Separating codes and traffic monitoring

Thomas Bellitto

Separating Codes and Traffic Monitoring

Thomas Bellitto[a,1]

[a]*LaBRI, University of Bordeaux, France*

Abstract

This paper studies the problem of traffic monitoring which consists of differentiating a set of walks on a directed graph by placing sensors on as few arcs as possible. The problem of characterising a set of individuals by testing as few attributes as possible is already well-known, but traffic monitoring presents new challenges that the previous models of separation fall short from modelling such as taking into account the multiplicity and order of the arcs in a walk. We introduce a new and stronger model of separation based on languages that generalises the traffic monitoring problem. We study three subproblems with practical applications and develop methods to solve them by combining integer linear programming, separating codes and language theory.

1. Introduction

Characterising objects by using as few properties as possible is an important task in diagnosis or identification problems and has been broadly studied under different names such as separating/identifying codes or the test cover problem. The notion of separating codes has many applications in a wide range of domains; in each case, we have to deliver a diagnosis with limited or expensive access to information. Notable examples include visualisation and pattern detection [3] [24], routing [19] or fault detection [21] in telecommunication networks, as well as many areas of bio-informatics, such as analysis of molecular structures [12] or in medical diagnosis, where test covers are the core of diagnostic tables (see [30]) and are therefore important for blood sampling or bacterial identification (see [29] for a survey on different methods). Separating codes have also been studied under the name of *sieves* in the context of logic characterisations of graphs; the size of a minimal separating code determines the complexity of the first-order logic formula required to describe a graph [17].

The problem this paper mainly addresses is called traffic monitoring. Assume that traffic is going through a network modelled as a directed graph (e.g. cars in a town, packets in a telecommunication network, skiers in a ski resort...) and that we are given the opportunity to install sensors on the arcs of the digraph. Each time an object walks in the digraph and goes through an equipped arc it activates a sensor, and we know how many times and in which order each sensor was activated by that object. We are given the set of possible walks the object can take. Our goal is to find where to place the sensors so that we are able to determine exactly which route the object took from the information given by the sensors. This problem has been proven NP-complete in [20], even in the case of acyclic digraphs. Aside from the complexity aspect, few results have been obtained on the problem. We have to take into account information such as the multiplicity and the order of the signals sent by the sensors, which place this problem beyond the expressive power of existing models for separating codes and their resolution methods. For the special case of monitoring skiers, Meurdesoif et al.

[1]This study has been carried out with financial support from the French State, managed by the French National Research Agency (ANR) in the frame of the "Investments for the future" Programme IdEx Bordeaux - CPU (ANR-10-IDEX-03-02).

developed a solution for acyclic digraphs [22]. Their (exponential time) algorithm is based on double-path detection and their approach is very different from ours. In this paper, we adopt a new, more flexible approach based on separating codes that allows us to handle more general problems.

The next section provides all the necessary notions. It gives formal definitions of the separating code and traffic monitoring problems, presents how to solve the standard problem of separating codes by reducing it to integer linear programming and outlines the limitations of this model that make it unsuitable for handling traffic monitoring. We present in Section 3 a new model of separation based on language theory that overcomes these limitations and even generalises the traffic monitoring problem. The next three sections focus on particular cases of traffic monitoring. Section 4 studies the case where the set of walks to separate is finite. Section 5 studies the case where we want to separate every walk starting from a set of given vertices to a set of potential destinations. Such sets of walks can be infinite and would therefore yield infinitely many constraints. We study the underlying language and exhibit some properties that enable us to reformulate the problem as a standard integer linear program with finitely many constraints. Finally, Section 6 solves a more general case where we can give a set of pairs of adjacent arcs that the object may not use consecutively in any walk through the digraph, making this model much more relevant for physical networks such as road networks.

2. Preliminaries

2.1. Separating codes

The separating code problem is, given a set of individuals $\mathcal{I}$ and a set of attributes $\mathcal{A}$, to find the smallest subset of attributes $\mathcal{C} \subseteq \mathcal{A}$ such that each individual is characterised by the attributes of $\mathcal{C}$ it possesses. Here, attributes have no value and are simply properties that each individual may or may not possess.

This problem has been particularly studied in the case of graphs, where a separating code is standardly defined as a subset $\mathcal{C}$ of vertices such that each vertex of the graph is characterised by the intersection of its closed neighbourhood with $\mathcal{C}$.

In many practical applications, some attributes are more expensive to test than others. Let a cost be associated to each attribute. Then, the weighted separation problem is the problem of finding a separating code of minimal cost.

Given a code, let the *signature* of an individual i be the set of attributes of the code it possesses. Thus, for separating codes of graphs, the signature of a vertex v is the intersection of its closed neighbourhood with the code. Hence, separating codes are the sets of attributes such that all the individuals have different signatures.

For example, let us consider the following graph and the code $\mathcal{C} = \{x, y, z\}$:

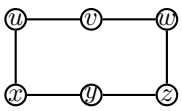

Figure 1: A basic example of a graph.

The signature of x is by definition $N[x] \cap \mathcal{C} = \{x, y\}$. One can see that the signatures of the vertices are pairwise distinct and that the set $\mathcal{C}$ is therefore a separating code.

The notion of separating code of graphs is often combined with another well-known problem of graph theory: domination. A *dominating set* on a graph is a set of vertices such that all the vertices of the graph are in the set or have a neighbour in the set. A set that is both separating and dominating is called an *identifying code*. Looking back at Figure 1, one can see

that the set $\{x, y, z\}$ is separating but not dominating since v has no neighbour in the code while the set $\{v, x, z\}$ is both separating and dominating and is therefore an identifying code. Requiring the additional constraint that each individual should have at least one attribute in a separating code is a well-studied problem. For example in fault detection and security (see [28]), domination ensures that we will notice quickly if a problem occurs and separation allows us to determine what the problem is.

While separation and domination seem unrelated at first sight, one can actually easily reduce identification to separation. Indeed, notice that separation consists of making the signatures of the individuals pairwise distinct while identification requires that the signatures are both pairwise distinct and non-empty. Therefore, all we have to do is add to our set of individuals an artificial individual that has no attribute and whose signature will thus necessarily be empty, thereby forcing all the other individuals to have non-empty signatures. Hence, the identification problem is resolvable if and only if no two individuals possess exactly the same attributes and each individual possesses at least one attribute.

The problem of separating code appeared first under the name of test cover (see [23] or [4] for important examples). Separating codes for graphs were introduced in [16] together with identifying codes and have been widely studied for many subclasses of graphs. One can also find variants of identification for hypergraphs, which consists of characterising the vertices by the hyperedges they belong to, or for bipartite graphs, which consists of separating the vertices of V_1 using only vertices of V_2 where (V_1, V_2) is a bipartition of the graph (see [5] [6]). These two problems are actually equivalent to the general problem given by a set of individuals and a set of attributes. The problem has also been studied on directed irreflexive graphs ([25] [26] [13]), a model whose expressiveness lies between the problem for graphs and the general problem with individuals and attributes. To the best of our knowledge, the first proof of the NP-completeness of the general test cover problem can be found in [10]. The problem of finding the smallest identifying code on a graph was shown to be NP-complete in [7]. The complexity of this problem on specific classes of graphs has also been broadly studied.

We now give a reformulation of the problem as an integer linear program (ILP) on which the methods developed in the next sections are based.

