy $M_i$ specifications by choosing connected vertices
9  for each $u \in V$ do
10     $con_u \leftarrow$ number of subsets $u$ contributes
11 for each $T_j \in T$ do
12     for each $u \in T_j$ with $con_u = 0$ do
13         select a subset $S_i$ where $m_{ij} > 0$
14         if $m_{ij} > 1$ then
15             select $v \in T_j$ where $v$ chosen by $S_i$, $v \neq r_j$ and $con_v > 1$
16             replace $v$ chosen by $S_i$ with $u$
17             $con_v = con_v - 1$, $con_u = con_u + 1$
18             $E_{sol} \leftarrow E_{sol} \cup (ur_j)$
19         else if $m_{ij} = 1$ then
20             replace $r_j$ chosen by $S_i$ with $u$
21             $con_{r_j} = con_{r_j} - 1$, $con_u = con_u + 1$
22             $E_{sol} \leftarrow E_{sol} \cup (ur_x)$, where $x \in T$ such that $m_{ix} > 0$
23 return $G = (V, E_{sol})$, $S$
```

Algorithm Explanation

• In lines $1 - 3$, the algorithm arbitrarily selects a vertex $u \in T_j$, $\forall T_j \in T$ as a *representative* r_j of that type. The value l_j, is the maximum number of vertices of T_j any subset requires.

• In line 4, a complete graph, K_T, is formed on all the *representatives* and the corresponding edges are added in the solution E_{sol}.

• In lines $5 - 6$, for each $T_j \in T$, $l_j - 1$ vertices are connected to r_j.

• In lines $7-8$, all the vectors are *satisfied* by *choosing* among the connected vertices. Obviously, at this point all the vectors can be *satisfied* since we have connected the maximum number of vertices presented for each type l_j. So, at this point we have acquired a collection S. Also, we have created a *1 to 1* connection between all types by forming the complete graph between

representatives.
• In lines $8-9$, value con_u is assigned to each vertex u, showing the number of subsets u *contributes*. All disjoint vertices will have a *contribution* of 0.
• Finally, in lines $11-22$, all the vertices are connected appropriately without affecting the subsets being *satisfied*. This is done as follows. For each disjoint vertex u, belonging to type T_j, we arbitrarily select one subset, say S_i, where at least one vertex of that type is required, $m_{ij} > 0$.
▷ If more than 1 vertices of T_j are required by S_i, in lines $14-18$, a vertex v is selected to replace u for the given subset. Vertex v must be different than the corresponding *representative*, $v \neq r_j$, and must contribute to more than one subset. After updating the vertex *contributions* vertex u is connected, by adding the edge (ur_j).
▷ In the case where only 1 vertex of T_j is required, in lines $19-22$, any *representative* r_x from the vertices already chosen by S_i, is selected, and then *replaces* r_j with u. Afterwards vertex u is connected by adding edge (ur_x) and the corresponding *contributions* are updated.

Algorithm illustration

In the following series of images we illustrate the algorithm step by step by applying it on an instance with $|V| = 12$, $|T| = 4$ and $|S| = 4$.

$$\boxed{\begin{array}{l} M_1 = [2,0,1,0] \\ M_2 = [0,1,1,1] \\ M_3 = [3,0,0,2] \\ M_4 = [1,2,1,2] \end{array}}$$

$$\boxed{\begin{array}{l} V = T_a \cup T_b \cup T_c \cup T_d \\ T_a = \{a_1, a_2, a_3, a_4\} \\ T_b = \{b_1, b_2, b_3\} \\ T_c = \{c_1, c_2\} \\ T_d = \{d_1, d_2, d_3\} \end{array}}$$

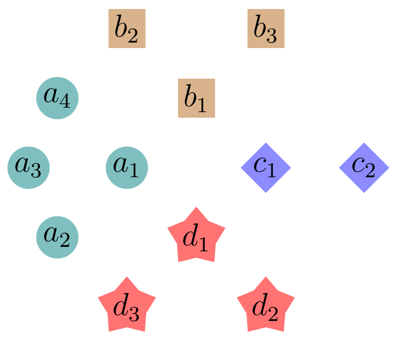


Figure 8: An instance of SU-SID with $|S| = 4$, $|V| = 12$, $|T| = 4$

In Figure 9, we observe that *representatives* have been chosen and the l_j values have been found. We also have the edge set E_{sol} consisting of $|K_T|$ and $l_j - 1$ edges of each type.

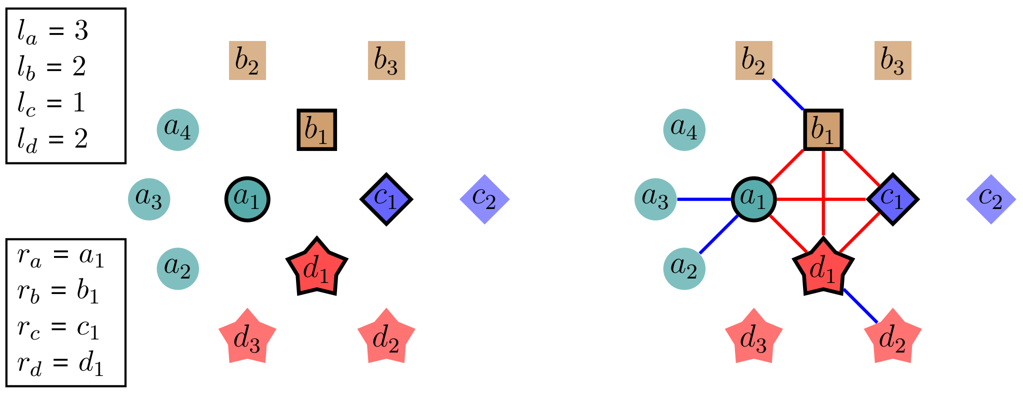


Figure 9: *Left* side illustrates lines $1-3$ of *Algorithm* 1. *Right* side illustrates lines $4-6$.

In Figure 10, we observe that there is a collection of subsets S which *satisfies* the specification matrix M, with the minimum number of vertices. Also the contribution of each vertex $u \in V$ has been calculated.

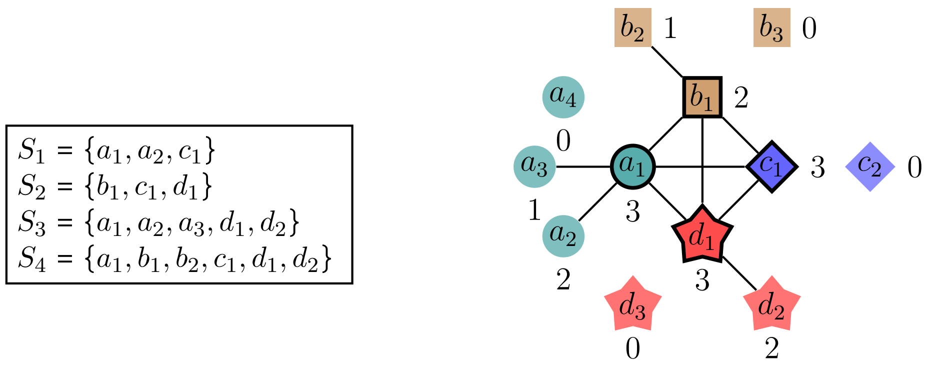
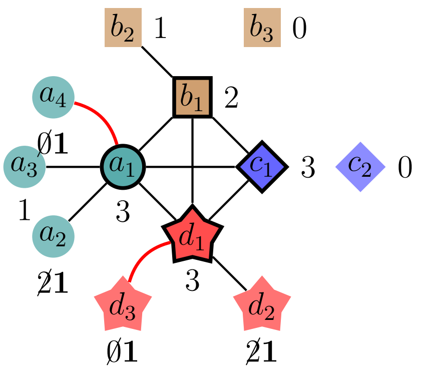
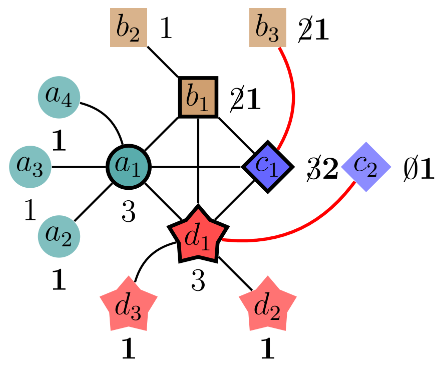

$S_1 = \{a_1, a_2, c_1\}$
$S_2 = \{b_1, c_1, d_1\}$
$S_3 = \{a_1, a_2, a_3, d_1, d_2\}$
$S_4 = \{a_1, b_1, b_2, c_1, d_1, d_2\}$
b_2 1
b_3 0
a_4 0
b_1 2
a_3 1
a_1 3
c_1 3
c_2 0
a_2 2
d_1 3
d_3 0
d_2 2

Figure 10: Illustration of lines $7-10$ of *Algorithm* 1.

Following, in Figure 11, is illustrated the last part of the algorithm. It is easy to observe the step by step transformation in the composition of the subsets and the *contribution* of each vertex in order to obtain a feasible solution. Both cases where $m_{ij} > 1$ and where $m_{ij} = 1$ are clearly depicted, on *left* and *right* side, respectively.



$S_1 = \{a_1, \cancel{a_2}, a_4, c_1\}$
$S_2 = \{b_1, c_1, d_1\}$
$S_3 = \{a_1, a_2, a_3, d_1, \cancel{d_2}, d_3\}$
$S_4 = \{a_1, b_1, b_2, c_1, d_1, d_2\}$



$S_1 = \{a_1, a_4, c_1\}$
$S_2 = \{\cancel{b_1}, b_3, c_1, d_1\}$
$S_3 = \{a_1, a_2, a_3, d_1, d_3\}$
$S_4 = \{a_1, b_1, b_2, \cancel{c_1}, c_2, d_1, d_2\}$

Figure 11: Illustration of lines $11-22$ of *Algorithm* 1. *Left* side depicts lines $14-18$ and *right* lines $19-22$.

Finally, in Figure 12 the solution yielded by *Algorithm* 1 is illustrated. We observe the collection S and graph $G = (V, E_{SOL})$, where $|E_{sol}| = \frac{|T|\times(|T|-1)}{2} + |V| - |T|$. Also, for the sake of completeness the final *contribution* of each vertex when the algorithm terminated is illustrated.

$S_1 = \{a_1, \mathbf{a_4}, c_1\}$
$S_2 = \{\mathbf{b_3}, c_1, d_1\}$
$S_3 = \{a_1, a_2, a_3, d_1, \mathbf{d_3}\}$
$S_4 = \{a_1, b_1, b_2, \mathbf{c_2}, d_1, d_2\}$

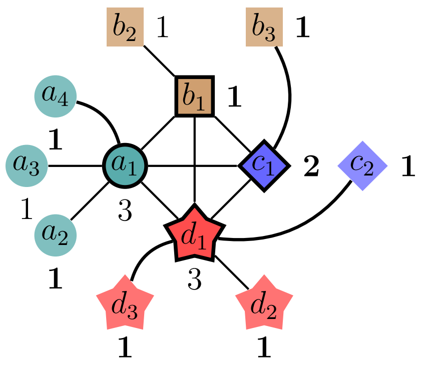


Figure 12: A solution of *Algorithm* 1 applied to the instance of Figure 8.

Correctness of the algorithm

Proposition 3.1: (Algorithm 1 correctness)
Algorithm 1 for SU-SID *yields always a feasible solution.*

Proof. To prove that a solution obtained from *Algorithm* 1 is feasible, we show that all constraints, as stated in Definition 2.2 are respected.
i) Each subset S_i satisfies the corresponding vector M_i. By line 8, we have a collection S where each S_i satisfies corresponding M_i. In lines 16 and 20 any *replacement* that takes place, is done only between vertices of the same type. Therefore, *satisfaction* of matrix M is not violated.
ii) Each induced subgraph $G[S_i]$ is connected. As previously, until line 8, before the *replacements* start $\forall\ S_i \in S$, $G[S_i]$ is connected. That connectivity is maintained after each *replacement.* This is because each new vertex $u \in T_j$ *chosen* will be directly connected with an edge (ur_x). And by the algorithm all *representatives* are *pairwise* connected. So, regardless if $r_x = r_j$ or else, the connectivity of $G[S_i]$ is not violated.
iv) Graph G is connected. The algorithm terminates only when $\forall\, u \in V, con_u > 0$. Value *con* shows the number of subsets a vertex has been *chosen* by. Since, each $G[S_i]$ is connected and $con_u \geq 1$, $\forall u \in V$, then G is connected.
v) Each edge $e \in E$ contributes to at least 1 subset. For the edges which connect vertices to *representatives*, this is true. Since, each such vertex u is *chosen* by at least 1 subset S_i, it is connected with exactly 1 edge to some *reprentative* r_x. Then edge (ur_x) will *contribute* to at least that subset S_i. For the edges between *representatives* this arguement is not completely true as we defined the algorithm. It is possible that some of those edges do not *contribute* to any subset. Nevertheless, it is *possible* to add a variable, denoted $count_e$ depicting the *contribution* $\forall\ e \in E$ and simply remove all edges where $count_e = 0$ in the end. We avoided it for the sake of simplicity. □

Tightness of the upper bound

Proposition 3.2: (Tightness of upper bound)
The upper bound for any optimal solution to SU-SID *provided in Theorem 3.1,* $\frac{|T|\times(|T|-1)}{2} + |V| - |T|$*, is tight.*

Proof. We will prove the tightness of the upper bound by giving a category of instances where the optimal solution, E_{OPT}, is always of that size, $|E_{OPT}| = \frac{|T|\times(|T|-1)}{2} + |V| - |T|$. Such an instance can be created as follows.
i) We take all $3-$ *combinations* of the set of types T. So we will have $\binom{|T|}{3}$ possible combinations.
ii) For each of these combinations, containing types T_i, T_j, T_k we will add 3 vectors M_a, M_b, M_c of the following form.

