Complejidad y resolución práctica de razonamientos espacio-temporales para criminología

---

Complexity and practical resolution of spatial temporal reasonings for criminology

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

Trabajo de Fin de Grado

Doble Grado en Ingeniería Informática y Matemáticas

Mayo 2022

Autor:

Víctor Fernández González

Dirigido por:

Natalia López Barquilla

Ismael Rodríguez Laguna

Resumen

La criminología es una ciencia social que se centra en el estudio del fenómeno del crimen, con el objetivo de prevenir estas conductas y de detectar al culpable en caso de que ya hayan sucedido. Es un campo necesariamente ligado al avance de la tecnología, ya que por una parte los avances tecnológicos proporcionan más armas a los criminales, y por otro lado pueden ayudar a determinar quién es el autor de un delito, así como a evitar que sucedan crímenes futuros a partir del análisis de las conductas más frecuentes en ciertos tipos de actos delictivos. Estudios previos, como [1] o [2] han acercado este problema al mundo de la algoritmia y la computación, dando lugar a la emergente rama de investigación llamada *criminología computacional*, donde utilizando técnicas de topología computacional y machine learning se buscan patrones en distintos crímenes de la misma índole, prestando atención entre otras cosas a los movimientos de una serie de personas (criminales y no criminales) sobre un espacio a lo largo del tiempo, con el objetivo de prevenir esta clase de acontecimientos en el futuro.

En este trabajo abordamos el tema de la *criminología computacional* desde un punto de vista más centrado en la resolución e identificación del autor de un crimen en concreto, aportando un modelo formal que permita realizar deducciones espacio-temporales, y reconstruir los hechos de una escena del crimen a partir de los testimonios de las distintas personas involucradas, donde intentaremos detectar algorítmicamente si alguno de los testigos está necesariamente mintiendo o si los testimonios de todos ellos son compatibles entre sí. Realizaremos un primer análisis del problema desde la perspectiva de la teoría de la complejidad computacional, comprobando que decidir si algún testigo miente o todos dicen la verdad se trata de un problema *NP*-completo, aunque veremos cómo imponiendo una serie de restricciones sobre la formulación del problema podemos obtener algoritmos polinómicos que lo resuelven.

Además, estudiaremos detalladamente el problema de obtener una solución que verifique el máximo número de testimonios posible, que se trata de un problema *NPO*. Debido a la intratabilidad de algunos de estos problemas, es común estudiar algoritmos que proporcionen soluciones aproximadas en tiempo polinómico, dando lugar a la definición varias clases de problemas en función de la calidad de la aproximación obtenible. En concreto probamos que una variante de nuestro problema es APX-dura. Finalmente, proponemos diversos algoritmos voraces para obtener soluciones aproximadas del problema, e implementamos también un algoritmo genético con el mismo objetivo, para terminar realizando un análisis comparativo sobre la calidad de las aproximaciones obtenidas por los distintos métodos.

Palabras clave

Criminología, reconstrucción de hechos, complejidad computacional, optimización, aproximabilidad, NP-completitud, APX-dureza, algoritmo genético.

Abstract

Criminology is a social science that focuses on the study of the nature of crime, with the aim of preventing these conducts, and finding the offender in case they have already happened. It is a field inevitably linked to the technological evolution, since, on the one hand, technological advances provide criminals with more tools to commit a crime, and on the other hand, they can help to determine who the perpetrator of a crime is, as well as to prevent crimes from occurring in the first place. Previous studies, such as [1] or [2] have brought this problem closer to the world of algorithms and computation, giving rise to an emerging research area called *computational criminology*, where different techniques such as computational topology and machine learning are used to search for patterns in different crimes of the same nature, paying attention, among other things, to the locations of a series of people (both offenders and non-offenders) over a given space and time, with the aim of preventing this kind of events in the future.

In this work we approach the subject of computational criminology from a different point of view, more focused on the resolution and identification of the perpetrator of a particular crime, bringing a formal model that allows us to make spatiotemporal reasonings, and to reconstruct the events surrounding the crime scene, provided the testimonies of the different parties involved. We will study how to algorithmically detect if any of the witnesses is lying, or if all evidence is compatible among itself. We will perform a first analysis of the problem from the perspective of computational complexity theory, proving that deciding whether any witness is lying or all of them are telling the truth is an *NP*-complete problem, but we will see however how imposing certain constraints on the problem allows us to obtain algorithms that solve it in polinomial time.