Let $\mathcal{I}$ be a set of individuals and $\mathcal{A}$ be a set of attributes. Let $(i, i') \in \mathcal{I}^2$ be such that $i \neq i'$. The separating set of i and i', denoted by $\mathrm{Sep}(i, i')$, is the symmetric difference of their attributes *i.e.* the set of attributes that exactly one of them possesses. Therefore, i and i' will have distinct signatures according to a set of attributes $\mathcal{C}$ if and only if $\mathcal{C}$ possesses at least one attribute of their separating set.

For each attribute $a \in \mathcal{A}$, let x_a be a binary variable that indicates whether $a \in \mathcal{C}$. We obtain the following system:

$$\begin{cases} \forall (i, i') \in \mathcal{I}^2 \text{ with } i \neq i', \displaystyle\sum_{a \in \mathrm{Sep}(i,i')} x_a \geqslant 1 \\ \text{minimise } \displaystyle\sum_{a \in \mathcal{A}} x_a \end{cases}$$

Polyhedra associated with identifying code problems have been studied thoroughly in [2].

This ILP has a solution if and only if all the separating sets are non-empty *i.e.* if and only if no two individuals possess exactly the same attributes.

Notice that we obtain an ILP formulation of the weighted separating code problem by taking into account the costs in the objective function. For the sake of simplicity, we will stay with the unweigthed separating code problem in what follows, but the method presented can easily be generalised to the weighted case.

2.2. The traffic monitoring problem

We model a network with a directed graph $G = (V, A)$ and have the option of installing sensors on the arcs of the graph. We want to be able to reconstruct the route of objects walking in the graph.

When an object walks in the graph, it activates a sensor each time it goes through an equipped arc. By moving in the graph, the object activates the sensors a certain number of times in a certain order. The ordered sequence of activated sensors is what we call the *signature* of the walk of the object.

In this problem, we are given the set $\mathcal{R}$ of potential routes the object can take and we are looking for the smallest possible set of arcs such that the signatures of all the routes of $\mathcal{R}$ are pairwise distinct. Is this is the case, the information given by the sensors is sufficient to determine exactly which route the object picked. Note that we make no assumption on the speed of the object; the time between the activation of two sensors cannot be used to determine what the object did in the meantime. However, we know the order in which the sensors were activated.

As before, variants where some arcs are more expensive to monitor than others or where we also want the walks of $\mathcal{R}$ to have non-empty signatures (so that we know if an object is walking in the network) are also of great practical interest.

Again, we have to distinguish a set of individuals (walks in a digraph) by testing as few attributes (the arcs the walks use) as possible. However, while this problem looks close to the the ones we have mentioned so far, it presents three major difficulties that we have not encountered yet which place it beyond the expressive power of the models described previously.

Let us illustrate these difficulties with an example:

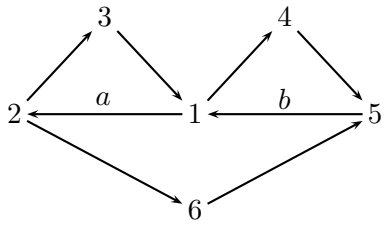

Figure 2: An example of network G. The arcs (1,2) and (5,1) are equipped with sensors called respectively a and b.

- The set of activated sensors is not sufficient to identify a walk; on the network of Figure 2, the walks $(1, 2, 3, 1)$ and $(1, 2, 3, 1, 2, 3, 1)$ are different but the sets of arcs they use are the same. Since they do not use these arcs the same number of times, we still can distinguish them - their signatures according to the sensor set $\{a, b\}$ are respectively a and aa - but this forces us to take into account the multiplicity of the attributes of our individuals, which cannot be done with the previous model.

- The number of times each sensor is activated is not sufficient to identify a walk; in the network of Figure 2, the walks $(1, 2, 3, 1, 4, 5, 1)$ and $(1, 4, 5, 1, 2, 3, 1)$ not only use the same arcs but they also use them the same number of times. They can still be separated; their signatures are indeed respectively ab and ba, but this requires considering the order of the attributes. This also illustrates the limit of the resolution method we showed in the previous section. Indeed, we said that a code separates two individuals if and only if it contains an element that does so, but one can see here that the sensor set $\{a, b\}$ separates the walks $(1, 2, 3, 1, 4, 5, 1)$ and $(1, 4, 5, 1, 2, 3, 1)$ while neither $\{a\}$ nor $\{b\}$ can.

- The set $\mathcal{R}$ of potential walks can be infinite; if the graph contains a cycle, the number of walks in it is infinite and $\mathcal{R}$ can be any subset of it. Therefore, even checking in finite time whether a given set of sensors separates $\mathcal{R}$ is non-trivial since it requires ensuring that all the walks of $\mathcal{R}$ have different signatures. A wrong intuition is that the problem can be reduced to separation on elementary paths since non-elementary walks are concatenations of elementary paths (which would be helpful since there are only a finite number of them), but this does not work. For example, assume that the set $\mathcal{R}$ we want to separate is the set of cycles starting from and leading to the vertex 1. Our set of sensors $\{a, b\}$ is not suitable since the cycles $(1, 2, 3, 1, 4, 5, 1)$ and $(1, 2, 6, 5, 1)$ both have the signature ab although all the elementary cycles ($(1, 2, 3, 1)$, $(1, 4, 5, 1)$ and $(1, 2, 6, 5, 1)$) have different signatures (respectively, a, b and ab).

3. A new model of separation: separation on a language

3.1. The problem

We recall that an *alphabet* is a non-empty finite set whose elements are called *letters*, a *word* is a finite sequence of letters and a *language* is a set of words on a given alphabet. The reader may refer to [14] for basic notions of language and automata theory.

Let A be an alphabet and let $\mathcal{C}$ be a subalphabet of A. Let the *projection on* $\mathcal{C}$, denoted $p_{\mathcal{C}}$, be the function that maps each word $u \in A^*$ to its longest subword which uses only letters of $\mathcal{C}$. For example, $p_{\{a,b\}}(abacacb) = abaab$.

Given a language L on an alphabet A, the problem of separation on a language consists of finding the smallest subalphabet $\mathcal{C} \subseteq A$ such that all of the words of L have different projections on $\mathcal{C}$.

For example, let $L = \{aabcc, acabc, baacb, cbaac\}$. One can immediately notice that $aabcc$ and $acabc$ use the same letters the same number of times, so one cannot separate them with an alphabet of one letter. Plus, the projection of both these words on the subalphabet $\{a, b\}$ is aab so this subalphabet also does not separate them. The projections of $acabc$ and $cbaac$ on $\{b, c\}$ are also the same (cbc). However, the projections of the four words of L on the subalphabet $\{a, c\}$ are respectively $aacc, acac, aac$ and $caac$ and are pairwise distinct. Hence, $\{a, c\}$ is a solution of the problem and is the only optimal solution.

The problem of separation on a language is a generalisation of the problem of separating code. Indeed, let $\mathcal{I}$ be a set of individuals, let $\mathcal{A}$ be a set of attributes and let $\leqslant$ be a total order on $\mathcal{A}$. Let us associate with each individual i a word on $\mathcal{A}$ composed of all the attributes i possesses, exactly once, in increasing order according to $\leqslant$. The subalphabet of $\mathcal{A}$ that separates the language of the words associated to the individuals of $\mathcal{I}$ are exactly the separating codes. Therefore, separation on a language is NP-complete in both the size of the language and the alphabet it is defined on.