	$T_1, \ldots$	T_i	T_j	T_k	$T_{\vert T \vert}$
$M_a =$	$[0, \ldots$	$\vert T_i \vert$	$\vert T_j \vert$	0	$\ldots 0]$
$M_b =$	$[0, \ldots$	0	$\vert T_j \vert$	$\vert T_k \vert$	$\ldots 0]$
$M_c =$	$[0, \ldots$	$\vert T_i \vert$	0	$\vert T_k \vert$	$\ldots 0]$

So, we will have a specifications matrix M with $3 \times \binom{|T|}{3}$ vectors.
Finally, we will have an instance with V, T and a collection of size $|S| = 3 \times \binom{|T|}{3}$. Each triplet of such vectors will impose a cycle between vertices of the corresponding 3 types. This will be taking place until a cycle between each triplet of types has been created and the remaining vectors can be *satisfied* by those. So, since all the vertices will be connected we will have $|V| - 1$ edges. And since that many cycles will have been imposed, we will end up with a solution of size $|E_{OPT}| = \frac{|T| \times (|T|-1)}{2} + |V| - |T|$. □

∘*Note 1:* In such instances we can remove any duplicate vectors without affecting the solution.
∘*Note 2:* A more thorough understanding of such instances should come right after when we study this behaviour. This is done in order to infer instances where the lower bound of the size of the solutions is increased.

3.2 Feasibility, cycles and others

Feasibility of instances

Identifying the conditions an instance satisfies, in order to be feasible is important. Those conditions can be used in later proofs.

Lemma 3.1 (Feasibility conditions of instances)
A feasible SU-SID *instances satisfies the following conditions:*
i) $\qquad \forall T_j \in T : \qquad \sum_{S_i \in S} m_{ij} \geq |T_j|$
ii) $\qquad \sum_{S_i \in S} (\sum_{T_j \in T} m_{ij} - 1) \geq |V| - 1$

Proof. (i) We require that each vertex $u \in V$ is *chosen* by at least 1 subset $S_i \in S$. Therefore, it is easy to observe that for ever type T_j, $\sum_{S_i \in S} m_{ij} \geq |T_j|$ holds. *(ii)* We require that each edge $e = (uv)$ in the solution *contributes* to at least 1 subset $S_i \in S$. Equivalently, endpoints vertices u, v, must have been *chosen* both by at least 1 subset. We also observed, previously, that in an optimal solution, each induced subgraph $G[S_i]$ is a tree. Thus, it holds that $\forall S_i \in S$ each $G[S_i]$ has exactly $\sum_{T_j \in T} m_{ij} - 1$ edges. Since, the lower bound for a feasible solution is a tree, then all subsets must induce at least $|V| - 1$ edges and *(ii)* holds. □

We must note that these conditions are not sufficient for an instance to be feasible. For example, there is one case that is not covered by Lemma 3.1. These conditions do not avoid the creation of a forest. Such a solution is not feasible and an instance is illustrated below in Figure 13. We can observe that even though condition are satisfied, the edges are not sufficient to connect graph G , making the instance infeasible.

$$\boxed{\begin{aligned} M_1 &= [2,0,0,0] \\ M_2 &= [3,0,0,1] \\ M_3 &= [0,2,0,0] \\ M_4 &= [0,3,0,1] \\ M_5 &= [0,0,2,0] \\ M_6 &= [0,0,3,1] \end{aligned}}$$

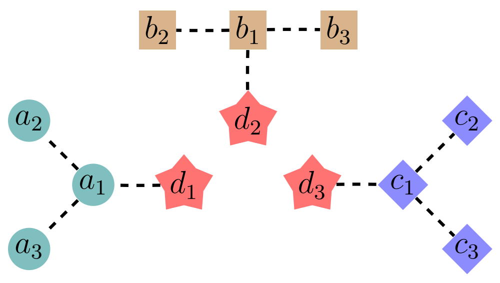


Figure 13: An infeasible SU-SID instance leading to a forest

We conclude that, knowing in advance if an instance is infeasible is important. Finding a way to do so, may not be trivial. So, it remains open how we can efficiently express all feasibility conditions for SU-SID instances.

Identifying cycles

As mentioned due to the complexity of SU-SID we do not expect to be solving instances to optimality, after some particular size. So, it is likely that we will be obtaining approximate solutions. Until now we have given an upper bound for the size of the optimal solutions. Moreover, we have shown the trivial lower bound of $|E_{sol}| \geq |V| - 1$. But we have seen that in many instances the optimal solution is greater than $|V| - 1$.
It is very important to be in place to identify in which cases the optimal solution of an instance is greater than $|V| - 1$. In this way it is possible to have a better understanding of the quality of the solutions we obtain. Since, in that way we will be comparing our solution with a tighter lower bound than that yielded by the graph connectivity constraint.

Towards that direction we present some preliminary results and give two conjectures about the creation of cycles in the solution. We believe the conjectures are true and it would be high interest to prove them. Obviously, each cycle in the optimal solution, increases $|E_{opt}|$ by 1. The optimality gap between an obtained solution will be respectively decreased.

Conjecture 3.1 (Size of the optimal solution)
Given a SU-SID *instance, if* $\forall\, T_j \in T : m_{ij} < |T_j|$, $\forall M_i \in M$, *then for the optimal solution,* $|E_{opt}| = |V| - 1$ *holds.*

SU-SID has a high complexity due to large number of possible combinations for each subset as seen in a previous section. All vertices of a type $u \in T_j$ are equivalent, so there exists a high symmetry between the possible solution. So, from a different perspective, the larger number of possible combinations can be seen as a bigger *flexibility* in finding solutions of smaller size. This perspective can also be justified by Theorem 3.1, where the higher the *uncertainty* the more bounded the size of an optimal solution is. What Conjecture 3.1 represents, is our strong intuition that only such cases can constrain that *flexibility.* So, only when there are subsets that require the maximum number of vertices of some type, can additional edges to the optimal solution be imposed.

Proving Conjecture 3.1 is of high interest. It is also highly interesting to specify more the cases were cycles are created, and identify them. In this direction lay the next observations. We gradually note down *patterns* or submatrices in M, that create cycles in the optimal solutions.

In Proposition 3.2 we proved the tightness of the upper bound by a providing a category of instances where the optimal solution has size of $|E_{OPT}| = \frac{|T|\times(|T|-1)}{2} + |V| - |T|$. But we actually observe, that it is only sufficient for each $m_{ij} = |T_j|$ to appear only once as in the following table.

	$T_1, ...$	T_i	T_j	T_k	$T_{\|T\|}$
$M_a =$	$[0, ...$	$\|T_i\|$	$1 \leq m_{aj} \leq \|T_j\|$	0	$...0]$
$M_b =$	$[0, ...$	0	$\|T_j\|$	$1 \leq m_{bk} \leq \|T_k\|$	$...0]$
$M_c =$	$[0, ...$	$1 \leq m_{ci} \leq \|T_i\|$	0	$\|T_k\|$	$...0]$

Moreover, it seems that this is not so strict as well. As we can see in the following table, more columns which are part of these submatrices can have m_{ij} values with $1 \leq m_{ij} \leq |T_j|$, for some M_i and T_j.

	$T_1, ...$	T_i	T_j	T_k	$T_{\|T\|}$
$M_a =$	$[0, ...$	$\|T_i\|$	$m_{aj} \geq 1$	$m_{ak} \geq 1$	$...0]$
$M_b =$	$[0, ...$	$m_{bi} \geq 1$	$\|T_j\|$	$m_{bk} \geq 1$	$...0]$
$M_c =$	$[0, ...$	$m_{ci} \geq 1$	$m_{cj} \geq 1$	$\|T_k\|$	$...0]$

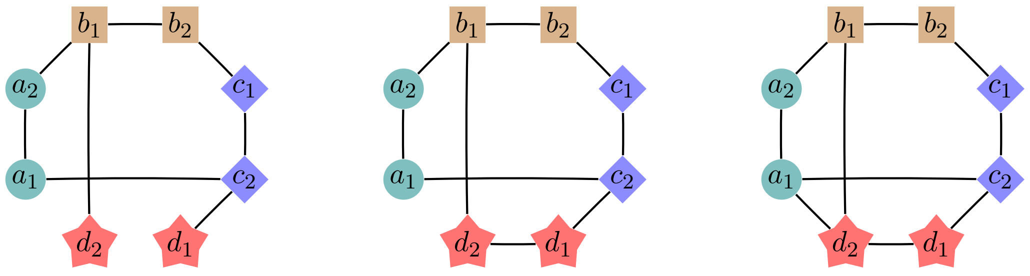
Finally it is possible for a cycle to be created among more than three types, as long as these vectors do not include all types $T_j \in T$.

	$T_1, \dots$	T_i	T_j	T_k	T_l	$T_{\|T\|}$
$M_a =$	$[0, \dots$	$\|T_i\|$	$m_{aj} \geq 1$	0	0	$\dots 0]$
$M_b =$	$[0, \dots$	0	$\|T_j\|$	$m_{bk} \geq 1$	0	$\dots 0]$
$M_c =$	$[0, \dots$	0	0	$\|T_k\|$	$m_{cl} \geq 1$	$\dots 0]$
$M_d =$	$[0, \dots$	$m_{di} \geq 1$	0	0	$\|T_l\|$	$\dots 0]$

Conjecture 3.2 (Cycles in the optimal solution)
Suppose a set of vectors in M creates a cycle in the optimal solution. This cycle is also preserved after any rows (or columns) are removed (or added).

This conjecture states that, if some vectors lead to the creation of a cycle between some types in the optimal solution, this is independent of the other vectors and types. In that way, if the submatrices in M that lead to cycles are strictly defined, possibly we will be able to identify them in advance. Of course when any column or row is removed, we should always respect the feasibility of the instance.

In Figure 14 three SU-SID instances, where cycles are part of the optimal solution, are illustrated. For the instances, $V = T_a \cup T_b \cup T_c \cup T_d$ with $T_a = \{a_1, a_2\}$, $T_b = \{b_1, b_2\}$, $T_c = \{c_1, c_2\}$ and $T_d = \{d_1, d_2\}$



$M_1 = [2,1,0,0]$
$M_2 = [0,2,1,0]$
$M_3 = [1,0,2,0]$
$M_4 = [0,0,1,1]$
$M_5 = [0,1,0,1]$

$M_1 = [2,1,0,0]$
$M_2 = [0,2,1,0]$
$M_3 = [1,0,2,0]$
$M_4 = [0,0,2,1]$
$M_5 = [0,1,0,2]$

$M_1 = [2,1,0,0]$
$M_2 = [0,2,1,0]$
$M_3 = [1,0,2,0]$
$M_4 = [0,0,2,1]$
$M_5 = [0,1,0,2]$
$M_7 = [2,0,1,0]$
$M_8 = [1,0,0,2]$

Figure 14: Instances with cycles in their optimal solutions

We observe how the aforementioned submatrices *(or patterns)* that exist in M, cause the creation of cycles between the respective types. On the left instance, we have a cycle between T_a, T_b and T_c caused by vectors M_1, M_2 and M_3. Respectively for the other two instances, a triple of vectors creates a cycle between 3 types. We most note that on the right instance, the optimal solution has reached the upper bound given by Theorem 3.1. Therefore, any additional vectors leading to a new potential cycle, e.g. between T_a, T_b, T_d would be *satisfied* by existing edges, in an optimal solution.

Other results

Corollary 3.2 (From Theorem 3.1)
Given an instance of SU-SID *with a set of vertices V partitioned into two types, $|T| = 2$, the optimal solution will always be a tree. Thus, $|E_{OPT}| = |V| - 1$, and such a solution can be found in polynomial time.*
Proof. From Theorem 3.1, and for $|T| = 2$ we have $|E_{OPT}| \leq |V| - 1$. Since, it also holds that $|E_{OPT}| \geq |V| - 1$, we conclude that $|E_{OPT}| = |V| - 1$, when $|T| = 2$. Thus the graph is a tree and such a solution can be found in polynomial time by applying Algorithm 1. □

What this corollary states is that our problem is $\mathcal{NP} - hard$ for $|T| \geq 3$. This was not obvious at first, since we expected the opposite behaviour for $|T| = 2$. We deduce that the more *uncertainty* there is, the more complex it is to find an optimal solution. But on the other hand, the size of such a solution is more bounded.