Furthermore, we will study in detail the problem of obtaining a solution that verifies the maximum possible number of such testimonies, in order to obtain suboptimal solutions, which is an *NPO* problem. Due to the intractability of some of these problems, it is common to study algorithms that provide approximate solutions in polynomial time, giving rise to the definition of several classes of problems depending on the quality of the approximation achievable. In particular, we prove that a variant of our problem is APX-hard. Finally, we propose several greedy algorithms to obtain approximate solutions to the problem, and we also implement a genetic algorithm with the same objective, to later draw conclusions based on the comparison of the quality of the approximations obtained by the different methods.

Keywords

Criminology, reconstruction of events, computational complexity, optimization, approximability, NP-completeness, APX-hardness, genetic algorithm.

Índice

1 Introducción **1**

2 Teoría y Conceptos Previos **5**
2.1 Problemas de Decisión y Clases de Complejidad 5
2.2 Problemas de Optimización y Clases de Aproximabilidad 7

3 Definición del Problema y sus Variantes **13**

4 Complejidad del Problema de Decisión *WP* **22**
4.1 Pertenencia a *NP* 22
4.2 Análisis de la Complejidad de Distintas Versiones 28

5 Complejidad del Problema de Optimización *WPO* **55**
5.1 Pertenencia de *rest-WPO(1)* a APX-duro 55
5.2 Algoritmos Voraces 58

6 Resolución práctica **66**
6.1 Introducción a los Algoritmos Genéticos 66
6.2 Algoritmo Genético para el Problema *WPO(1)* 68
6.3 Resultados Obtenidos 73

Bibliografía **84**

Capítulo 1

Introducción

Motivación

A lo largo de los últimos años, el rápido avance de la tecnología ha supuesto una enorme revolución en nuestra forma de vida, generándonos la constante necesidad de adaptarnos al mundo cada vez más complejo en el que vivimos. Desde la perspectiva de la criminología se observa constantemente como surgen nuevas formas de delinquir, utilizando estas tecnologías a su favor para cometer delitos antes imposibles, así como por su parte la policía y detectives también cuentan con nuevos y mejores métodos de recolección de información y pruebas para llevar a cabo investigaciones más documentadas.

En este paradigma, donde cada vez existe más y más información, puede llegar a ser complicado para un ser humano llegar a ciertas deducciones complejas que involucren poner en común los datos de muchas fuentes diferentes, lo cuál puede suponer la diferencia entre que un criminal sea atrapado por la ley o no. Por ello en este trabajo nos planteamos cómo se pueden reconstruir algorítmicamente los eventos que tuvieron lugar en una escena del crimen, con el objetivo de poder llegar a razonamientos lógicos que garanticen la inocencia o culpabilidad de las distintas personas involucradas en él. Estudios previos, como [1] y [2] han abordado problemas similares en un campo de estudio emergente denominado *criminología computacional*, buscando predecir y prevenir futuros actos delictivos.

El problema en concreto que nos planteamos en este texto consiste en determinar la veracidad o falsedad de una serie de testimonios, dados por varios testigos, que hablan sobre las ubicaciones en un determinado espacio de distintas personas a lo largo de un intervalo de tiempo. Este problema, observado desde el punto de vista de la reconstrucción de hechos de una escena del crimen, pretende determinar si existen rutas para todas las personas involucradas que sean consistentes con lo que esas mismas personas hayan testificado, juntándolo posiblemente con otras fuentes de información como cámaras de seguridad, alarmas, y otro tipo de evidencias que permitan ser formalizadas utilizando nuestro lenguaje, todo esto sujeto siempre a la posibilidad de que alguna de las fuentes podría ser errónea o estar mintiendo.

Objetivos

Abordaremos este problema desde la teoría de complejidad computacional, siendo nuestro principal objetivo clasificar distintas variantes del problema en función de su dureza y pertenencia en distintas clases de complejidad, obteniendo resultados sobre la intratabilidad e inaproximabilidad del problema, al mismo tiempo que encontrando restricciones suficientes del enunciado del problema para que este pueda ser resuelto en tiempo polinómico.