3.2. Relation to traffic monitoring

Let us consider the set A of arcs of a digraph as an alphabet. Since a walk on a digraph is a sequence of arcs, it is a word and the set $\mathcal{R}$ of possible routes is a language on A. Given a set of sensors, the signature of a route is its projection on the subalphabet composed of the arcs that have a sensor. Hence, traffic monitoring is a particular case of separating code on a language (however, the problems are not equivalent since not all languages can be written as a set of routes on the alphabet of the arcs of a digraph).

This reduction allows us to use tools arising from separating codes and language theory to address the traffic monitoring problem. However, the problem remains NP-complete in the size of the language that we want to separate. Moreover, we saw that in some instances, this language can be infinite. Hence, we intend not to solve the problem in general, but to address some classes of sets of routes which are of practical interest.

4. Separation of a finite language

The easiest place to start is the case where the language we want to separate is finite. This happens in particular when we want to solve traffic monitoring on an acyclic graph or when we can bound the length of the walks on the network. This problem already covers a wide range of applications.

The ILP we designed in the first section was based on the central notion of separating sets. The fundamental property of the separating set of two individuals i and i' is that a set of attributes separates i and i' if and only if it contains an attribute of their separating set. However, we saw that this property no longer holds for traffic monitoring and separation on a language (see Subsection 2.2). What still holds however is that if a set separates two words w and w', so do its supersets. Hence, we can still look for the minimal sets of attributes that separate two words w and w' and we know that a set of attributes separates w and w' if and only if it contains one of those minimal sets. We thus define the *separating set* $\mathrm{Sep}(w, w')$ of two words w and w' as follows: given two words w and w' on an alphabet A, a subalphabet $\mathcal{C}$ of A belongs to the separating set $\mathrm{Sep}(w, w')$ of w and w' if and only if $\mathcal{C}$ separates w and w' and none of its strict subsets does. The separating set of two words is therefore a set of sets of letters.

It follows from the definition that a subalphabet $\mathcal{C} \subseteq A$ separates two words w and w' if and only if there exists $\mathcal{C}' \subseteq \mathcal{C}$ such that $\mathcal{C}' \in \mathrm{Sep}(w, w')$.

We now exhibit some properties of the structure of separating sets that are useful in computing them efficiently (Theorem 2).

Lemma 1. *A word $u \in A^*$ is characterised by its projections on the subalphabets of A of cardinality 2.*

Proof. The letter a is the first letter of a word u if and only if it is the first letter of all projections of u on the subalphabets of cardinality 2 containing a. One can then remove the a in the first position of the projection of u on the subalphabets containing a and find the second letter of u. This can be iterated until u is entirely determined. □

For example, let $A = \{a, b, c\}$ and $u \in A^*$ such that $p_{\{a,b\}}(u) = abba$, $p_{\{a,c\}}(u) = aca$, $p_{\{b,c\}}(u) = bbc$. Since a is the first letter of the projection of u on $\{a, b\}$ and $\{a, c\}$, we know that a is the first letter of u. Thus, there exists v such that $u = av$. From the projection of u, we deduce that $p_{\{a,b\}}(v) = bba$, $p_{\{a,c\}}(v) = ca$, $p_{\{b,c\}}(v) = bbc$ and the first letter of v is therefore b. Thus, there exists w such that $u = abw$ and by iterating the process, we determine that $u = abbca$.

Theorem 2. *The separating set of two words contains only sets of cardinality at most 2.*

Proof. Let u and v be two words on an alphabet A and let $\mathcal{C}$ be a subalphabet of A of cardinality at least 3 such that no strict subalphabet of $\mathcal{C}$ separates u and v. Hence, for every subalphabet $\mathcal{C}'$ of $\mathcal{C}$ of cardinality 2, $p_{\mathcal{C}'}(u) = p_{\mathcal{C}'}(v)$. Since $\mathcal{C}'$ is a subalphabet of $\mathcal{C}$, we know that $p_{\mathcal{C}'}(u) = p_{\mathcal{C}'}(p_{\mathcal{C}}(u))$ and $p_{\mathcal{C}'}(v) = p_{\mathcal{C}'}(p_{\mathcal{C}}(v))$. Hence, $p_{\mathcal{C}}(u)$ and $p_{\mathcal{C}}(v)$ are two words on $\mathcal{C}$ whose projections on every subalphabet $\mathcal{C}'$ of $\mathcal{C}$ are identical and therefore, by Lemma 1, $p_{\mathcal{C}}(u) = p_{\mathcal{C}}(v)$ which means that $\mathcal{C}$ does not separate u and v. □

Let u and v be two words of A^*. We can build their separating set as follows:

- If a letter a does not appear the same number of times in u and in v, then a alone suffices to separate them. We thus add $\{a\}$ to $\mathrm{Sep}(u, v)$ and while the pairs containing a all separate u and v, they do not belong to the separating set.

- If a letter a appears neither in u nor in v, containing it would be of no help to separate u and v. Thus, the separating set contains no pair containing a. What is left to investigate are pairs $\{a, b\}$ composed of two letters appearing the same number of times in u and v. Let k be the number of occurrences of a in u and v ($k \neq 0$). Thus, there exists $u_0, \cdots, u_k, v_0, \cdots, v_k$ such that $u = u_0 a u_1 a \cdots a u_k$ and $v = v_0 a v_1 a \cdots a v_k$. The pair $\{a, b\}$ belongs to the separating set if and only if there exists $i \in [\![0, k]\!]$ such that u_i and v_i do not contain the same number of occurrences of the letter b. These decompositions of u and v can then be reused when investigating other pairs containing a.

We can take advantage of Theorem 2 to separate a finite language as follows: let $L \subseteq A^*$ be the language we want to separate. For each letter $a \in A$, let $x_{\{a\}}$ be a binary variable that indicates whether $a \in \mathcal{C}$ where $\mathcal{C}$ is the optimal separating subalphabet we want to build. For each pair of letters $\{a, b\}$, let $x_{\{a,b\}}$ be a binary variable that indicates whether both a and b belong to $\mathcal{C}$ (thus, $x_{\{a,b\}} = \min(x_{\{a\}}, x_{\{b\}})$). We obtain the following ILP:

$$\begin{cases} \forall a \neq b \in A, 2x_{\{a,b\}} \leqslant x_{\{a\}} + x_{\{b\}} \quad \text{(ensures that } x_{\{a,b\}} \leqslant \min(x_{\{a\}}, x_{\{b\}})) \\ \forall a \neq b \in A, x_{\{a,b\}} + 1 \geqslant x_{\{a\}} + x_{\{b\}} \quad \text{(ensures that } x_{\{a,b\}} \geqslant \min(x_{\{a\}}, x_{\{b\}})) \\ \forall v \neq v' \in L, \sum_{S \in \mathrm{Sep}(v,v')} x_S \geqslant 1 \quad \text{(where } S \text{ can denote a singleton or a pair)} \\ \text{minimise} \sum_{a \in A} x_{\{a\}} \end{cases}$$

The solution is thus $\mathcal{C} = \{a \in \mathcal{A} \mid x_{\{a\}} = 1\}$. The inequalities of the second line are not necessary to ensure the validity and the optimality of the solution but they ensure that the values of the pair-variables are determined exactly by the values of the singleton-variables. Hence the number of degrees of freedom of the problem is linear in the cardinality of A and not quadratic.