Conjecture 3.3 (Intra-type connectivity)
Given an optimal solution to the SU-SID *problem, the induced subgraph $G[T_j]$, $\forall\, T_j \in T$, does not contain any cycles.*
Our intuition is that, supposedly there was one cycle. There would always be a step of transformations that could be done to an edge of that cycle. A transformation would be a change in the composition of some subsets. Possibly with the *choosing* of another endpoint vertex u neighbouring the endpoint vertex of the edge which was removed.

This proposition gives some valuable information regarding the structure of the solutions. Knowing that there do not exist cycles between vertices of any type $T_j \in T$, is useful and can be potentially be exploited.

4 MILP formulation

In this section we model our problem using Mixed Integer Linear Programming, denoted *MILP*. In general solving MILP in polynomial time is not possible, and therefore has its limitations. Nevertheless, this approach makes it possible to solve the problem to optimality. We start by presenting the mathematical model and explaining it in depth. Following, we make some important observation on the formulation we followed. Finally, we give some formulations of additional constraints which can be of use with respect to its applications. We must note that this is the model implemented and later used for our experiments which are presented in a next section.

4.1 The model

For the modelling we extend the existing MILP formulation for SID, which is based on flows [2]. First we present the variables used to model the problem, then presents the constraints of our formulation and explain them.

$$y_e = \begin{cases} 1 & \text{if edge } e \text{ is selected.} \\ 0 & \text{otherwise.} \end{cases}$$

$$y_e^i = \begin{cases} 1 & \text{if edge } e \text{ belongs to } G[S_i]. \\ 0 & \text{otherwise.} \end{cases}$$

$$z_u^i = \begin{cases} 1 & \text{if vertex } u \text{ is } \textit{chosen} \text{ by subset } S_i. \\ 0 & \text{otherwise.} \end{cases}$$

$$s_u^i = \begin{cases} 1 & \text{if vertex } e \text{ is the source of flow for } G[S_i]. \\ 0 & \text{otherwise.} \end{cases}$$

f_a^i : quantity of flow originating from σ_i and going through arc a.

We denote by $A_i^+(u)$, (resp.$A_i^-(u)$), the subsets of arcs of $D[S_i]$ exiting, (resp. entering) node u, and by $|S_i|$ the total number of vertices of each type required by each vector M_i, $|S_i| = \sum_{T_j \in T} m_{ij}$. For the sake of the connectivity of the graph G we arbitrarily choose a $u \in V$ as a source and denote it *src*.

The formulation, given the above notations and assumptions, is as follows:

$$\text{minimize} \quad \sum_{e \in E} y_e \tag{1}$$

subject to:

$$\sum_{u \in T_j} z_u^i = m_{ij} \quad , \forall\ T_j \in T,\ S_i \in S \tag{2}$$

$$\sum_{u \in V} s_u^i = 1 \quad , \forall\ S_i \in S \tag{3}$$

$$\sum_{a \in A(u)^+} f_a^i - \sum_{a \in A(u)^-} f_a^i = (|S_i| \cdot s_u^i) - z_u^i \quad , \forall\ u \in V,\ S_i \in S \tag{4}$$

$$\sum_{a \in A(u)^+} f_a - \sum_{a \in A(u)^-} f_a = \begin{cases} |V| - 1 & \text{if } u = src \\ -1 & \text{if } u \neq src \end{cases} \quad , \forall\ u \in V \tag{4a}$$

$$f_a^i \leq |S_i| \cdot y_{e(a)}^i \quad , \forall\ a \in A,\ S_i \in S \tag{5}$$

$$f_a \leq |V| \cdot y_{e(a)} \quad , \forall\ a \in A \tag{5a}$$

$$y_{uv}^i \leq z_u^i \quad , \forall\ (uv) \in E,\ S_i \in S \tag{6}$$

$$y_{uv}^i \leq z_v^i \quad , \forall\ (uv) \in E,\ S_i \in S \tag{7}$$

$$y_e \geq y_e^i \quad , \forall\ S_i \in S,\ e \in E \tag{8}$$

$$y_e \leq \sum_{S_i \in S} y_e^i \quad , \forall\ e \in E \tag{9}$$

Solving an instance of SU-SID problem consists of minimizing the number of edges while having a feasible solution. Therefore, we have introduced one binary variable y_e for each possible edge $e = (uv)$ of the undirected complete graph on $|V|$ vertices. Objective function (1) expresses this minimization of the edge variables y. We also form the directed graph $D = (V, A)$ where two arcs $a = (u, v)$ and $a' = (v, u)$ replace each edge $e = (uv)$, which is necessary to express the flow. We denote by $e(a)$ the edge that corresponds to arcs a and a'. Thus, a feasible solution must satisfy the following:

• To ensure the *correctness of subsets* we have introduced the z binary variables, for every vertex u and each subset S_i. This is done in Equation (2) where it is checked that every subset $S_i \in S$ has the exact number of vertices of each type $T_j \in T$, as specified by the subsets specifications matrix M.

• As mentioned, since we have a flow-based formulation, we need a source of flow for each subset. As, we do not know the exact composition of each subset we cannot assign any vertex as a source in advance. So, we introduce the s binary variable, for each vertex u and each subset S_i . With Equation (3) we ensure that in each subset exactly one vertex u will be chosen as a source. This is necessary, as otherwise we could have two ore more connected components in each subset, thus violating the subset connectivity constraint.

• To enforce the *connectivity of subsets*, we provide each source with flow that must reach all other vertices *chosen* in S_i, by using only arcs on $D[S_i]$. The continuous variables $f_a^i \in \mathbb{R}$ are introduced for this reason. They express the quantity of flow of each subset S_i going through arc a. Equation 4 expresses the flow conservation constraint. If the vertex is *chosen* and is not the source then we will have 1 unit of flow collected. If the vertex is *chosen* and is also the source then $|S_i| - 1$ units of flow will originate from that vertex. In any other case the value will be 0 meaning that it will not affect the flow occurring for S_i.

• Inequality (5) expresses two main things. Firstly, it expresses the *capacity constraints*. Meaning that, no flow can use arc a when the corresponding edge is not chosen and that the flow circulating along any subgraph $G[S_i]$ never exceeds the number of vertices in that subgraph, $|S_i|$. Secondly, the *symmetry* that if there is some flow through the arcs $a = (u, v)$ or $a' = (v, u)$, only then the corresponding edge $e = (uv)$ can be selected by a subset.

• Constraints (4a) and (5a) similarly to (4) and (5) satisfy the connectivity of the graph of the solution $G = (V, E)$. As mentioned *src* can be any vertex. So, we have a flow of $|V| - 1$ units originating from *src*, and each vertex $u \neq src$ collects 1 unit.

• Inequalities (6) and (7) make sure that vertices u and v are *chosen* by a subset S_i if the corresponding edge $e = (uv)$ *contributes* to that subset.

• Constraint (8) ensures that if an edge e *contributes* to one or more subsets, then e will be part of the solution.

• Inequality (9) assures that an edge e which is part of the solution must *contribute* to at least one subsets as required by the definition of the problem.

4.2 Observations and additional formulations

Notes on the model

∘*Note 1:* Constraints (4a) and (5a) are necessary, as mentioned, to ensure the connectivity of the graph $G = (V, E)$. A thought for simplifying the formulation by adding one extra vector/subset containing all vertices, $M_{all} = [|T_1|, |T_2|, ..., |T_r|]$, is wrong. This is because, it can lead to the *false-positive* edges which were explained in the problem definition.

∘*Note 2:* In an extended model we would also have a real variable $x_a^i \in [0, 1]$. This would be positive if arc a carried flow for subset S_i. In that way the *symmetry* and *capacity constraints* would be separated and be easier to observe. It was redundant so we have ignored it. Nevertheless, for the sake of completeness we show how Inequality (5), resp. (5a), would be replaced.

$$f_a^i \leq |S_i| \cdot x_a^i \qquad , \forall\ a \in A,\ S_i \in S$$
$$x_a^i \leq y_{e(a)}^i \qquad , \forall\ a \in A,\ S_i \in S$$

∘*Note 3:* The formulation can be turned into a decision formulation. This can be done by removing the objective function (1) and by introducing Inequality (10). This will give a feasible solution with k or less edges, if one exists.

$$\sum_{e \in E} y_e \leq k \tag{10}$$

∘*Note 4:* Moreover, by using the decision formulation it is possible to enumerate all feasible solutions of size k or less. This can be done by adding Constraint (11). Every time a solution is found with $|E_{sol}| < k$ we should add it to $\mathcal{S}$, which represents the solutions found so far. If we repeat the procedure until no feasible solutions exist, we will have acquired all solution $|E_{sol}| < k$. We must note that with (11) two solutions are considered different with respect to the edge set E, ignoring the subset composition S.

$$\sum_{e \in E_{sol}} y_e < k \qquad , \ \forall\ E_{sol} \in \mathcal{S} \tag{11}$$

Additional constraints

We present a few extra constraints that have been implemented and can be useful additions to the model according to the application.

The first one, bounds the *maximum degree* of graph G, $\Delta(G)$, by a value, denoted *maxDEG*. It refers to the maximum number of neighbours of any vertex in the graph. This can be useful in the context of Structural Biology where we can expect a solution to have a bounded degree, due to natural space limitations. It can be represented with the following constraint.

$$\sum_{v \in V} y_{(uv)} \leq maxDEG \qquad , \forall\ u \in V \tag{12}$$

For the same motivation, it can be of use to bound the maximum number of types of vertices any vertex is neighbouring with by a value, denoted $maxNT$. To make this possible we have to introduce a new binary variable n_u^j and two constraints, (13) and (14), as follows.

$$n_u^j = \begin{cases} 1 & \text{if vertex } u \text{ neighbours with a vertex of type } T_j. \\ 0 & \text{otherwise.} \end{cases}$$

$$\sum_{T_j \in T} n_u^j \leq maxNT \qquad , \forall\ u \in V \tag{13}$$

$$y_{(uv)} \leq n_u^j \qquad , \forall\ v \in T_j,\ T_j \in T,\ u \in V \tag{14}$$

Finally, it is possible to bound the diameter D of the graph G, by a value, denoted $maxD$. The diameter can be defined as the greatest distance between any pair of vertices. We have used the formulation presented in previous work [18], which we present. We need to define a binary variable ${}_vp_u^i$ and an integral variable h_u^i, and four additional constraints, (15) - (18).

$${}_vp_u^s = \begin{cases} 1 & \text{if } u \text{ is parent of } v \text{ in the spanning tree rooted at } s. \\ 0 & \text{otherwise.} \end{cases}$$

h_u^s : the *height* of u in the spanning tree rooted at s, $h_u^s \in \mathbb{Z}^+$.

$$\sum_{u \in V} {}_vp_u^s = \begin{cases} 0 & \text{if } v = s \\ 1 & \text{if } v \neq s \end{cases} \qquad , \forall\ v \in V \forall\ s \in V \tag{15}$$

$$h_u^s - h_v^s + (maxD + 1) \cdot {}_vp_u^s \leq maxD \cdot y_{e(a)}^i \qquad , \forall\ u, v \in V, \forall\ s \in V \tag{16}$$

$$h_u^s \leq maxD \qquad , \forall\ u \in V, \forall\ s \in V \tag{17}$$

$${}_vp_u^s \leq y_{uv} \qquad , \forall\ u, v \in V, \forall\ s \in V \tag{18}$$

The main idea of the formulation is that for every vertex u we consider a spanning tree rooted in, $u = s$. Thus, by constraining that no tree has more height than $maxD$, we force graph G to have a diameter at most $maxD$.

An alternative modelling

While modelling the problem, the main difficulty consists of satisfying the connectivity of each induced subgraph $G[S_i]$. It gets even more complicated as we do not know in advance the exact composition of each subset in matter of vertices. On the proposed model we decided to add the extra variable s_u^i, to represent if a vertex u is chosen as the source of subset S_i. So, in total we add $|V| \times |S|$ variables and $|S|$ constraints.
Another alternative way to handle the *subset uncertainty* for the connectivity is the following. We construct an *extended* graph by adding $\forall\ S_i \in S$ a *meta-source*, denoted σ_i. Then $\forall\ S_i$ we select one type of vertices, denoted $T(\sigma_i)$, and add an edge $e = (u\sigma_i)$, $\forall u \in T(\sigma_i)$. So we now have a graph $G' = (V', E')$, where $|V'| = |V| + |S|$ and $|E'| = |K_V| + |S| \cdot |V|$.

Afterwards, Equations (3) and (4) should be replaced with the following.