Adicionalmente, propondremos e implementaremos algoritmos voraces y algoritmos genéticos para obtener soluciones aproximadas también de una versión más restringida del problema, con las que experimentaremos en la práctica analizando los resultados obtenidos y comparando cuáles de ellos resultan más eficaces, en términos estadísticos.

Estructura

El trabajo está constituido por esta memoria junto con el código implementado, que puede encontrarse en https://github.com/victorchuu/TFG_informatica. La memoria se organiza en seis capítulos:

1. Introducción al tema que trataremos durante el trabajo, motivación, objetivos, estructura y metodología.
2. Introducción a los conceptos previos de teoría de complejidad que utilizaremos en el resto de capítulos.
3. Definición del problema a tratar y de distintas versiones del mismo.
4. Estudio del problema de decisión de decidir si todos los testimonios son compatibles o no.
5. Estudio del problema de optimización de encontrar el mayor conjunto de testigos diciendo la verdad simultáneamente.
6. Resolución práctica del problema de optimización mediante algoritmos genéticos, y comparación estadística con otros métodos voraces.

Metodología

Como se puede apreciar a lo largo del trabajo, el método utilizado para abordar el tema consiste en comenzar analizando el comportamiento de las versiones más restringidas y simples del problema, para posteriormente ir introduciendo nuevos elementos que incrementen la dificultad y adaptando los resultados obtenidos previamente a estas modificaciones. Esto lleva a un desarrollo incremental del trabajo, estudiando diversas variaciones del problema por el camino.

Introduction

Motivation

Over the last few years, the accelerated advance of technology has brought about an enormous revolution in everyday life, generating a constant need to adapt to the increasingly complex world in which we live. In the field of criminology we are constantly witnessing how new kinds of delinquency emerge, using the newest technologies to their advantage, as well as police and detectives also have access to newer and improved methods of gathering evidence and information, which leads to more complex investigation processes.

The huge amounts of information makes it almost imposible for human beings to reach complex deductions that involve pooling data from many different sources, which can mean the difference between a criminal ending up arrested or not, so in this paper we will consider how to algorithmically reconstruct the events that occurred during given a crime scene, in order to reach logical reasonings to ensure the innocence or culpability of the parties involved. Previous studies, such as [1] and [2] have addressed similar problems, studying an emerging branch of computer science called *computational criminology*, aiming to predict and prevent future criminal acts.

The particular problem posed in this text consists on determining the veracity or falsity of a series of testimonies, gathered from several witnesses, who report the locations of different people in a certain space over a period of time. Looking at the problem from the perspective of criminology, it aims to determine whether there exist routes for all involved people which are consistent with what those same people testified, as well as possibly other sources of information such as security cameras, alarms, and any other kind of evidence that can be formalized using our language, subject to the possibility that some of the sources may be lying.

Objectives

We will approach this problem in terms of computational complexity theory. Our main objective will be to classify different variants of the problem according to their hardness and membership in different complexity classes, obtaining intractability and inapproximability results, while finding sufficient constraints on the problem statement for it to be solved in

polynomial time.

Additionally, we will be proposing and implementing both greedy algorithms and genetic algorithms to obtain approximate solutions for a restricted version of the problem, to experiment in practice and analyze the obtained results, comparing which ones seem to work better from a statistical point of view.

Structure

The project consists of this report together with the implemented code, which can be found at https://github.com/victorchuu/TFG_informatica. The paper is organized into six chapters as follows:

1. Introduction to the problem studied in the paper, our objectives and the metodology used.
2. Introduction to the notions of complexity theory that will be required to understand the following chapters.
3. Definition of the problem to be addressed and its different variants.
4. Analysis of the decision problem of determining whether all the testimonies are compatible or not.
5. Study of the optimization problem of finding the largest set of witnesses simultaneously telling the truth.
6. Implementation of a solution to the optimization problem using genetic algorithms, statistical comparison with some greedy methods.