5. Separation of walks from a given set of starting points to a given set of destinations

In the problem that we call *total separation*, we are given a set of potential starting points V_I and a set of potential destinations V_F. The set $\mathcal{R}$ of routes we want to separate is the set of all the routes leading from a vertex of V_I to a vertex of V_F. If the graph contains a cycle, there will be infinitely many such routes.

Of course, since the number of routes to separate can be infinite, it is not feasible to compute the separating set of each pair of routes. However, notice that the separating set of two routes is included in the powerset of the set A of arcs of the graph and can therefore only take a finite number of values. Thus, while the linear program presented in the previous section would have infinitely many constraints, only a finite number of them would be distinct. If we can determine which values of $\mathrm{Sep}(v, v')$ are actually reached, we would therefore be able to describe the same polytope with only a finite number of constraints.

5.1. Study of the reachable languages

We call a language L on an alphabet A *reachable* if and only if there exists a directed graph $G = (V, A)$ (A is both the alphabet of L and the set of arcs of the graph), a set of vertices $V_I \subseteq V$ and a set of vertices $V_F \subseteq V$ such that L is the set of walks on G leading from a vertex of V_I to a vertex of V_F. Let $\mathrm{Reach}(A)$ denote the set of reachable languages on an alphabet A.

Lemma 3. *If $L \subseteq A^*$ is reachable, then:*

$$\forall u, u', v, v' \in A^*, \forall a \in A, \begin{cases} uav \in L \\ u'av' \in L \end{cases} \Rightarrow \begin{cases} uav' \in L \\ u'av \in L \end{cases}$$

Proof. The idea of the proof is that since all the walks on the network are possible, the choices the walker has at a given time only depend on the current vertex and cannot be restricted depending on the route previously used.

More formally, let $G = (V, A)$ be a directed graph and let V_I and $V_F \subseteq V$ be such that L is the set of walks leading from V_I to V_F. Let o_a and t_a be the origin and target of the arc a. If uav and $u'av'$ are in a reachable language L, this means that:

- u describes a walk that leads from a vertex $i \in V_I$ to o_a
- u' describes a walk that leads from a vertex $i' \in V_I$ to o_a
- v describes a walk that leads from t_a to a vertex $f \in V_f$
- v' describes a walk that leads from t_a to a vertex $f' \in V_f$

Hence, uav' and $u'av$ also describe valid walks leading from a vertex of V_I to a vertex of V_F and therefore belong to L too. □

Let $\mathrm{Rat}(A)$ denote the set of rational languages on an alphabet A. We have:

Proposition 4. $\mathrm{Reach}(A) \subsetneq \mathrm{Rat}(A)$

Proof.

• $\mathrm{Reach}(A) \subseteq \mathrm{Rat}(A)$. Given an instance of total separation, one can easily construct an automata recognising the language of all possible routes between V_I and V_F on a directed graph $G = (V, A)$; A is its alphabet, V is its set of states, $T = \{(u, (u, v), v) \mid (u, v) \in A\}$ is its set of transitions, V_I is its set of initial states and V_F is its set of final states. By the theorem of Kleene [18], this proves that reachable languages are rational.

• $\mathrm{Reach}(A) \neq \mathrm{Rat}(A)$. Note that the previous construction provides very specific automata: each letter of the alphabet labels at most one transition. There are rational languages that cannot be recognised by such automata. For example, let L be reachable and such that $ababa \in L$. Then, $(ab)a(ba) \in L$ and $\varepsilon a(baba) \in L$. Hence, according to Lemma 3, $(ab)a(baba)$ and $a(ba)$ both belong to L too. Hence, the language $\{ababa\}$ although rational, is not reachable. □

5.2. Restrictions of a rational language

We recall that rational languages are exactly those that can be described by regular expressions. We define a restriction of a rational language L and we denote by $\overline{L}$ the language built from a regular expression of L as follows.

- $\overline{\varnothing} = \varnothing$
- $\forall a \in A^*, \overline{\{a\}} = \{a\}$
- for all rational languages L_1 and L_2, $\overline{L_1 + L_2} = \overline{L_1} + \overline{L_2}$
- for all rational languages L_1 and L_2, $\overline{L_1 L_2} = \overline{L_1}\ \overline{L_2}$
- for all rational language L, $\overline{L^*} = \varepsilon + \overline{L} + \overline{L}^2$.

This notion will be useful for the reduction theorems (Theorem 6 and Theorem 11).

Notice that the restriction of a language L is not unique. Indeed, two regular expressions can denote the same language but their associated restrictions can differ. For example, $L^{**} = L^*$ but unless $L = \varnothing$, $\overline{L^{**}} = \sum_{i=0}^{4} \overline{L^i} \neq \sum_{i=0}^{2} \overline{L^i} = \overline{L^*}$.

Proposition 5. *Every restriction $\overline{L}$ of a rational language L is finite.*

Proof. The proof by induction is immediate. Indeed, restricted languages are empty, singletons or built from other restricted languages using only finite unions or concatenations, which are operations that preserve the finiteness of the language. □

5.3. Reduction theorem and resolution

Theorem 6 (Reduction Theorem)**.** *For all reachable languages L on an alphabet A, for all restrictions $\overline{L}$ of L, if $A' \subseteq A$ separates $\overline{L}$, then it separates L.*

Proof. This theorem will be proved in a more general case in the next section (see Theorem 11). □

Note that the converse is obviously true since $\overline{L} \subseteq L$. Thus, the languages that separate $\overline{L}$ are exactly those that separate L.

Hence, given a directed graph, a set V_I of potential starting points and a set V_F of potential destinations, we proceed as follows to solve the traffic monitoring problem:

- we know that the language of possible routes leading from a vertex of V_I to a vertex of V_F is rational and therefore admits a regular expression. The graph directly provides an automata which recognises it and we can use Arden's lemma [1] to find an expression of the associated reachable language that we want to separate.
- we use the regular expression of the language to determine a restriction. We know by Proposition 5 that this language is finite.
- we know by the reduction theorem that the solutions on the restricted language are exactly the solutions on the initial language. All that is left to do is use the method described in the previous section to solve the problem on the restricted language that is finite.

6. Separation of routes with forbidden transitions

6.1. Motivation and definition of the problem

Let us consider the following graph that depicts a very simple road:

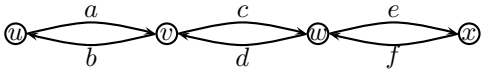

Figure 3: An instance of traffic monitoring.

In this case, a driver who wants to go from vertex u to vertex x will simply use the path ace. Still, the model that we presented in the previous section requires us to distinguish all the routes of $(a(c(ef)^*d)^*b)^*a(c(ef)^*d)^*c(ef)^*e$. Taking into account such paradoxical behaviours is not only superfluous but it also leads to prohibitive computation times and tremendously increases the cost of the solutions. Here, at least three sensors are required to distinguish all the roads leading from u to x while only one of those roads makes sense in practice.

One could be tempted to get around this problem by contracting the arcs a, c and e and the arcs b, d and f in a pre-processing step, which would result in a bi-directed path of length 1 and bring the language to separate down to $(ab)^*a$. However, this is not possible on a more complex road network: indeed, a road network can for example feature crossroads on v and w and transversal roads could lead to those vertices or leave from them. While this does not change the fact that a driver who wants to go from u to x will always pick the route ace, it can make the contraction of the vertices v and w impossible and force us to consider many absurd walks.