$$\sum_{u \in T_{(\sigma_i)}} y_{\sigma_i u} = 1 \qquad , \forall\ S_i \in S \tag{3}$$

$$\sum_{a \in A(u)^+} f_a^i - \sum_{a \in A(u)^-} f_a^i = \begin{cases} |S_i| & \text{if } u = \sigma_i \\ -z_u^i & \text{if } u \neq \sigma_i \end{cases} \qquad , \forall\ u \in V,\ S_i \in S \tag{4}$$

Equation (3) now represents that only one edge should exist between the source of each subset so that the corresponding subgraph will be connected. And, Equation (4) as before represents that from the source $|S_i|$ units of flow are emitted. Similarly, each other vertex collects 1 unit of flow. Finally, by the end, the *meta-sources* should not be considered in the solution.

We observe, that there is a trade-off between the two formulations which lead to the same results. The first one has a larger set of variables but the other is solved on an *extended* graph. It is not clear which one of the formulations is *better*, and in fact it seems to depend on the instances. Finally, we should note that there can be possibly more different formulations, as graph connectivity is a well-studied problem in the literature.

5 An approximation algorithm

In this section we propose an approximation algorithm for SU-SID. Initially, we give some definitions and algorithms which will be used as a part of our main algorithm. Following, we present the algorithm, explain it and illustrate it with a small instance. Finally, we analyse the algorithm, by proving its correctness, its complexity and looking into its approximation ratio.

5.1 Preliminary definitions

Representability

In this sections the notions of *represantibility* and *representative* vertices are expressed which is the main concept that is later used in the algorithm.

Definition 5.1 (Representability criterion)
We are given a SU-SID *instance with a specifications matrix M and a vertex set V partitioned into T types. A type of vertices T_j will be called* ***representable*** *if it satisfies the following criterion:*

$$\sum_{S_i \in S : m_{ij} > 1} (m_{ij} - 1) \geq |T_j| - 1$$

∘*Note:* If T_j is *representable* $\forall\, T_j \in T$, then the instance is *representable.*

That criterion indicates that there exists a feasible solution to an instance, where a vertex $u \in T_j$ is *chosen* by every $S_i \in S$ with $m_{ij} > 1$.
Suppose we have four instances with $|T_j| = 5$ and the following matrices M.

Instance 1:
$M_1 = [..., 1, ...]$
$M_2 = [..., 1, ...]$
$M_3 = [..., 1, ...]$
$M_4 = [..., 1, ...]$
$M_5 = [..., 1, ...]$

Instance 2:
$M_1 = [..., 3, ...]$
$M_2 = [..., 2, ...]$
$M_3 = [..., 1, ...]$
$M_4 = [..., 1, ...]$
$M_5 = [..., 1, ...]$

Instance 3:
$M_1 = [..., 3, ...]$
$M_2 = [..., 2, ...]$
$M_3 = [..., 1, ...]$

Instance 4:
$M_1 = [..., 3, ...]$
$M_2 = [..., 2, ...]$
$M_3 = [..., 3, ...]$

Figure 15: Illustration of 4 different instances, 2 *representable* and 2 not.

We have $|T_j| - 1 = 4$ and $\sum_{\substack{S_i \in S: \\ m_{ij} > 1}} (m_{ij} - 1) = 0, 4, 3, 5$ respectively. Therefore, according to Definition 5.1, instances $2, 4$ are *representable* and $1, 3$ not.

Choosing types representative-wise

Definition 5.2 (Rep-wise *choosing*)
We are given a SU-SID *instance with a specifications matrix M and a partition of vertices T. For a type T_j a vertex $u \in T_j$ is arbitrarily selected. Vertex u is called representative, denoted r_j. In a collection S, a type T_j will have been* ***rep-wise chosen*** *if the following hold:*
i)*It satisfies the corresponding column M_j of matrix M according to* SU-SID *Definition .*
ii)*The representative r_j is chosen by the maximum number of subsets.*
iii)*A vertex $u \in T_j \setminus \{r_j\}$ chosen by a subset S_i where $r_j \notin S_i$, cannot be any other subset S_k where $r_j \in S_k$.*

Having types of vertices *rep-wise chosen* prosaically means that, if a type is *representable* it is simply *chosen* by each subset $S_i \in S : m_{ij} > 1$. If it is not, it is *chosen* by the most subsets possible while maintaining feasibility.
Below we consider the instances depicted in Figure 15 and we illustrate how they would be if they were *rep-wise chosen.* Partition is $T_j = \{u_1, u_2, u_3, u_4, u_5\}$ and u_1 is selected as the *representative* r_j.

Inst. 1:	*chosen*
$m_{1j} = 1$	$\mathbf{r_j}$
$m_{2j} = 1$	u_2
$m_{3j} = 1$	u_3
$m_{4j} = 1$	u_4
$m_{5j} = 1$	u_5

Inst. 2:	*chosen*
$m_{1j} = 3$	$\mathbf{r_j}, u_2, u_3$
$m_{2j} = 2$	$\mathbf{r_j}, u_4$
$m_{3j} = 1$	u_5
$m_{4j} = 1$	$\mathbf{r_j}$
$m_{5j} = 1$	$\mathbf{r_j}$

Inst. 3:	*chosen*
$m_{1j} = 3$	$\mathbf{r_j}, u_2, u_3$
$m_{2j} = 2$	$\mathbf{r_j}, u_4$
$m_{3j} = 1$	u_5

Inst. 4:	*chosen*
$m_{1j} = 3$	$\mathbf{r_j}, u_2, u_3$
$m_{2j} = 2$	$\mathbf{r_j}, u_4, u_5$
$m_{3j} = 3$	$\mathbf{r_j}, u_2$

Figure 16: Type T_j of Figure 15 instances being *rep-wise chosen.*

Corollary 5.1 (Number of subsets containing the *representative*)
Given a SU-SID *instance, we have a type of vertices T_j which is rep-wise chosen. The corresponding representative r_j will be chosen by:*

$$\mathbf{max}\left(\sum_{\substack{S_i \in S: \\ m_{ij} > 0}} 1 \; , \; \sum_{\substack{S_i \in S: \\ m_{ij} > 1}} (m_{ij} - 1) + 1 \right)$$

Proof. The left part of the function is straightforward. It is the case of *representable* types, where by definition there exists a feasible collection S where r_j can be *chosen* by each subset $S_i : m_{ij} > 0$. The right part refers to the *non-representable* types. By the criterion for *representability* the lower bound for a type T_j to be *representable* is $\sum_{S_i \in S: m_{ij} > 1} (m_{ij} - 1) = |T_j| - 1$. We observe that the difference from this bound corresponds to the number of subsets $S_i \in S : m_{ij} > 0$, not *choosing* r_j. $\square$

Following we present an algorithm and briefly explain a polynomial algorithm for having a type of vertices *rep-wise chosen.*

Algorithm 2: (repWise) Having a type of vertices *rep-wise chosen.*

```
Result: Given a SU-SID instance with a type $T_j$ and a specifications
        matrix $M$, it return a collection of subsets $S_j^*$ where $T_j$ is
        rep-wise chosen.
1  $counter, pointer \leftarrow 0$
2  $listVert \leftarrow list[T_j \setminus \{r_j\}]$
3  $allPlaced \leftarrow False$
4  $noReps_j \leftarrow |S| - |T_j| + \sum_{S_i \in S : m_{ij} > 1} (m_{ij} - 1) + 1$
5  for $S_i \in S : m_{ij} > 0$ with decreasing order (w.r.t $m_{ij}$) do
6      if $counter < noReps_j$ then
7          $S_i \leftarrow r_j$
8          $counter \leftarrow counter + 1$
9      if $allPlaced$ is True then
10         $pointer \leftarrow 0$
11     while $|S_i| < m_{ij}$ do
12         $u \leftarrow listVert[pointer]$
13         $S_i \leftarrow S_i \cup u$
14         if $pointer = |T_j| - 2$ then
15             $allPlaced \leftarrow True$
16             $pointer \leftarrow 0$
17         else
18             $pointer \leftarrow pointer + 1$
19 return $S_j$
```

Algorithm Explanation

• In line $1-4$, the initialization takes place. *Counter* indicates the number of subsets that have *chosen* a *representative* at each step. *Pointer* shows at each step which vertex has to be *chosen* by the subset. *listVert* a list of vertices including all $u \in T_j \setminus \{r_j\}$. *allPlaced* is a flag which will be *True* when each vertex $u \in T_j$ has been *chosen* at least once. Finally, *noReps*$_j$ represents the number of subsets that will *choose* a *representative*, obtained from Corollary 5.1.

• In line 5, we proceed by *choosing* vertices for each subset S_i one by one in a decreasing order, with respect to m_{ij}, in line 6.

• Lines $6-8$, represent that exactly *noReps*$_j$ subsets will choose as a first vertex the representative r_j.

• At some point, each vertex $u \in T_j$ will be *chosen* by one subset. Then *pointer* will indicate that vertices should be *chosen* from the beginning. This is shown in lines $9-10$.
• Afterwards, in lines $11-18$, vertices are added one by one until each S_i is *satisfied.* More specifically, in lines $12-13$, vertex u where the *pointer* currently indicates, is added. If the *pointer* reaches the end of *listVert*, then this means that all the *vertices* have been chosen and thus *allPlaced* becomes *True.* This is shown in lines $14-18$. We should note that his can happen only once, and the limit is $|T_j|-2$, as r_j has not been in *vertList* at first place.

Lemma 5.1 (Algorithm repWise complexity)
Algorithm repWise has a $\mathcal{O}(|T_j||S|)$ time complexity. Moreover, Applying the algorithm to the set of types of vertices T has $\mathcal{O}(|V||S|)$ complexity.
Proof. It is easy to observe that each $S_i \in S$, in lines $5-18$, is considered only once in each iteration. Also, in each iteration at most $|T_j|$ vertices are considered. So, the complexity is $\mathcal{O}(|T_j||S|)$.
In order to *rep-wise choose* all types $T_j \in T$ of an instance we have to apply the algorithm $|T|$ times. But since T is partition of V, each vertex $u \in V$ will be considered only in execution of the algorithm. So, it is clear that the complexity is $\mathcal{O}(|V||S|)$. □

Finding collections with representatives

<u>**Definition 5.3**</u> (1-rep property)
In the SU-SID *context we will say that a collection of subsets S, satisfying the specification matrix M, has the* ***1-rep property*** *if:*
i) Each type of vertices $T_j \in T$ is rep-wise chosen.
ii) Each subset $S_i \in S$ has chosen at least one representative.

So, a collection S has the *1-rep property* if every type is *rep-wise chosen* and the *representatives* are *chosen* in such a way that each subset S_i has one. Following we give a conjecture on the existence of the *1-rep property* and an intuition on the proof.

Conjecture 5.1 (Existence of the *1-rep property*)
Each feasible SU-SID *instance has the 1-rep property.*
Our intuition is that the proof is made possible through the conditions for feasibility of an instance. This is possibly done by taking the *tightest* instances, which are closer to being infeasible and handling appropriate the constraints. Then we can reduce any other instance to one of those *tight* ones. We observe that even in the *tightest* instances we construct in order to

verify the existence of the *1-rep property* there is a large number of pair-wise transformations. Perhaps it is not possible to completely prove this conjecture if the all the infeasibility conditions have not been identified before.

Following, we show a simple algorithm to find a collection S with the *1-rep property* in polynomial time.

Algorithm 3: (1-rep) Finding a collection with the *1-rep property.*

```
Result: Given a collection $S^*$ where each type $T_j$ is rep-wise chosen,
        it returns a collection $S$ with the 1-rep property.
1 for $S_i \in S^*$ do
2     $reps_i \leftarrow \{representatives \ \in S_i\}$
3 while $\exists\, S_i$ with $|reps_i| = 0$ do
4     for $T_j \in T$ where $m_{ij} = 0$ do
5         for $S_k \in S$ where $|reps_i| > 1$ do
6             if $r_j \in S_k$ then
7                 swap($r_j$, $u \in T_j \cap S_i$)
8                 update($reps_i$), update($reps_k$)
9 return $S$
```

Algorithm Explanation

• The algorithm starts by creating a set, denoted *reps*$_i$ of the *representatives* already *chosen* by each subset S_i. Afterwards, until each S_i has one *representative* all the other subsets with are examined.

• When a subset S_k, with more than two *representatives* is found, vertices are *swapped.* This means that S_i *chooses* r_j removing a vertex $u \in T_j$ and similarly S_k. Finally, the corresponding set of *representatives* are updated.

Remark 5.1 (Algorithm 3 correctness)
The correctness of the algorithm follows Conjecture 5.1. There always exists a collections S having the *1-rep property* and there is a finite number of possible pair-wise swaps. Since, the algorithm examines all possible pairwise swaps, eventually, such a collection S will be found.

Lemma 5.2 (Algorithm 3 complexity)
Algorithm 1-rep has a $\mathcal{O}(|S|^2|T|)$ time complexity.