Methodology

Throughout the paper, we undertake the study of a complex problem through the analysis of multiple simplifications of the original question. We then gradually increase the difficulty of these relaxations of the problem and adjust the previously obtained results to match the introduction of new elements. By using this approach, many interesting variations of the problem arise naturally on our road to answer the major question.

Capítulo 2

Teoría y Conceptos Previos

Comenzamos el trabajo definiendo los conceptos elementales que utilizaremos de la teoría de complejidad computacional, en términos de la cual hablaremos durante el resto de secciones. Definiremos las clases de problemas de decisión P, NP y hablaremos de la gran importancia que tienen en la teoría de la computación.
Introducimos también los problemas de optimización NPO, así como las distintas clases de aproximabilidad, y estudiaremos los métodos para demostrar la pertenencia y la dureza en una cierta clase. La información respecto a las clases P y NP ha sido obtenida de [3], mientras que la teoría sobre problemas de optimización y clases de aproximabilidad está basada en [4], [5] y [6].

2.1 Problemas de Decisión y Clases de Complejidad

Comenzamos definiendo uno de los conceptos más básicos con el que trataremos a lo largo de todo el trabajo, los problemas de decisión, que son aquellos para los que dada una instancia $\mathcal{I}$ su solución es "sí" o "no". Los inputs del problema pueden ser interpretados como cadenas sobre un alfabeto, en particular sobre el alfabeto binario $\{0,1\}^*$. Como la solución de los problemas es "si" o "no", podemos definir formalmente los problemas de la siguiente forma:

Definición 2.1 *(Problema de decisión) Un problema de decisión es una función definida sobre el alfabeto binario* $A : \{0,1\}^* \longrightarrow \{0,1\}$*, que transforma instancias* $\mathtt{x} \in \{0,1\}^*$ *del problema en* 1 *("si") o* 0 *("no").*

Podemos identificar esta función A *con el lenguaje* $L_A = \{\mathtt{x} \in \{0,1\}^* \mid A(\mathtt{x}) = 1\}$*, formado por las instancias cuya solución es afirmativa.*

Estos problemas se pueden clasificar en función de su complejidad, según el tiempo necesario para ser resueltos, teniendo especial interés las clases P y NP, definidas como sigue:

Definición 2.2 *(Clase P). La clase P es el conjunto de todos los problemas de decisión A que pueden ser resueltos en tiempo polinómico por una máquina de Turing determinista. Es decir, $A \in P$ si existen una máquina de Turing M y un polinomio p tales que para cualquier instancia $\mathtt{x} \in \{0,1\}^*$ la máquina de Turing decide si $\mathtt{x}$ está en L_A o no en un número de pasos menor o igual que $p(|\mathtt{x}|)$.*

Definición 2.3 *(Clase NP). La clase NP es el conjunto de todos los problemas de decisión A que pueden ser verificados en tiempo polinómico por una máquina de Turing determinista. Es decir, $A \in NP$ si existen una máquina de Turing M y polinomios p y q tales que:*

1. $\mathtt{x} \in \{0,1\}^*$ *pertencece a* L_A *si y solo si existe* $\mathtt{y} \in \{0,1\}^*$ *tal que* $|\mathtt{y}| \leq q(|\mathtt{x}|)$ *y* $M(\mathtt{x},\mathtt{y}) = 1$.
2. $\forall \mathtt{x}, \mathtt{y} \in \{0,1\}^*$ *la máquina de Turing M tarda como mucho* $p(|\mathtt{x}|+|\mathtt{y}|)$ *pasos en ejecutar.*

Existen definiciones alternativas para la clase *NP*, pero ésta es la que emplearemos nosotros para demostrar que nuestro problema pertenece a esta clase. Se comprueba fácilmente que $P \subseteq NP$ (basta con tomar la misma máquina de Turing que aparece en la definición de *P*, y considerar que $\mathtt{y} = \varepsilon$, la palabra vacía), pero la pregunta de si se trata de una inclusión estricta, o si se cumple la igualdad $P = NP$ sigue sin estar resuelta actualmente, y es uno de los problemas más importantes en teoría de la computación, puesto que poder resolver ciertos problemas *NP* en tiempo polinómico tendría grandes implicaciones en muchos campos científicos diferentes.