The approach we choose here is to set up a new model where certain pair of edges, although adjacent, cannot be taken consecutively. A pair of edge $((u, v), (v, w))$ where u, v and w are vertices is what we call a *transition*. We define a *forbidden-transition graph* (or FTG) as a triplet (V, A, F) where V and A are the sets of vertices and arcs respectively and where F is a set of *forbidden transitions*. A *permitted walk* on a FTG is a walk that does not use consecutively two arcs a and b with $(a, b) \in F$. To avoid any ambiguity, in this section, we will refer to graphs (*i.e.* pairs (V, A) with no set of forbidden transitions) as "usual graphs".

For example, on the road network of Figure 3, we can assume that drivers will not make half-turn in the middle of the road and forbid the transitions ab, cd and ef. Thereby, we reduce the set of roads leading from u to x down to $\{ace\}$. This also enables us to model situations where certain turns are prohibited, which is very common on road networks. By choosing wisely the forbidden transitions, we only discard routes that would be prohibited or absurd in practice and we can significantly reduce the computation time and the cost of the optimal solutions on large instances.

Graphs with forbidden transitions or equivalent models have already appeared in the literature, see [9] for one of the first examples. The complexity of finding paths or cycles with various properties has also received attention (see [27], [15] or [8] for notable examples) as well as several closely related notions such as properly coloured paths (see [11] for a survey).

Just like in the previous section, an instance of total separation on a FTG is given by a FTG (V, A, F), a set V_I of potential starting points and a set V_F of potential destinations and the set of routes $\mathcal{R}$ we want to separate is the set of all the permitted routes leading from a vertex of V_I to a vertex of V_F.

6.2. Study of the FTG-reachable languages

As in Subsection 5.1, we call a language L a *FTG-reachable language* if and only if there exists an instance of total separation on a FTG where L depicts the set of routes to separate. We denote by $\mathrm{FTGR}(A)$ the set of FTG-reachable languages on an alphabet A.

Proposition 7. $\mathrm{Reach}(A) \subsetneq \mathrm{FTGR}(A)$.

Proof.

• $\mathrm{Reach}(A) \subseteq \mathrm{FTGR}(A)$. Since usual graphs are particular cases of FTGs (with $F = \varnothing$), the languages that are reachable with graphs are clearly reachable with FTGs.

• $\mathrm{Reach}(A) \neq \mathrm{FTGR}(A)$. Let L be a reachable language on $A = \{a, b, c, d\}$ such that $ac, ad, bc \in L$. Since both ac and ad are in L, we know that c and d denote arcs starting from the same vertex v and leading to a vertex of V_F. Furthermore, since both ac and bc belong to the language, we know that a and b both denote arcs leading to v and starting from a vertex of V_I. Hence, $bd \in L$ too for every reachable language that contains ac, ad and bc.

Let us now consider the instance of traffic monitoring on the FTG obtained by forbidding the transition (b, d) on the graph given in Figure 4, with $V_I = \{u, w\}$ and $V_F = \{w, x\}$. One can see that the associated FTG-reachable language contains ac, ad and bc since they describe permitted walks leading from a vertex of V_I to a vertex of V_F but not bd since it is forbidden. Hence, this FTG-reachable language is not reachable. □

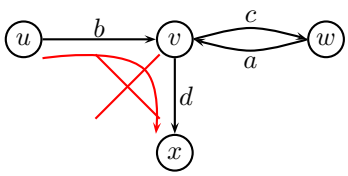

Figure 4: An example of FTG.

It is important to note that Lemma 3 also holds for FTG-reachable languages:

Lemma 8. *For all FTG-reachable languages $L \subseteq A^*$:*

$$\forall u, u', v, v' \in A^*, \forall a \in A, \begin{cases} uav \in L \\ u'av' \in L \end{cases} \Rightarrow \begin{cases} uav' \in L \\ u'av \in L \end{cases}$$

Proof. Let $G = (V, A, F), V_I, V_F$ be an instance of total separation on a FTG such that the set of routes to separate is L. Let us call L' the set of words that denote all the walks leading from a vertex of V_I to a vertex of V_F on the underlying usual graph $G = (V, A)$ with no forbidden transitions. Hence, L' contains all the words of L plus eventually some words that contain transitions of F. Since L' is reachable by construction and $\{uav, u'av'\} \in L \subseteq L'$, we know by Lemma 3 that uav' and $u'av$ belong to L'. Therefore, the only way for uav' or $u'av$ not to belong to L' is to contain two consecutive letters that denote a forbidden transition. However, every sequence of two letters in uav' and $u'av$ also appears in uav or $u'av'$ which both belong to L. Thus, uav' and $u'av$ are in L, which concludes the proof of the lemma. □

Proposition 9. $\mathrm{FTGR}(A) \subsetneq \mathrm{Rat}(A)$.

Proof.

• $\mathrm{FTGR}(A) \subseteq \mathrm{Rat}(A)$. Let L be a FTG-reachable language and $G = (V, A, F), V_I, V_F$ be an instance of total separation on a FTG such that $\mathcal{R} = L$. To prove that L is rational, we create from G an automata that recognises L. To do so we create copies of each vertex of the graph for each possible incidence. For example, if an arc (u, v) leads to a vertex v, instead of just having a state v in our automata, we create a state uv that is final if and only if v is final and from which we can reach any out-neighbour w of v unless $((u, v), (v, w)) \in F$. More formally, here is a construction of an automata that recognises L:

- its alphabet is the set A of arcs of the graph
- its set of states is $V_I \cup \{uv \mid (u, v) \in A\}$
- for all $v \in V_I$, for all out-neighbours w of v, we create a transition from v to vw labelled by the arc (v, w). For all states uv, for all out-neighbours w of v, we create a transition from uv to vw labelled by the arc (v, w) if and only if $((u, v), (v, w)) \notin F$
- the set of initial states is still V_I
- the final states are those whose name ends with a vertex $v \in V_F$.

• $\mathrm{FTGR}(A) \neq \mathrm{Rat}(A)$. The counter-example to the other inclusion is the same as in the proof of Proposition 4: the language *ababa* is rational but not FTG-reachable since it does not satisfy Lemma 8. □

For example, with the graph in Figure 5, $F = \{(a, b), (c, d), (d, c), (f, e)\}, V_I = \{v\}$ and $V_F = \{w\}$, the associated FTG-reachable language is recognised by the automata presented in Figure 6 whose initial and final states are respectively $\{v\}$ and $\{vw, xw\}$. Initial and final states are denoted by incoming and outgoing dashed arrows respectively.

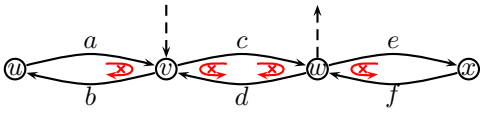

Figure 5: An instance of total separation on a FTG.

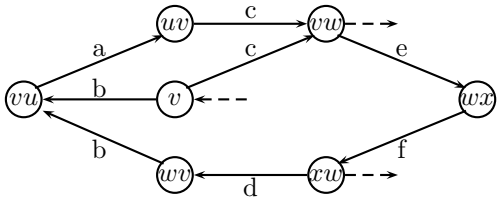

Figure 6: The automata recognising the associated FTG-reachable language.

Note in particular that since FTG-reachable languages are rational, they admit a regular expression and therefore, a restriction.

6.3. Reduction theorem and resolution

Irreducible walks in FTGs

In a usual graph, an elementary walk is a walk that does not use the same vertex more than once. Given an instance of traffic monitoring, we can define similarly an elementary word as a word that denotes an elementary walk. Note that with our definition, cycles are not elementary.