Proof. The algorithm will iterate until each subset has more than one *representative.* At each iteration, it will check every type T_j one by one. For each $T_j \in T$, it examines each subset S_k eligible for swapping, if $r_j\ inS_k$. Therefore, obviously the complexity of Algorithm 1-rep is $\mathcal{O}(|S|^2|T|)$. □

5.2 Algorithm presentation

Following, the approximation algorithm is presented and explained.

Algorithm 4: Approximation algorithm for solving SU-SID

```
Result: Given A vertex set $V$ partitioned into a set of $T$ types and a
        matrix of subset specifications $M$, it return an approximate
        solution with $S$ and $G = (V, E_{app})$.
1  for each subset $T_j \in T$ do
2      $S^* \leftarrow$ repWise ($M$, $T$)
3  $S \leftarrow$ 1-rep ($S^*$, $M$, $T$)
4  $E_{out} \leftarrow \emptyset$, $E_{reps} \leftarrow \emptyset$
5  for each subset $S_i \in S$ do
6      $head_i \leftarrow$ one representative $r_j \in S_i$
7      for each vertex $u \in S_i$ do
8          $T_x \leftarrow T[u]$
9          if $r_x \in S_i$ then
10             $E_{out} \leftarrow E_{out} \cup (u, r_x)$
11         else if $r_j \notin S_i$ then
12             $E_{out} \leftarrow E_{out} \cup (u, head_i)$
13 $S_{reduced} \leftarrow$ reduceSubsets($S$)
14 $E_{reps} \leftarrow$ SID($S_{reduced}$)
15 $E_{app} \leftarrow E_{out} \cup E_{reps}$
16 return $G = (V, E_{app})$ , $S$
```

Algorithm Explanation

• In line $1-3$, Algorithm repWise is applied for each type $T_j \in T$ resulting in a collection S^* with *rep-wise chosen* types. Then, Algorithm 1-rep is applied on S^* yielding a collection S with the *1-rep property*.

• In lines $5-12$, the set of edges E_{out} is created which includes all edges of the solution of which exactly one endpoint vertex is a *representative*. Since S has the *1-rep property*, each subset $S_i \in S$ contains at least 1 *representative*. In line 6, one of them, denoted $head_i$, is arbitrarily selected. Afterwards, each vertex $u \in S_i$ in the following way. In lines $9-10$, if the *representative* of the type of u, denoted r_x, is part of the subset, then the edge (u, r_x) is added. If $r_x \notin S_i$ vertex u is connected to the *head* of that subset $(u, head_i)$, as seen in lines $11-12$.

• In line 13, a *reduced*, $S_{reduced}$, is created. Each subset in $S_{reduced}$ originates from a subset corresponding in S but for which only the *representatives* are considered and all the other vertices have been eliminated.

• In line 14, the SID problem is solved for $S_{reduced}$ on a vertex set considering only *representatives*. At this point the SID problem can be solved either exactly or by using the *greedy* algorithm presented in textcolorredREF!. In any case a set of edges E_{reps} with edges between the *representatives* is generated.

- Finally, graph $G = (V, E_{out} \cup E_{reps})$ and a collection S is returned.

An example

Suppose we have the following instance.

$$\boxed{\begin{array}{l} M_1 = [1,1,0] \\ M_2 = [0,1,1] \\ M_3 = [2,0,1] \\ M_4 = [1,2,1] \\ M_5 = [1,2,0] \\ M_6 = [3,1,0] \end{array}} \qquad \boxed{\begin{array}{l} V = R \cup G \cup B \\ R = \{r_1, r_2, r_3, r_4, r_5\} \\ G = \{g_1, g_2, g_3, g_4\} \\ B = \{b_1, b_2\} \end{array}}$$

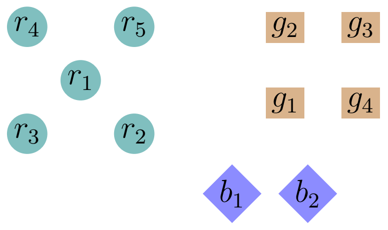


Figure 17: An instance of SU-SID with $|V| = 12, |S| = 6, |T| = 3$.

By line 4, a collection S with the *1-rep property* will be generated, as illustrated below. As *representatives*, r_1, g_1 and b_1, have been arbitrarily selected. And the corresponding *heads* selected in line 6 are underlined below.

$$S = \{S_1, S_2, S_3, S_4, S_5, S_6\}$$

$$S_1 = \{\underline{r_1}, g_4\} \qquad S_2 = \{g_1, \underline{b_1}\} \qquad S_3 = \{\underline{r_1}, r_4, b_2\}$$

$$S_4 = \{r_1, \underline{g_1}, g_2, b_1\} \qquad S_5 = \{r_5, \underline{g_1}, g_3\} \qquad S_6 = \{r_1, r_2, r_3, \underline{g_1}\}$$

In Figure 18, the edge set E_{out} is created according to lines $5-12$ of the Algorithm. On the left side, is the case where r_x is present in the subset, lines $9-11$, and edges between vertices of the same type are added. On the right side, is the case where $r_x \notin S_i$, lines $12-13$, and edges between different types and exactly one endpoint is a *representative* are added.

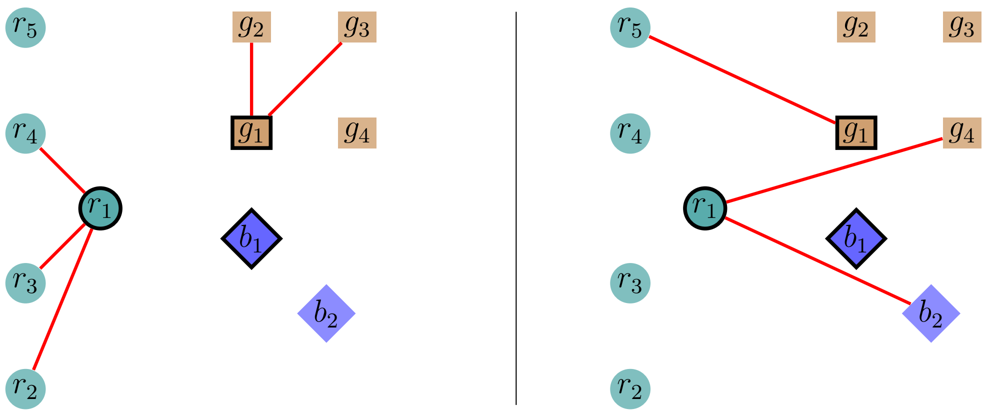


Figure 18: Illustration of the steps of the algorithm need to create E_{out}.

At this point, the *reduceSubsets* algorithm is applied on the collection S, as explained above. The following subsets are obtained:

$$\begin{array}{ccc} S_1 = \{r_1\} & S_2 = \{g_1, b_1\} & S_3 = \{r_1\} \\ S_4 = \{r_1, g_1, b_1\} & S_5 = \{g_1\} & S_6 = \{r_1, g_1\} \end{array}$$

By removing singletons we end up $S_{reduced} = \{(g_1, b_1), (r_1, g_1, b_1), (r_1, g_1)\}$, originating from subsets S_2, S_4, and S_6 respectively.

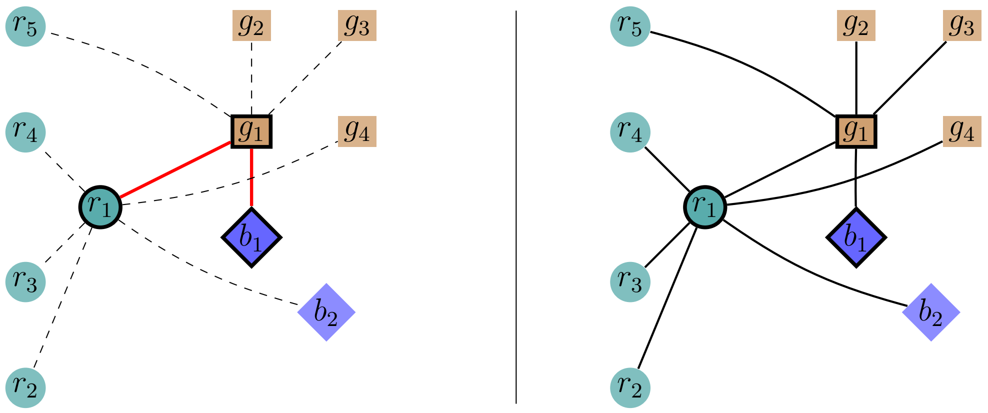


Figure 19: Edge set,E_{reps}, created by solving SID and a final solution.

On the left side of Figure 19 the edge set E_{reps} is illustrated, which is obtained by solving the SID on $S_{reduced}$ and the *representatives*. Finally, on the right side of the figure the edge set of the solution E_{app} which is a union of the two previous edge sets is illustrated. This concludes the illustration of Algorithm 4, for the instance of Figure 17.

5.3 Algorithm analysis

Correctness and Complexity

Proposition 5.1: (Approximation algorithm correctness)
Algorithm 4 for SU-SID *yields always a feasible solution.*
Proof. To prove the correctness *Algorithm* 4, we have to show that the constraints posed in Definition 2.2 are not violated.
i) Each subset S_i satisfies the corresponding vector M_i. This is obviously true by the definition of the *1-rep property.*
iv) Graph G is connected. This is because by the time E_{out} is created, all the vertices are connected to some *representative.* So, at this point there exist $|T|$ connected components. Later on, when SID is solved these $|T|$ components are connected. Otherwise, the solution to SID would not be feasible.
v) Each edge $e \in E$ contributes to at least 1 subset. For the edges $e \in E_{out}$ it is true because they connect each vertex to a *representative.* Thus, they *contribute* to at least one subset. For the E_{reps} edges SID problem is solved. If an edge $e \in E_{reps}$ did not *contribute* to any subsets, such an edge would not be yielded from solving SID.
ii) Each induced subgraph $G[S_i]$ is connected. For each $S_i \in S$, the *chosen* vertices have a direct edge to a *representative chosen* by that subset. Also from *1-rep property* each S_i has *chosen* at least one *representative.* So, it remains to prove that the *representatives chosen* by S_i are connected. This is obvious, as SID is solved between the *representatives.* □

Lemma 5.3 (Approximation algorithm complexity)
Algorithm 4 has a $\mathcal{O}(|S||V| + |S|^2|T| + |S||T|^2 + |T|^4)$ time complexity.
Proof. Algorithm repWise $\forall\, T_j \in T$, from Lemma 5.1 has $\mathcal{O}(|V||S|)$ complexity. *Algorithm 1-rep* has $\mathcal{O}(|S|^2|T|)$ complexity, from Lemma 5.2. For the creation of E_{out}, in lines $5 - 12$ each subset S_i each considered and then for each subset every vertex $u \in S_i$ is considered. So, it requires $\mathcal{O}(|V||S|)$ time complexity. *Reduction* of subsets has also a $\mathcal{O}(|V||S|)$ time complexity. Finally, for solving *SID* on the *representatives*, there exists a $\mathcal{O}(|T|^4 + |S||T|^2)$ greedy algorithm [19]. Summing up, the dominating of these four values will define the time complexity. Therefore, it results in a $\mathcal{O}(|S||V| + |S|^2|T| + |S||T|^2 + |T|^4)$ time complexity. □

Lemma 5.4 (Size of solutions)
A solution from Algorithm 4 is bounded by $|E_{app}| \leq \frac{|T|\times(|T|-1)}{2} + |V| - |T|$
Proof. Every vertex $u \in V$ apart from the *representatives* neighbours with exactly one *representative.* So we have $|V| - |T|$ edges. Between the *rep-*

resentatives there can exist at most $\frac{|T|\times(|T|-1)}{2}$ edges, when the complete graph $K_{|T|}$ is formed. So, in total, an edge set E_{app} is bounded by $|E_{app}| \leq \frac{|T|\times(|T|-1)}{2} + |V| - |T|$. □

Approximation ratio

Proposition 5.2: (Algorithm 4 approximation ratio)
Algorithm 4 is a λ-approximation algorithm for SU-SID*, with* $\lambda = \left(\frac{|T|^2}{2|V|} + 1\right)$
Proof. To find the approximation ratio we have to compare the size of a calculated solution with that of an optimal solution. Any feasible solution is lower bounded by $|V| - 1$, thus, $E_{OPT} \geq |V| - 1$. From Lemma 5.4, a solution is bounded by $|E_{app}| \leq \frac{|T|\times(|T|-1)}{2} + |V| - |T|$. We deduce that in the worst case an optimal solution is $|E_{OPT}| = |V| - 1$ and the algorithm yields a solution with $|E_{app}| = \frac{|T|\times(|T|-1)}{2} + |V| - |T|$.

$$\lambda = \frac{|E_{app}|}{|E_{OPT}|} = \frac{|T| \times (|T| - 1)}{2} \cdot \frac{1}{|V| - 1} = \frac{|T|^2 - 3|T| + 2|V|}{2|V| - 2} = \cdots$$

Regardless of the ratio between $|V|$ and $|T|$ terms $(-3|T|)$ and (-2) are dominated as $|V|$ and $|T|$ grow , so we proceed as follows.