Sin embargo, existen numerosos resultados condicionales que confirman que, dados dos problemas de decisión *A* y *B*, si el problema *B* fuese resoluble en tiempo polinómico, entonces el problema *A* también podría ser resuelto en tiempo polinómico. Estos resultados reciben el nombre de reducciones polinómicas, y consisten en tomar instancias de *A* y transformarlas en instancias de *B* de tal forma que la primera tenga solución si y solo si lo tiene la segunda.

Definición 2.4 *($\leq_p$) Dados dos problemas de decisión A y B decimos que A se reduce polinómicamente a B, y escribimos $A \leq_p B$ si existe una función $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ computable en tiempo polinómico que transforma instancias $\mathtt{x}$ de A en instancias $f(\mathtt{x})$ de B, de tal forma $\mathtt{x} \in L_A$ si y solo si $f(\mathtt{x}) \in L_B$.*

Esto permite definir dos nuevas clases de problemas de decisión de suma importancia en teoría de la computación: *NP*-duro y *NP*-completo.

Definición 2.5 *(NP-duro) Un problema A es NP-duro si todos los problemas A' en NP se reducen polinómicamente a él ($A' \leq_p A$).*

Definición 2.6 *(NP-completo) Un problema A es NP-completo si*

- $A \in NP$
- *A es NP-duro*

Las siguientes dos observaciones ilustran la importancia que tienen estas clases de problemas:

- Si se demostrara que un problema $A \in NP$-duro pertenece a P, es decir, puede ser resuelto en tiempo polinómico, entonces todos los problemas de NP podrían ser resueltos también en tiempo polinómico y se cumpliría $P = NP$.
- Si se demostrara que un problema $A \in NP$-completo no pertenece a P, no sólo está consiguiendo probar que $P \neq NP$, sino que ningún otro problema NP-completo pertenece a P.

El siguiente teorema nos permitirá demostrar la NP-dureza de los diferentes problemas que estudiaremos.

Teorema 1 *Si A,B y C son problemas de decisión tales que $A \leq_p B$ y $B \leq_p C$, entonces se verifica que $A \leq_p C$. En otras palabras, la relación $\leq_p$ es transitiva.*

Demostración: Como $A \leq_p B$ y $B \leq_p C$ existen funciones $f, g : \{0,1\}^* \longrightarrow \{0,1\}^*$ computables en tiempo polinómico tales que

$$\mathtt{x} \in L_A \Leftrightarrow f(\mathtt{x}) \in L_B \quad \text{y} \quad \mathtt{x} \in L_B \Leftrightarrow g(\mathtt{x}) \in L_C$$

La función $g \circ f : \{0,1\}^* \longrightarrow \{0,1\}^*$ es computable en tiempo polinómico, basta con computar primero f y posteriormente g, y cumple la condición de que

$$\mathtt{x} \in L_A \Leftrightarrow f(\mathtt{x}) \in L_B \Leftrightarrow g(f(\mathtt{x})) = (g \circ f)(\mathtt{x}) \in L_C \quad \square$$

Por lo tanto, para demostrar que un determinado problema A es NP-duro, basta con tomar otro problema B que ya sepamos que es NP-duro, y probar la reducción $B \leq_p A$. De esta forma, para cualquier otro problema $C \in NP$, se tiene que $C \leq_p B$ por la dureza de B, y consecuentemente se obtiene $C \leq_p A$ por transitividad.

2.2 Problemas de Optimización y Clases de Aproximabilidad

Además de los problemas de decisión trataremos con problemas de optimización, en los que se pretende maximizar o minimizar un cierto valor sujeto a una serie de restricciones.