Elementary walks or words are exactly those that cannot be decomposed as a concatenation of three walks $W_1W_2W_3$ such that W_2 is non-empty and W_1W_3 is also a walk on the graph. Indeed, such a decomposition is possible if and only if W_1 and W_2 end on the same vertex, which means that the walk is non-elementary. Therefore, by iterating the deletion of W_2 until the walk is elementary (which will happen within a finite number of iterations since the length of the walk strictly decreases), one can extract from every walk W an elementary walk W' with the same start point and end point as W and that uses only vertices and edges that W uses.

On the graph G depicted in Figure 7, one can see that the walk $W = abcde$ is not elementary since it uses the vertex v twice. As explained previously, one can extract from W the walk $W' = ae$ that still goes from u to y and uses only vertices and arcs that W uses but is elementary. However, note that W' uses a transition (here, (a, e)) that W does not use. If we now forbid the transition (a, e), the vertices u and y can still be connected but not by an elementary walk (which cannot happen in usual graphs) and the walk W, even though non-elementary, does not observe the characterisation of non-elementary walks that we gave in the previous paragraph.

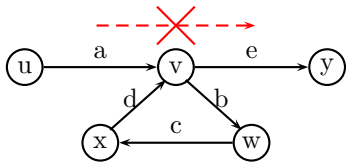

Figure 7: The graph G.

We define an *irreducible walk* in a FTG as a walk that cannot be decomposed as a concatenation of three walks $W_1W_2W_3$ such that W_2 is non-empty and W_1W_3 is a permitted walk on the graph. Hence, the walk $abcde$ in Figure 7 is irreducible. As before, by iterating the deletion of W_2, one can extract from any permitted walk W on a FTG an irreducible walk W' with same starting and endpoint as W and that uses only vertices or edges that W uses. We call such a walk W' a reduced form of W. Note that unlike in usual graphs, the reduced form of a walk in a FTG is not necessarily unique. Similarly, we can define irreducible walks with respect to some additional constraints. For example, if WX is a walk, we can define W' as a form of W which is as reduced as possible such that $W'X$ is still permitted. Hence, W' would not necessarily be irreducible but we know for sure that we could not iterate the decomposition and the deletion of W_2 without making a forbidden transition appear when we concatenate W' and X.

Preliminary results

Lemma 10. *For all rational languages L, for all restrictions $\overline{L}$ of L and for all words $u \in L \setminus \overline{L}$, there exists words v, w_1, w_2, w_3 and x such that $u = vw_1w_2w_3x$, w_1, w_2 and w_3 are all non-empty and u still belongs to L if we remove one or several of the w_i.*

Proof. We prove the lemma by induction on a regular expression describing the language L:

- if $L = \varnothing$, $L \setminus \overline{L} = \varnothing$ and the lemma holds on L (universal property on an empty set).
- if $L = \{u\}$ with $u \in A^*$, $L \setminus \overline{L} = \varnothing$ and the lemma holds.
- if $L = L_1 + L_2$ and the lemma holds for L_1 and L_2:
  $L \setminus \overline{L} = (L_1 + L_2) \setminus (\overline{L_1} + \overline{L_2}) \subseteq (L_1 \setminus \overline{L_1}) + (L_2 \setminus \overline{L_2})$ and the lemma holds for L.
- if $L = L_1L_2$ and the lemma holds for L_1 and L_2: let $u \in L_1$ and $v \in L_2$ and let us observe that for uv not to belong to $\overline{L_1L_2}$, it is necessary that $u \notin \overline{L_1}$ or $v \notin \overline{L_2}$. Hence, $L \setminus \overline{L} \subseteq (L_1 \setminus \overline{L_1})L2 + L1(L_2 \setminus \overline{L_2})$ and the lemma still holds for L.
- if $L = L'^*$ and the lemma holds for L': we set $M = L' \setminus \{\varepsilon\}$. Note that $L'^* = M^*$. We also set $\overline{M} = \overline{L'} - \varepsilon$.

$$\begin{aligned} L \setminus \overline{L} &= \left(\sum_{i\in\mathbb{N}} L'^i\right) \setminus \left(\sum_{i=0}^{2} \overline{L'}^i\right) = \left(\sum_{i\in\mathbb{N}} M^i\right) \setminus \left(\sum_{i=0}^{2} \overline{M}^i\right) \\ &\subseteq \sum_{i=0}^{2} (M^i \setminus \overline{M}^i) + \sum_{i\geqslant 3} M^i \\ &\subseteq \varnothing + (M \setminus \overline{M}) + \underbrace{M^2 \setminus \overline{M}^2}_{=M(M\setminus\overline{M})+(M\setminus\overline{M})M} + M^3M^* \end{aligned}$$

Since the lemma holds for all words of $M \setminus \overline{M} = L' \setminus \overline{L'}$, it holds for all words of $M\setminus\overline{M}+M^2\setminus\overline{M}^2$. Let us now prove it for $u \in M^3M^*$. By definition, u is the concatenation of $v = \varepsilon$, w_1, w_2 and $w_3 \in M$ (which are non-empty by construction of M) and $x \in M^*$. Hence, even if we remove some of the w, u is still a concatenation of words of M and therefore still belongs to $M^* = L$.

This proves the lemma. □

The reduction theorem

Even though reachable languages are strictly included in FTG-reachable languages, the reduction theorem still holds:

Theorem 11. *For all FTG-reachable languages L on an alphabet A, for all restrictions $\overline{L}$ of L, if $A' \subset A$ separates $\overline{L}$, then it separates L.*

Proof. Let $L \subseteq A^*$ be a FTG-reachable language, let $A' \subseteq A$ and $u, v \in L$ be such that $u \neq v$ but $p_{A'}(u) = p_{A'}(v) = a_0 \cdots a_n$. Thus, $u = u_0a_0u_1a_1 \cdots a_nu_{n+1}$ and $v = v_0a_0v_1a_1 \cdots a_nv_{n+1}$ where for all i, u_i and v_i belong to $(A \setminus A')^*$. We want to prove that for all restriction $\overline{L}$ of L, there exist two different words in $\overline{L}$ that have the same signature.

Since $u \neq v$, we know that there exists i such that $u_i \neq v_i$. Moreover, since L is FTG-reachable, by using Lemma 8 twice, we find that $u_0a_0 \cdots u_{i-1}a_{i-1}\boxed{v_i}\underbrace{a_iu_{i+1} \cdots u_{n+1}}_{=y}$ still belongs to L. To keep the notation as simple as possible, we set $y = a_iu_{i+1} \cdots u_{n+1}$.

Let x be the longest common prefix of u_i and v_i. Hence, $u_i = xbu'_i$ and $v_i = xcv'_i$ where b and $c \in (A \setminus A') \cup \{\varepsilon\}$ are the first letters of the suffix after x (or ε if it is empty). Thus, $b \neq c$ and b or c is empty if and only if $x = u_i$ or $x = v_i$ respectively (hence, if $b = \varepsilon$ then $u'_i = \varepsilon$ too and the same holds for c and v'_i). We also set $z = u_0 a_0 \cdots u_{i-1} a_{i-1} x$. Thus, $u = zbu'_i y$ and we know that $zcv'_i y \in L$ too.