$$\lambda = \frac{|E_{app}|}{|E_{OPT}|} = \mathcal{O}\left(\frac{|T|^2 + 2|V|}{2|V|}\right) = \mathcal{O}\left(\frac{|T|^2}{2|V|} + 1\right) \square$$

We observe that the performance of the algorithm depends on the number of types. If $|T| = \mathcal{O}(1)$ or $|T| = \mathcal{O}(\sqrt{|V|})$, then the approximation ratio is constant, with $\lambda = \mathcal{O}(1)$. So, when we have a *average* or *high uncertainty*, Algorithm 4 is proved to perform *well*. On the other hand, if we have *low uncertainty*, $|T| = \mathcal{O}(V)$, then Algorithm 4 can potentially yield *far* from optimal solutions, having a $\mathcal{O}(|V|)$ approximation ratio.

Proposition 5.3:
Algorithm 4 returns and optimal solution when $|T| = 2$.
Proof. A complete graph on 2 vertices has only 1 edge. Therefore, from Lemma 5.4, $|E_{app}| = |V| - 1$. The solution will always be optimal, being a tree, matching the results of Corollary 3.1. □

Other results

Proposition 5.4: (Reduction of subsets)
Reducing a collection S can improve the size of the solution only if the instance is not representable.
Proof. If the instance is *representable*, it means that the respective *representatives* will be *chosen* by each subset $S_i : m_ij > 0, \forall T_j \in T$. This means that for the collections will hold that $|S_{reduced}| = |S|$. Moreover, no pairwise relation between any types will be reduced. This will end up, in no improvement for the size of the solution, as all constraints between types will be preserved, when solving SID between the *representatives*. $\square$

We deduce from the above, that we expect Algorithm 4 to behave *better* when instances are not *representable*. Moreover, the more types $T_j \in T$ of an instance are not *representable* the better results we can expect.

Proposition 5.5:
An optimal solution to SID *between representatives, does not ensure optimality of solution to* SU-SID.
Proof. We prove this, by giving a counter-example, illustrated below.

$M_1 = [0,1,2]$
$M_2 = [1,2,0]$
$M_3 = [1,0,1]$
$M_4 = [2,1,2]$

$V = T_a \cup T_b \cup T_c$
$T_a = \{a_1, a_2, \}$
$T_b = \{b_1, b_2\}$
$T_c = \{c_1, c_2\}$

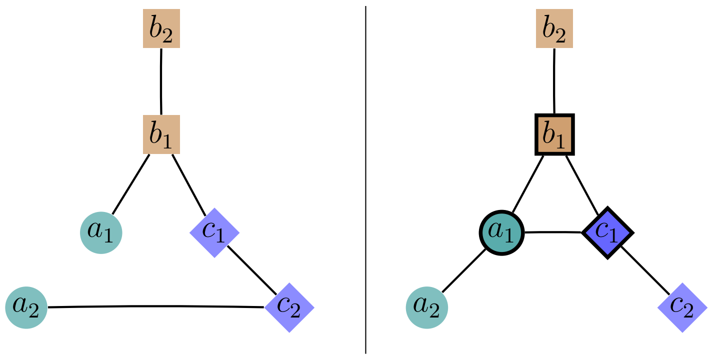


Figure 20: A SU-SID instance, as a counter-example for Proposition 5.5.

On the left side of Figure 20, an optimal solution is illustrated, with size $|E_{opt}| = 5$. On the right side, a solution by Algorithm 4 can be seen, with size $|E_{app}| = 6$. SID between the *representatives* is solved optimally, but despite that fact, $|E_{app}| \geq |E_{opt}|$. Thus, the proposition holds. $\square$

So, although solving SID between the *representatives* is $\mathcal{NP} - hard$, solving it optimally does not guarantee optimality for the SU-SID instance. Nevertheless, it gives the best possible solution this algorithm could yield.

6 LP-based heuristics

So far, we have presented an MILP formulation that solves the problem to optimality. Unfortunately, since we showed that SU-SID is $\mathcal{NP}-hard$, we do not expect MILP to be appropriate for *large* instances. This claim becomes obvious in the next section where we present some experimental results. Moreover, previously we presented an approximation algorithm, the ratio of which depends on $|T|$ and $|V|$. So, there can exist instances with *bad* performance.
In this section, we approach the problem in a heuristic manner. Ideally, those algorithms should provide *good* solutions in polynomial time, without guarantees on the size of the solution. Both of the algorithms presented, are based on an iterative scheme and on the *LP-relaxation* of the MILP model.

6.1 Iterative Rounding of edges

The first algorithm presented is based on an iterative scheme which constructs step by step a feasible solution. Each time a *relaxed* version of the problem is solved, one or more edges are selected for the solution. The algorithm can be described as follows.

Algorithm 5: (iterEdge) Solving SU-SID by iteratively solving a *relaxed* version and choosing edges.

```
Result: Given a SU-SID instance it returns a feasible solution.
1  repeat
2      $S, \boldsymbol{y} \leftarrow$ solve $SUSID_{RELAX}(\boldsymbol{y})$
3      for $y_e \in \boldsymbol{y}$ do
4          if $y_e = 1$ then
5              fix $y_e = 1$
6          else if $y_e = 0$ then
7              fix $y_e = 0$
8      if $\boldsymbol{y}$ has not changed then
9          $e \leftarrow weightedRandomSelection(\boldsymbol{y})$
10         fix $y_e = 1$
11 until $\boldsymbol{y}$ is integral
12 return $S$, $G = (V, E)$
```

Algorithm Explanation
• The algorithm iterates until a integral solution is found. Vector $\boldsymbol{y}$ is the y_e variables $\forall\, e \in E$ as presented in the MILP formulation, which are now *relaxed.* At each iteration a *relaxed* version of the problem, denoted

$SUSID_{RELAX}$, is solved. Moreover, one or more variables $y_e \in \boldsymbol{y}$ are fixed to 1, thus, gradually constructing a feasible solution.
• All variables y_e, which got an integral value, $y_e = 1$, by solving $SUSID_{RELAX}$ are fixed to 1 for the rest of the algorithm. This means that they will be part of the final solution. This can be seen in lines, $4 - 5$.
• All variables y_e, which had not even a fractional solution, $y_e = 0$, when solving $SUSID_{RELAX}$ are fixed to 0. In this way, they will not be considered any more when solving $SUSID_{RELAX}$. This is represented in lines, $6 - 7$.
• Finally, if no y_e has been fixed to 1 during an iteration, then an fractional edge, $y_e \in (0, 1)$, is randomly selected to be fixed. The probability of an edge e to be selected is proportional to its fractional value and all the variables in $\boldsymbol{y}$. This can be observed in lines, $8 - 10$.

Algorithm iterEdge will iterate at most $\frac{|V| \times (|V|-1)}{2}$ times. This is because at each step at least one edge is selected in the solution and a solution can have at most $|E| = \frac{|V| \times (|V|-1)}{2}$ edges. In practice, we observe that the number of iterations is smaller as in many steps a notable amount of edges are selected. We analyse this behaviour in the experimental results section.

The correctness of *Algorithm iterEdge* is straightforward. This is because the algorithm terminates only after it has an integral edge set E. A fractional edge y_e can only be rounded up. So, such a rounding can only increase the size of E and without violating any feasibility constraint.

Notes on the algorithm

For the *linear programming relaxation* of any MILP, all binary variables, are replaced by continuous ones belonging to the $[0, 1]$ interval. This is done in order to turn an $\mathcal{NP} - hard$ problem into a related one which can be solved in polynomial time. For our problem there is a difficulty which we were not able to tackle so far. This is the *relaxation* of connectivity. For our flow-based formulation there does not seem to be a way to *relax* all variables and still maintain connectivity. So, when we refer to $SUSID_{RELAX}$, it is actually a *partially relaxed* version where all but one, s_u^i, binary variables are *relaxed*. More specifically:

Constraints:	$y_e \in \{0, 1\}$, $y_e^i \in \{0, 1\}$ and $z_u^i \in \{0, 1\}$
Become:	$0 \leq y_e \leq 1$, $0 \leq y_e^i \leq 1$ and $0 \leq z_u^i \leq 1$

We must note that $SUSID_{RELAX}$ is therefore not an LP, thus not being polynomial. Nevertheless, it is an MILP which is much less *complex* related to the initial MILP model.

This algorithm is a heuristic, without any guarantees on performance. We must not rule out the possibility that with proper analysis and adaptation we could devise an approximation algorithm. Such seemingly similar methods, of iteratively solving *relaxed* problems, are described in [17] and could be of potential use.
Also another important direction for improvement is to formulate differently our problem, in order to tackle the connectivity issue, when *relaxing* the MILP. In such a way we could prove *Algorithm iterEdge* to be polynomial. So, the algorithm would be able to solve larger instances.

6.2 Iterative Rounding of vertices

The second algorithm presented is a *2-stage* algorithm. The main idea consists of finding, in the first stage, a collection of subsets S, which will later yield a *good* solution. To find the collection S, a *relaxed* version of the MILP is solved iteratively. This is done until the specification matrix M is satisfied. On the second stage, the SUBSET INTERCONNECTION DESIGN is solved on S returning a feasible solution for the initial SU-SID instance.

Algorithm 6: (iterVertex) Solving SU-SID by iteratively solving a *relaxed* version to find S, and then solving SID.

```
   Result: Given a SU-SID instance it returns a feasible solution.
1  repeat
2      $S$, $\boldsymbol{z}$ ← solve $SUSID_{RELAX}(\boldsymbol{z})$
3      for $z_u^i \in \boldsymbol{z}$ do
4          if $z_u^i = 1$ then
5              fix $z_u^i = 1$
6      if $\boldsymbol{z}$ has not changed then
7          $(S_i, u)$ ← weightedRandomSelection($\boldsymbol{z}$)
8          fix $z_u^i = 1$
9  until $\boldsymbol{z}$ is integral
10 $E$ ← SID($S$)
11 return $S$, $G = (V, E)$
```

Algorithm Explanation
• The first stage is explained in lines $1 - 10$ and consists of finding a feasible collection of subsets S. It iterates until each $S_i \in S$ has integral values. Matrix $\boldsymbol{z}$ is the z_u^i variables $\forall\, S_i \in S, \forall\, u \in V$ as presented in the MILP formulation, which are now *relaxed*. As before, in each iteration a *relaxed*

version of the problem, denoted $SUSID_{RELAX}$, is solved.
• If a variable z_u^i, has an integral value, $z_u^i = 1$, by solving $SUSID_{RELAX}$, is then fixed to 1 for the rest of the algorithm. This means that vertex u is *chosen* by subset S_i, gradually satisfying the specifcations matrix M. This is seen in lines, $4 - 5$.
• If no z_u^i has been fixed to 1 during an iteration, then a fractional edge, $y_e \in (0, 1)$, is randomly selected to be fixed. The probability of being *chosen* is proportional to its fractional value and the remaining variables in $\boldsymbol{z}$. This is observed in lines, $6 - 8$
• The first stage of the algorithm yields a collection S by the end of line 9. So, since there is no *uncertainty*, the SID problem can be now solved on S. This can be done in any possible way, either optimally, using an MILP formulation [2], or approximately using a greedy algorithm [4,19].

For the first stage of *Algorithm iterVertex* there will be at most $|V| \cdot |S|$ iterations. This is because at each step at least one vertex will be *chosen* by a subset. In practice thought, as previously, the number of iterations is much smaller. More details can be seen in the experimental analysis.

Algorithm iterEdge always yields a feasible solution. At the end of the first stage there exists an integral composition of subsets which will be feasible. Collection S satisfies M. So, by solving SID we can be sure that a feasible solution for S will be a feasible solution for SUSID, as well.

Notes on the algorithm

The core idea of this algorithm is to turn a highly complex $\mathcal{NP} - hard$ problem, as SU-SID, into another $\mathcal{NP} - hard$ problem, as SID, in order to use the existing results. So, ideally with an extra time complexity we could obtain a collection S where we could apply the existing techniques for SID. Although, until now, the aforementioned problem of *relaxing* the connectivity prevents *Algorithm iterVertex* from being polynomial.
In any case, this *2-stage* concept can be very important for devising improved algorithms. On one direction, it could be possible to examine how *bad* could a solution be when a *bad* collection S was found with some method. So by examining it, one could afterwards solve the problem greedily, adding an extra factor to the existing approximation ratio [4] and thus yielding an approximation algorithm for SU-SID. On the other hand, one other approach would be to find on the first stage a collections S satisfying M with specific structure, as in [14,15,16]. Then, on the second stage, such SID instances could be solved polynomially to optimality.

7 Experimental results

In this section we analyse the results obtained by undertaking a series of experiments on different types of SU-SID instances. The experiments are done with all four proposed methods. The main goal of these experiments, is to have a clear view of the complexity of the problem in practice and to practically evaluate the performance of all algorithms. By doing so, we are in place to see the limitations of each method and to propose the most appropriate for different categories of instances. Also, it is possible to study the size and the structure of the solutions yielded by each algorithm. This gives perspectives for further improvement.

The experiments were undertaken using the following tools and technologies.
→ **Sagemath**[22], open-source mathematics software system, GPL licensed.
→ **Python**, programming language.
→ **CPLEX**, optimization software package *(solver)*, provided by IBM.

We start by presenting some results and explaining them accordingly. Following, we compare all the algorithms together giving suggestions for the appropriateness of each regarding to different types of instances.

7.1 Experimental results

For the instances we have used a *random generator* to create them. More details on the generation of the instances are provided in the Appendix. We have conducted the experiments on 15 different instances. The range on the number of vertices is $|V| \in \{10, 15, 25, 40, 60\}$. And the number of types for each V is roughly $|T| = \mathcal{O}(\sqrt{V})$, $\mathcal{O}(\frac{V}{2})$ and $\mathcal{O}(V)$. So, we have chosen to experiment on *average* to *high uncertainty.* For the number of subsets we have $|S| = |V|$. We must note that $|S|$ highly influences the runtime of all non-polynomial algorithms. For all experiments we have considered a plausible time limit of $t = 1200sec.$
The results presented are partial, showing 8 out of 15 instances. The full results can be seen in Tables 3 and 4 of the Appendix.

Mixed Integer Linear Program

In left part of Table 1, the results of the MILP execution are reported. Value *Opt.Gap.* refers to the optimality gap given by the solver if it did not manage to find an optimal solution, by the end of time limit t. It can be defined as the ratio between the best found integer solution, *sol*, over the best known lower bound, at least $|V|-1$, as: $Opt.Gap. = \frac{(sol)-(lbound)}{(lbound)}\%$.

We observe that the *MILP* can be solved to optimality up to a threshold of almost $15-20$ vertices. A solution with a relatively small optimality gap. can be computed up to a threshold of almost $30-40$ vertices. After that threshold the optimality gap grows *so much* that the solutions provided are not of particular use.

	MILP			**APRX$_{\mathbf{MILP}}$**			**APRX$_{\mathbf{Greedy}}$**	
Instance	$\lvert E\rvert$	*Time*	*Opt.Gap*	$\lvert E\rvert$	*Time*	*Opt.Gap*	$\lvert E\rvert$	*Time*
$\lvert V\rvert = 10, \lvert T\rvert = 5$	**10**	1	0.00%	11	1	0.00%	11	1
$\lvert V\rvert = 15, \lvert T\rvert = 4$	**14**	3	0.00%	16	1	0.00%	16	1
$\lvert V\rvert = 15, \lvert T\rvert = 7$	**15**	2	0.00%	16	2	0.00%	16	1
$\lvert V\rvert = 25, \lvert T\rvert = 5$	**25**	1200	4.00%	28	1	0.00%	28	1
$\lvert V\rvert = 25, \lvert T\rvert = 12$	**30**	1200	20.00%	34	2	0.00%	35	1
$\lvert V\rvert = 40, \lvert T\rvert = 6$	46	1200	15.22%	**41**	1	0.00%	42	1
$\lvert V\rvert = 40, \lvert T\rvert = 18$	131	1200	70.23%	**55**	1200	17.35%	55	1
$\lvert V\rvert = 60, \lvert T\rvert = 25$	218	1200	72.94%	**94**	1200	19.15%	97	1

Table 1: Partial experimental results for *MILP* and approximation algorithm.

Approximation algorithm

Regarding the approximation algorithm we have experimented with two different versions. The difference is in the way SID is solved between the *representatives*. For both the existing methods in [2] were used. The first one, denoted $APRX_{MILP}$, uses the MILP, solving it to optimality. We also observe here the *Opt.Gap.* when the MILP does not finish in time. We must note that this gap refers to the optimal solution between the *representatives* which can be different as shown in Proposition 5.5. The second, denoted $APRX_{Greedy}$, uses a Greedy algorithm to solve SID. The results can be seen on the right part of Table 1.

It is notable that $APRX_{Greedy}$ has always a runtime of 1 sec. If $|T|$ is small and the *MILP* between the *representatives* finds an optimal or near optimal solution, then $APRX_{MILP}$ yields a better solution. That difference between *Greedy* and *MILP* is roughly up to 3 vertices. We expect that for large instances and a large $|T|$ the MILP will no longer be able to find feasible solutions. Otherwise the solutions will have a *bad* ratio, therefore $APRX_{Greedy}$ will for sure provide better results.

Heuristics

Following the results from the two different heuristics are presented. For algorithm $iterVert$ we also have two approaches for the second stage of the algorithm. In $iterVert_{MILP}$, SID is solved to optimality. As previously, we must note that the *Opt.Gap.* can differ from the real one. It refers to the optimality gap for the specific collection that was found in the first stage. Similarly, in $iterVert_{Greedy}$, SID is solved in a greedy manner. The results can be seen in Table 2. When there is the *NO* value, it means that the algorithm did not manage to terminate in time and find a feasible solution.

	iterEdge		**iterVert$_{\mathbf{Greedy}}$**		**iterVert$_{\mathbf{MILP}}$**		
Instance	$\lvert E\rvert$	*Time*	$\lvert E\rvert$	*Time*	$\lvert E\rvert$	*Time*	*Opt.Gap*
$\lvert V\rvert = 10, \lvert T\rvert = 5$	12	2	11	2	11	2(2)	0.00%
$\lvert V\rvert = 15, \lvert T\rvert = 4$	17	6	14	14	15	114(14)	0.00%
$\lvert V\rvert = 15, \lvert T\rvert = 7$	18	5	18	10	18	10(9)	0.00%
$\lvert V\rvert = 25, \lvert T\rvert = 5$	28	28	28	118	29	122(121)	0.00%
$\lvert V\rvert = 25, \lvert T\rvert = 12$	38	34	35	86	36	144(80)	0.00%
$\lvert V\rvert = 40, \lvert T\rvert = 6$	52	441	49	1107	51	1200(1155)	22.67%
$\lvert V\rvert = 40, \lvert T\rvert = 18$	66	280	59	465	58	1200	17.35%
$\lvert V\rvert = 60, \lvert T\rvert = 25$	115	1044	NO	1200	NO	1200	-

Table 2: Partial experimental results for the heuristics *iterEdge* and *iterVert*

The value in parentheses in $iterVert_{MILP}$ refers to the time the execution required for iteratively solving the *relaxed* versions. This is, obviously, the same value for $iterVert_{Greedy}$. Algorithm *iterVert* seems to be yielding slightly better results than *iterEdge* but it seems to be scaling a little less, as $|V|$ increases. Finally, in Table 4 of the Appendix, the number of iterations the *relaxed* version was solved for each instance is denoted by value *#iter*.

7.2 Comparison of algorithms

As expected, solving the problem to optimality using *MILP* has its limitations. SU-SID is highly complex and these limitations are observed even when there is small number of vertices. So, depending on the instance, and the time limit given we expect to get *good* results for up to a threshold of roughly $|V| = 40$.
The approximation algorithm has a very good behaviour in practice. We observe that when *MILP* reaches its limitations it is the best possible approach. Notably, even when *MILP* yields the best solution, the difference in the size observed is up to 4 edges. Another important observation is the fact

that even when there is a big number of types $|T| = \mathcal{O}(|\mathcal{V}|)$, we obtain very good results. So, apart from theory the approximation algorithm behaves *very good* even in practice.
Regarding *iterEdge* algorithm, we observe that it scales up to a threshold of $|V| = 60$ with the given time limit. The results obtained can sometimes be close to the optimal and some times far from it. Nevertheless, at the point when the *MILP* stops yielding good results there is an interval where *iterEdge* behaves better. This is until algorithm *iterEdge* itself stops scaling.
For algorithm *iterVert* we observe that it is yielding slightly better solutions compared to *iterEdge*. It is competitive with *MILP*, when it stops scaling but it has strong limitation itself. We observe that most of the time of the execution is required to find the collection S. Although the *relaxed* version is solved few times compared to $|V|$ it still requires a lot of time.

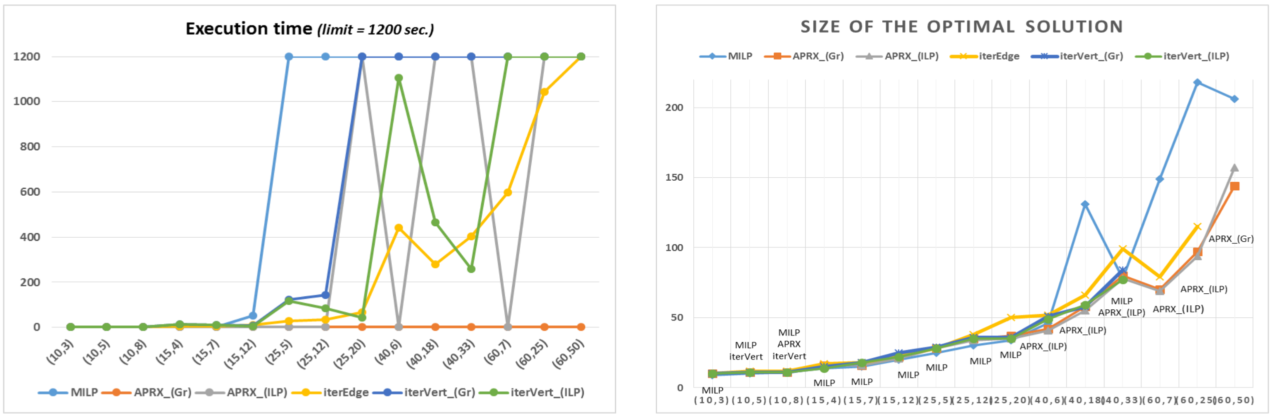


Figure 21: Charts of size of the solution and execution time of experiments.

We summarize this section by highlighting the following observations.
∘ Solving the problem to optimality with *MILP* has limitations which are reached *relatively fast*.
∘ The approximation algorithm behaves very good in practice and almost optimally when $|T|$ is small.
∘ The heuristics proposed, do not behave better than the approximation algorithm in general. Nevertheless, they are competitive to the *MILP* and there is a room for improvement, to make them scale better, each of them in a different direction.

For the *2-stage* concept of *iterVert*, a *good* collection S could be found much faster. Even if that would require completely changing the idea of iteratively solving the *relaxed* version.

For *iterEdge* a different formulation could be used as mentioned previously. This would allow the algorithm to behave much better in practice.

8 Conclusions

8.1 Summary

In this report we defined and studied the Subset Interconnection Design problem with Subset Uncertainty which has not been addressed before. In the first section, we started by presenting the current work and results on SID, some of which are used and others extended in the current work. In the next section, we defined and explained SU-SID and its motivations. We also considered its hardness, showing it is an $\mathcal{NP}-hard$ problem being a generalization of SID. In the third section, we made an in-depth analysis on the problem, acquiring a tight upper bound for optimal solutions. We then presented various other results and gave directions for further work on identifying cycles in the solution. In the following section, we presented a Mixed Integer Linear Programming formulation which solves the problem to optimality. Alongside, several variations and extensions to the model were given. In the fifth section, we proposed a polynomial time algorithm. We analysed its complexity, its approximation ratio, showing it is $\left(\frac{|T|^2}{2|V|}+1\right)$, and some other properties. Next section, engaged in the devise of heuristic of efficiently solving the problem. Both heuristics presented are based on the *LP-relaxation* of the MILP model and on an iterative scheme. After analysing them, we gave guidelines for potential improvement in future work. Finally, the experiments undertaken on different instances were presented. All methods studied in the report were considered in the experiments and their behaviour is analysed, giving suggestions for their use.

More specifically, in the context of the internship, the work undertaken, in close cooperation with my supervisors, and the respective contribution, can be summarised in the following milestones.

◦ Studied the existing framework for SID, results of which were later used.
◦ Formally defined Subset Uncertainty-SID and studied its complexity.
◦ Thoroughly analysed SU-SID, giving among other results, a tight upper bound for the size of the solution.
◦ Modelled the problem with MILP, solving the problem to optimality.
◦ Gave an approximation algorithm for the problem and analysed it.
◦ Devised two different heuristics, based on *LP-relaxation* and SID problem.
◦ Implemented all proposed methods in *Python.*
◦ Experimented on various instances, with all algorithms, studying their behaviour in practice.
◦ Gave various directions for further work on various directions*(right after).*

8.2 Further work

As mentioned, SUBSET INTERCONNECTION DESIGN WITH SUBSET UNCERTAINTY has not been addressed before. Therefore, there many possible directions to follow for further work, some of which have already been proposed within the respective sections. The core idea is to tackle the high complexity of the SU-SID and be able to find *good* solution in a *plausible* time.
The first main direction, is in analysing the problem and the structure of the solutions. It would be very important if some more solid results regarding cycles in the solution were found. So, we would like to find out under which conditions such cycles exist and how we can identify them efficiently
Regarding the MILP formulation there is also room for improvement. It remains open if there exists another formulation, which can be *relaxed* without affecting the connectivity, which seems to be the case now. Also an integration of any other theoretical results about the structure of the solution could provide effective cuts reducing the search space.