Let $\mathrm{red}(z)$ be a form of z which is as reduced as possible such that both $\mathrm{red}(z)b$ and $\mathrm{red}(z)c$ are permitted (we iterate the reduction as long as it is possible). Similarly, let $\mathrm{red}(u'_i)$ and $\mathrm{red}(v'_i)$ be forms of u'_i and v'_i which are as reduced as possible such that $b\,\mathrm{red}(u'_i)$ and $c\,\mathrm{red}(v'_i)$ are permitted and let $\mathrm{red}(y)$ be a form of y which is as reduced as possible such that $\overline{u} = \mathrm{red}(z)b\,\mathrm{red}(u'_i)\,\mathrm{red}(y)$ and $\overline{v} = \mathrm{red}(z)c\,\mathrm{red}(v'_i)\,\mathrm{red}(y)$ are both permitted. Since $\overline{u}$ and $\overline{v}$ are permitted and have same origin and destination as u, they still belong in L.

First case: both $\overline{u}$ and $\overline{v}$ belong to every restriction $\overline{L}$ of L. We know that $b\,\mathrm{red}(u'_i)$ and $c\,\mathrm{red}(v'_i)$ cannot both be empty. If one of them is, they are thus necessarily distinct and if none of them are, we know that b and c denote different letters. In all cases, $b\,\mathrm{red}(u'_i) \neq c\,\mathrm{red}(v'_i)$ and thus, $\overline{u} \neq \overline{v}$.

By definition, a reduced form of a word only uses letters that the word itself uses. Since u_i and $v_i \in A \setminus A'$, we know that $b\,\mathrm{red}(u'_i)$ and $c\,\mathrm{red}(v'_i) \in (A \setminus A')^*$. Hence,

$$\begin{aligned} p_{A'}(\overline{u}) &= p_{A'}(\mathrm{red}(z))\,\underbrace{p_{A'}(b\,\mathrm{red}(u'_i))}_{=\varepsilon}\,p_{A'}(\mathrm{red}(y)) \\ &= p_{A'}(\mathrm{red}(z))\,\underbrace{p_{A'}(c\,\mathrm{red}(v'_i))}_{=\varepsilon}\,p_{A'}(\mathrm{red}(y)) \\ &= p_{A'}(\overline{v}) \end{aligned}$$

Second case: at least one of $\overline{u}$ and $\overline{v}$, say $\overline{u}$ does not belong to a restriction $\overline{L}$ of L. Hence, by Lemma 10, we know that $\overline{u} = sw_1w_2w_3t$ where w_1, w_2 and w_3 are all non-empty and can be removed. We recall that $\overline{u}$ can also be written $\mathrm{red}(z)b\,\mathrm{red}(u'_i)\,\mathrm{red}(y)$.

By definition, $\mathrm{red}(z)$ is a form of z which is as reduced as possible such that $\mathrm{red}(z)b$ and $\mathrm{red}(z)c$ are both permitted. This notably means that any reduction on z would change its last letter since it changes the letter we can write after it. Hence, a removable factor of $\mathrm{red}(z)$ is necessarily a suffix. Since we know by Lemma 10 that we can remove w_1 without making forbidden transitions appear, this proves in particular that w_1 cannot end strictly before $\mathrm{red}(z)$. Similarly, we can prove that a removable factor of $\mathrm{red}(y)$ is necessarily a prefix and therefore, that w_3 cannot begin strictly after $\mathrm{red}(y)$. Hence, we know that w_2 is a factor of $b\,\mathrm{red}(u'_i)$.

By definition of $\mathrm{red}(u'_i)$, we know that it does not contain any factor we can remove without making a forbidden transition appear with b. Therefore, since w_2 is removable and a factor of $b\,\mathrm{red}(u'_i)$, it has to contain b. Putting it all together, we find out that w_2 is a prefix of $b\,\mathrm{red}(u'_i)$, which means that w_1 is a suffix of $\mathrm{red}(z)$ and since w_1 is non-empty, that s is a strict prefix of $\mathrm{red}(z)$. Since w_3 is removable and $\mathrm{red}(u'_i)$ cannot contain a removable factor, we know that w_3 is the concatenation of a suffix of $\mathrm{red}(u'_i)$ and a non-empty prefix of $\mathrm{red}(y)$. Finally, we know that t is a strict suffix of $\mathrm{red}(y)$.

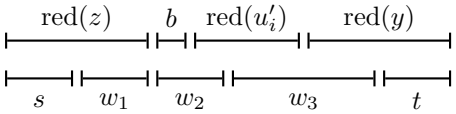

Figure 8: The two decompositions of $\overline{u}$.

Let $\mathrm{red}(w_2)$ be a form of w_2 which is as reduced as possible such that $\mathrm{red}(w_2)$ is non-empty (possible since w_2 is non-empty itself) and $s\,\mathrm{red}(w_2)t$ still denotes a permitted walk. By

Lemma 10 on $\overline{u} = sw_1w_2w_3t$, we know that st and sw_2t both belong to L and therefore, that $s\,\mathrm{red}(w_2)t \in L$ too.

- By definition, $\mathrm{red}(w_2)$ is non-empty so $st \neq s\,\mathrm{red}(w_2)t$.
- Since $\mathrm{red}(w_2)$ uses only letters that w_2 uses and w_2 is a factor of $b\,\mathrm{red}(u'_i) \in (A \setminus A')^*$, it has an empty signature and therefore, $p_{A'}(st) = p_{A'}(s\,\mathrm{red}(w_2)t)$.
- By definition, $\mathrm{red}(w_2)$ cannot contain a removable factor of $s\,\mathrm{red}(w_2)t$ unless it is removable itself. We also know that a removable factor of $\mathrm{red}(z)$ is necessarily a suffix and that s is a strict prefix of $\mathrm{red}(z)$, which means that s does not contain a removable factor. Finally, we know that a removable factor of $\mathrm{red}(y)$ is necessarily a prefix and since t is a strict suffix of $\mathrm{red}(y)$, it cannot contain one. This means that $s\,\mathrm{red}(w_2)t$ can contain at most two disjoint removable factors (one starting in s and finishing in w_2 and one starting in w_2 and finishing in t). Similarly, st can contain at most one removable factor. By Lemma 10, this means that st and $s\,\mathrm{red}(w_2)t$ belong to every reduction $\overline{L}$ of L.

We proved that for all FTG-reachable languages L on an alphabet A, for all subalphabets $A' \subseteq A$, if there exist $u \neq v$ in L such that $p_{A'}(u) = p_{A'}(v)$, then there also exist two words in every restriction $\overline{L}$ of L that are different but still have the same projection on A'. The contrapositive of this result is our theorem. □

Hence, the method we described at the end of Section 5 to solve the total separation problem on usual graphs also applies for FTGs: given an instance of FTG-reachable separation, we use the construction described in the proof of Proposition 9 to build an automata that recognises the associated language, we determine a regular expression of this language, restrict it and use the tools developed in Section 4 to solve the problem on the resulting language that is finite.

Finally, let us note that while the reduction theorem holds for reachable languages and even for the more general class of FTG-reachable languages, it does not hold for rational languages in general. For example, let $L = (ab)^* + ababa$ be a rational language (which is neither FTG-reachable nor reachable, we proved in the proof of Proposition 4 that a language that contains $ababa$ and that observes Lemma 3 has to contain aba and $abababa$ too which is not the case here). A restriction of L is $\overline{L} = \varepsilon + ab + abab + ababa$. One can see that the alphabet $A' = \{a\}$ separates $\overline{L}$ while $ababa$ and $ababab$, that both belong to L, have the same image under $p_{\{a\}}$.