On the design of approximation algorithms there are two straightforward directions that have already been mentioned. One is to adapt the iterative scheme with *LP-relaxation* in such a way that, an appropriate analysis can yield an approximation algorithm. On the other hand, finding a collection from the *relaxation* and solving with a *2-stage* scheme could also give guarantees on the size of the solution.
We know that there exist categories of SID instances that can be solved in polynomial time. Constructing collections that satisfy the corresponding specification matrices and fall into one of these categories would be of interest. That would not guarantee optimality in total, but optimality in the SID part. What would then be the difference from an optimal solution for SU-SID is the question that follows.
Closing, two questions come up. Firstly, we know that in the context of Structural Biology there exists some specific structure between different *proteins-vertices*. It would be compelling to see if this could possibly be exploited. Also, we have seen that is already difficult to find one *good*, if not optimal feasible solution. How we could possibly produce a set of such *good* solutions seems intriguing. Finally, it would be of importance to associate SU-SID with other well-studied problems and find useful results and applications.

Bibliography

[1] F. AbuAli, R. Wainwright and D. Schoenenfeld. Solving the subset interconnection design problem using genetic algorithms. *In Proc. of the ACM Symposium on Applied Computing*, 1996.
[2] D. Agarwal, J. Arauzo, C. Caillouet, F. Cazals, D. Coudert and S.Perennes. Connectivity inference in mass spectrometry based structure determination. *European Symposium on Algorithms (ESA)*, 2013.
[3] D. Agarwal, C. Caillouet, D. Coudert and F. Cazals. Unveiling contacts within macro-molecular assemblies by solving minimum weight connectivity inference problems. *Molecular and Cellular Proteomics In*, 2015.
[4] D. Angluin, J. Aspnes and L. Reyzin. Inferring social networks from outbreaks. *Proc. of the 21st International Conference on Algorithmic Learning Theory (ALT)*, 2010.
[5] C. Chen, H. Jacobsen and R. Vitenberg. Algorithms based on divide and conquer for topic-based publish/subscribe overlay design. *Networking, IEEE/ACM Transactions on*, 2014.
[6] J. Chen, C. Komusiewicz, R. Niedermeier, M. Sorge, O. Suchy and M. Weller. Polynomial-time data reduction for the subset interconnection design problem. *SIAM Journal on Discrete Mathematics*, 2015.
[7] G. Chokler, R. Melamed, Y. Tock and R. Vitenberg. Constructing scalable overlay network for pub-sub with many topics. *In Proc. of the ACM Symposium of Distributed Computing (PODC)*, 2007.
[8] D.-Z. Du. An optimization problem on graphs. *Journal Discrete Applied Mathematics (JDAM)*, 1986.
[9] D.-Z. Du and D.F. Kelley. On complexity of subset interconnection designs. *Journal of Global Optimization*, 1995.
[10] D.-Z. Du and Z. Miller. Matroids and subset interconnection design. *SIAM Journal on Discrete Mathematics (SIDMA)*, 1988.
[11] H. Fan and Y.-L. Wu. Interconnection graph problem. *Proc. of International Conference of Foundations of Computer Science*, 2008.
[12] H. Fan, C. Hundt, Y.-L. Wu and J.Ernst. Algorithms and implementation for interconnection graph problem. *International Conference on Combinatorial Optimization and Applications (COCOA)*, 2008.
[13] J. Hosoda, J. Hromkovic, T. Izumi, H. Ono, M. Steinova and K. Wada. On the approximability and hardness of minimum topic connected overlay and its special instances. *Theoretical Computer Science*, 2012.
[14] E. Korach and M. Stern. The clustering matroid and the optimal clustering tree. *Mathematical Programming*, 2003.
[15] E. Korach and M. Stern. The complete optimal stars-clustering-tree problem. *Journal Discrete Applied Mathematics (JDAM)*, 2008.

[16] E. Korach and M. Stern. On a clustering problem in the industry. *(http://www.researchgate.net/publication/238430641_ON_A_CLUSTERING_PROBLEM_IN_THE_INDUSTRY)*, 2008.
[17] L. Chi Lau, R. Ravi and M. Singh. Iterative methods in combinatorial optimization. *Cambridge University Press*, 2011.
[18] T. Noronha, C. Ribeiro and A. Santos. Solving diameter constrained minimum spanning tree problem by constraint programming. *International Transactions in Operational Research*, 2010.
[19] E. Prisner. Two algorithms for the subsets interconnection design problem. *Networks*, 1992.
[20] M. Sharon and C. Robinson. The role of mass spectrometry in structure elucidations of dynamic protein complexes. *Annual Review of Biochemistry 76*, 2007.
[21] T. Taverner, J. Hernandez, M. Sharon, R. Ruotolo, D. Matak-Vinkovic, D. Devos, R. Russel and C. Robinson. Subunit architecture of intact protein complexes from mass spectrometry and homology modelling. *Accounts of Chemical Research 41*, 2008.
[22] The Sage Developers. Sage Mathematics software.
(http://www.sagemath.org).

Appendix

Figures 22 and 23 illustrate the different structure of the solutions obtained for the instance with $|V| = 15, |T| = 7$, by the four different algorithms.

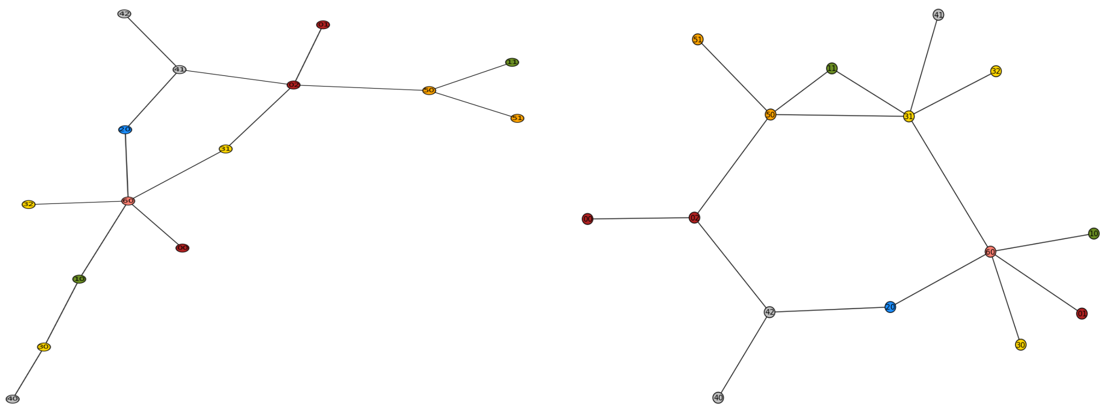


Figure 22: Solution from MILP *(left)* and Approximation algorithm *(right)*.

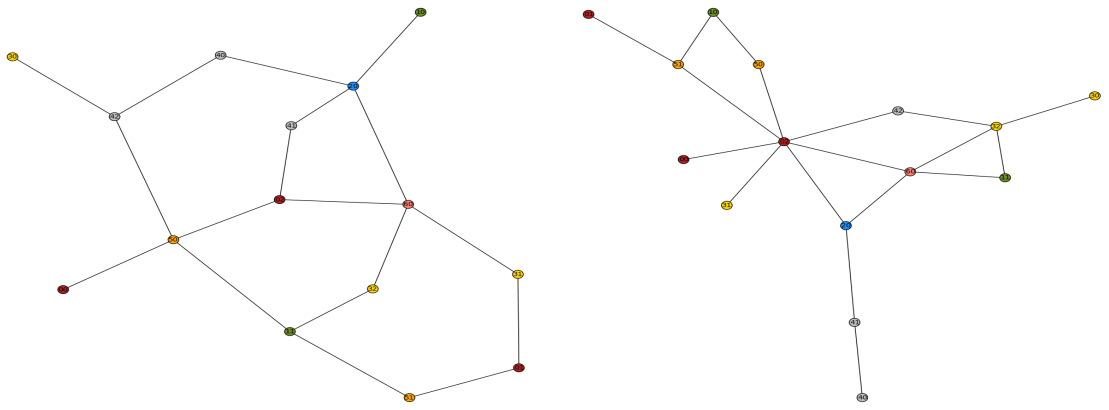


Figure 23: Solution obtained from iterVert *(left)* and iterEdge *(right)*.

	MILP			$\text{APRX}_{\text{Greedy}}$		$\text{APRX}_{\text{MILP}}$		
Instance	$\lvert E\rvert$	Time	Opt.Gap	$\lvert E\rvert$	Time	$\lvert E\rvert$	Time	Opt.Gap
$\lvert V\rvert = 10, \lvert T\rvert = 3$	**9**	1	0.00%	10	1	10	1	0.00%
$\lvert V\rvert = 10, \lvert T\rvert = 5$	**10**	1	0.00%	11	1	11	1	0.00%
$\lvert V\rvert = 10, \lvert T\rvert = 8$	**11**	1	0.00%	**11**	1	**11**	1	0.00%
$\lvert V\rvert = 15, \lvert T\rvert = 4$	**14**	3	0.00%	16	1	16	1	0.00%
$\lvert V\rvert = 15, \lvert T\rvert = 7$	**15**	2	0.00%	16	1	16	1	0.00%
$\lvert V\rvert = 15, \lvert T\rvert = 12$	**20**	52	0.00%	22	1	21	2	0.00%
$\lvert V\rvert = 25, \lvert T\rvert = 5$	**25**	1200	4.00%	28	1	28	1	0.00%
$\lvert V\rvert = 25, \lvert T\rvert = 12$	**30**	1200	20.00%	35	1	34	2	0.00%
$\lvert V\rvert = 25, \lvert T\rvert = 20$	**34**	1200	24.62%	37	1	35	1200	14.28%
$\lvert V\rvert = 40, \lvert T\rvert = 6$	46	1200	15.22%	42	1	**41**	1	0.00%
$\lvert V\rvert = 40, \lvert T\rvert = 18$	131	1200	70.23%	58	1	**55**	1200	17.35%
$\lvert V\rvert = 40, \lvert T\rvert = 33$	**78**	1200	24.38%	80	1	**78**	1200	18.28%
$\lvert V\rvert = 60, \lvert T\rvert = 7$	149	1200	60.40%	70	1	**69**	1	0.00%
$\lvert V\rvert = 60, \lvert T\rvert = 25$	218	1200	72.94%	97	1	**94**	1200	19.15%
$\lvert V\rvert = 60, \lvert T\rvert = 50$	206	1200	60.04%	**144**	1	157	1200	39.83%

Table 3: Experimental results obtained from *MILP* and *Approximation* alogirithms.

	iterEdge			$\text{iterVer}_{\text{Greedy}}$		$\text{iterVert}_{\text{MILP}}$			iterVert
Instance	$\lvert E\rvert$	Time	#iter.	$\lvert E\rvert$	Time	$\lvert E\rvert$	Time	Opt.Gap	#iter.
$\lvert V\rvert = 10, \lvert T\rvert = 3$	10	1	4	10	2	10	2(2)	0.00%	4
$\lvert V\rvert = 10, \lvert T\rvert = 5$	12	2	8	11	2	11	2(1)	0.00%	3
$\lvert V\rvert = 10, \lvert T\rvert = 8$	12	2	10	11	2	**11**	2(1)	0.00%	3
$\lvert V\rvert = 15, \lvert T\rvert = 4$	17	6	11	15	14	14	114(14)	0.00%	7
$\lvert V\rvert = 15, \lvert T\rvert = 7$	18	5	10	18	10	18	10(9)	0.00%	6
$\lvert V\rvert = 15, \lvert T\rvert = 12$	23	2	22	25	2	22	9(4)	0.00%	2
$\lvert V\rvert = 25, \lvert T\rvert = 5$	28	28	22	29	118	28	122(121)	0.00%	12
$\lvert V\rvert = 25, \lvert T\rvert = 12$	38	34	25	36	86	35	144(80)	0.00%	13
$\lvert V\rvert = 25, \lvert T\rvert = 20$	52	66	49	36	45	35	1200(44)	14.87%	7
$\lvert V\rvert = 40, \lvert T\rvert = 6$	52	441	43	51	1107	49	1200(1155)	6.10%	35
$\lvert V\rvert = 40, \lvert T\rvert = 18$	66	280	60	58	465	59	1200(536)	22.67%	19
$\lvert V\rvert = 40, \lvert T\rvert = 33$	99	405	99	84	260	77	1200(258)	16.37%	9
$\lvert V\rvert = 60, \lvert T\rvert = 7$	79	598	56	NO	1200	NO	1200	-	-
$\lvert V\rvert = 60, \lvert T\rvert = 25$	115	1044	110	NO	1200	NO	1200	-	-
$\lvert V\rvert = 60, \lvert T\rvert = 60$	NO	1200	-	NO	1200	NO	1200	-	-

Table 4: Experimental results obtained from *iterEdge* and *iterVert* alogirithms.