Conclusion

We studied the problem of traffic monitoring from the point of view of separating codes. To overcome the limitations of this approach (as mentioned in Subsection 2.2), we introduced a new model of separation based on language and addressed the traffic monitoring with tools stemming from language theory. The problem of separation on a language being NP-complete in the size of the language, we outlined three subproblems relevant in practice, namely finite, total and FTG-reachable separations, and we designed algorithms to solve each of these subproblems, even if the set of routes to separate is infinite. The strength and flexibility of our model enabled us to address the case of cyclic graphs, infinite sets of roads to separate and even to impose additional constraints on the sets we separate such as avoiding certain transitions. The expressiveness of our new model of separation on a language and the limitations it overcomes also offer hope that it could be of help in a much wider range of applications than traffic monitoring alone.

Of course, this study also opens the door to many possibilities for improvement. In our opinion, the two most important ones are a deeper understanding of the ILP our reduction leads to and the study of divide-and-conquer algorithms. Other possibilities for further work include the investigation of subproblems other than finite or total separations on graphs or forbidden-transition graphs that could be more suitable for some practical applications.

Acknowledgements

I would like to thank my thesis advisor Arnaud Pêcher for his invaluable guidance for the writing of this paper and throughout the research that led to it as well as Feiran Yang and Stefan Bard for their assistance in the proofreading.

References

[1] Dean N. Arden. Delayed-logic and finite-state machines. In *Proceedings of the 2Nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1961)*, FOCS '61, pages 133–151, Washington, DC, USA, 1961. IEEE Computer Society.

[2] Gabriela R. Argiroffo, Silvia M. Bianchi, and Annegret Katrin Wagler. Polyhedra associated with identifying codes. *Electronic Notes in Discrete Mathematics*, 44:175–180, 2013.

[3] Koen M. J. De Bontridder, Bjarni V. Halldórsson, Magnús M. Halldórsson, Cor A. J. Hurkens, Jan Karel Lenstra, R. Ravi, and Leen Stougie. Approximation algorithms for the test cover problem. *Math. Program.*, 98(1-3):477–491, 2003.

[4] Koen M. J. De Bontridder, Bjarni V. Halldórsson, Magnús M. Halldórsson, Cor A. J. Hurkens, Jan Karel Lenstra, R. Ravi, and Leen Stougie. Approximation algorithms for the test cover problem. *Math. Program.*, 98(1-3):477–491, 2003.

[5] Emmanuel Charbit, Irène Charon, Gérard D. Cohen, Olivier Hudry, and Antoine Lobstein. Discriminating codes in bipartite graphs: bounds, extremal cardinalities, complexity. *Adv. in Math. of Comm.*, 2(4):403–420, 2008.

[6] Irène Charon, Gérard D. Cohen, Olivier Hudry, and Antoine Lobstein. Discriminating codes in (bipartite) planar graphs. *Eur. J. Comb.*, 29(5):1353–1364, 2008.

[7] Irène Charon, Olivier Hudry, and Antoine Lobstein. Minimizing the size of an identifying or locating-dominating code in a graph is np-hard. *Theor. Comput. Sci.*, 290(3):2109–2120, 2003.

[8] Zdenek Dvorak. Two-factors in orientated graphs with forbidden transitions. *Discrete Mathematics*, 309(1):104–112, 2009.

[9] Herbert Fleischner and Bill Jackson. Compatible path-cycle-decompositions of plane graphs. *J. Comb. Theory, Ser. B*, 42(1):94–121, 1987.

[10] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[11] Gregory Gutin and Eun Jung Kim. Properly coloured cycles and paths: Results and open problems. In *Graph Theory, Computational Intelligence and Thought, Essays Dedicated to Martin Charles Golumbic on the Occasion of His 60th Birthday*, pages 200–208, 2009.

[12] Teresa W. Haynes, Debra J. Knisley, Edith Seier, and Yue Zou. A quantitative analysis of secondary rna structure using domination based parameters on trees. *BMC Bioinformatics*, 7:108, 2006.

[13] Iiro S. Honkala, Tero Laihonen, and Sanna M. Ranto. On strongly identifying codes. *Discrete Mathematics*, 254(1-3):191–205, 2002.

[14] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation - international edition (2. ed)*. Addison-Wesley, 2003.

[15] Mamadou Moustapha Kanté, Fatima Zahra Moataz, Benjamin Momège, and Nicolas Nisse. Finding paths in grids with forbidden transitions. In *Graph-Theoretic Concepts in Computer Science - 41st International Workshop, WG 2015, Garching, Germany, June 17-19, 2015, Revised Papers*, pages 154–168, 2015.

[16] Mark G. Karpovsky, Krishnendu Chakrabarty, and Lev B. Levitin. On a new class of codes for identifying vertices in graphs. *IEEE Transactions on Information Theory*, 44(2):599–611, 1998.

[17] Jeong Han Kim, Oleg Pikhurko, Joel H. Spencer, and Oleg Verbitsky. How complex are random graphs in first order logic? *Random Struct. Algorithms*, 26(1-2):119–145, 2005.

[18] S. C. Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, 1956.

[19] Moshe Laifenfeld, Ari Trachtenberg, Reuven Cohen, and David Starobinski. Joint monitoring and routing in wireless sensor networks using robust identifying codes. *MONET*, 14(4):415–432, 2009.

[20] S. Maheshwari. Traversal marker placement problem are np-complete. In *Research report no CU-CS-092-76*, Dept. of Computer Science, University of Colorado at Boulder, 1976.

[21] Gernot Metze, Donald R. Schertz, Kilin To, Gordon Whitney, Charles R. Kime, and Jeffrey D. Russell. Comments on "derivation of minimal complete sets of test-input sequences using boolean differences. *IEEE Trans. Computers*, 24(1):108, 1975.

[22] Philippe Meurdesoif, Pierre Pesneau, and François Vanderbeck. Meter installation for monitoring network traffic. In *International Network Optimization Conference (INOC)*, Spa, Belgium, 2007.

[23] B. Moret and H. Shapiro. On minimizing a set of tests. *SIAM Journal on Scientific and Statistical Computing*, 6(4):983–1003, 1985.

[24] Patrenahalli M. Narendra and Keinosuke Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Trans. Computers*, 26(9):917–922, 1977.

[25] Suk Jai Seo and Peter J. Slater. Open neighborhood locating-dominating in trees. *Discrete Applied Mathematics*, 159(6):484–489, 2011.

[26] Suk Jai Seo and Peter J. Slater. Open neighborhood locating-domination for infinite cylinders. In *ACM Southeast Regional Conference*, pages 334–335, 2011.

[27] Stefan Szeider. Finding paths in graphs avoiding forbidden transitions. *Discrete Applied Mathematics*, 126(2-3):261–273, 2003.

[28] Rachanee Ungrangsi, Ari Trachtenberg, and David Starobinski. An implementation of indoor location detection systems based on identifying codes. In *INTELLCOMM*, pages 175–189, 2004.

[29] B. Holmes W. R. Willcox, S. P. Lapage. A review of numerical methods in bacterial identification. *Antonie van Leeuwenhoek*, 46(3):233–299, 1980.

[30] W. R. Willcox and S. P. Lapage. Automatic construction of diagnostic tables. *Comput. J.*, 15(3):263–267, 1972.