Definición 2.7 *Un problema de optimización es una tupla* $\mathcal{P} = (\mathcal{I}, \texttt{Sol}, \texttt{m}, goal)$ *donde:*

- $\mathcal{I}$ *es el conjunto de instancias del problema.*
- $\texttt{Sol}$ *es una función que asocia a cada instancia* $\texttt{x} \in \mathcal{I}$ *un conjunto de soluciones factibles* $\texttt{Sol}(\texttt{x})$.
- $\texttt{m}$ *es la función que asocia a cada solución de una instancia un número racional, que será el valor a maximizar o minimizar. Si* $\texttt{x} \in \mathcal{I}$ *y* $\texttt{y} \in \texttt{Sol}(\texttt{x})$, *entonces* $\texttt{m}(\texttt{x}, \texttt{y}) \in \mathbb{Q}$.
- $goal \in \{max, min\}$ *indica si se trata de un problema de maximización o minimización.*

Dada una instancia de un problema de optimización, definimos su valor óptimo como el mejor valor (máximo o mínimo) alcanzable por alguna solución factible.

Definición 2.8 *Sea* $\mathcal{P} = (\mathcal{I}, \texttt{Sol}, \texttt{m}, goal)$ *un problema de optimización,* $\texttt{x} \in \mathcal{I}$ *una instancia del problema*

$$\texttt{opt}(\texttt{x}) := goal\{\texttt{m}(\texttt{x}, \texttt{y}) \mid \texttt{y} \in \texttt{Sol}(\texttt{x})\}$$

Una solución $\overline{\texttt{y}} \in \texttt{Sol}(\texttt{x})$ *es óptima si* $\texttt{m}(\texttt{x}, \overline{\texttt{y}}) = \texttt{opt}(\texttt{x})$.

De forma similar a los problemas de decisión, vamos a caracterizar los problemas de optimización en dos clases, *PO* y *NPO*, en función de su complejidad.

Definición 2.9 *(NPO) Un problema de optimización* $\mathcal{P} = (\mathcal{I}, \texttt{Sol}, \texttt{m}, goal)$ *pertenece a la clase NPO si:*

1. *El conjunto de instancias* $\mathcal{I}$ *es reconocible en tiempo polinómico, es decir, el problema de decisión que consiste en determinar si* $\texttt{x}$ *pertenece o no a* $\mathcal{I}$, *es un problema de la clase P.*
2. *Dada una instancia* $\texttt{x} \in \mathcal{I}$, *se puede computar en tiempo polinómico si una solución* $\texttt{y}$ *es o no es factible (si pertenece o no a* $\texttt{Sol}(\texttt{x})$*).*
3. *Dada una instancia* $\texttt{x} \in \mathcal{I}$, *se puede obtener al menos una solución factible en tiempo polinómico.*
4. *La función* $\texttt{m}$ *se puede computar en tiempo polinómico.*

Definición 2.10 *(PO) Un problema de optimización pertenece a PO si pertenece a NPO y existe un algoritmo que dada cualquier instancia* $\texttt{x} \in \mathcal{I}$ *calcula en tiempo polinómico una solución óptima* $\overline{\texttt{y}} \in \texttt{Sol}(\texttt{x})$.

Los problemas *NPO* surgen de forma natural en distintos ámbitos y tienen muchas aplicaciones en el mundo real. Puesto que no conocemos métodos eficientes (polinómicos) para resolver muchos de estos problemas, ni parece que los vayamos a hallar en un futuro cercano, una posible orientación consiste en buscar algoritmos que en tiempo polinómico obtengan una solución que aproxime la óptima. Desafortunadamente, aquí nos chocamos con otra

barrera, pues la calidad de estas aproximaciones también parece presentar límites teóricos, y se pueden definir distintas subclases de la clase *NPO*, llamadas clases de aproximabilidad, en función del ratio de aproximación alcanzable. A continuación damos la definición de algunas de estas clases, así como las de dureza en las mismas clases y explicamos métodos para demostrar la pertenencia y dureza.

Definición 2.11 *Dado un problema de optimización* $\mathcal{P} = (\mathcal{I}, \mathtt{Sol}, \mathtt{m}, goal)$*, una instancia* $\mathtt{x} \in \mathcal{I}$ *y una solución factible* $\mathtt{y} \in \mathtt{Sol}(\mathtt{x})$*, definimos el ratio de aproximabilidad de la solución* y *respecto a* x *como:*

$$R_{\mathcal{P}}(\mathtt{x}, \mathtt{y}) = max\left\{ \frac{\mathtt{m}(\mathtt{x}, \mathtt{y})}{\mathtt{opt}_{\mathcal{P}}(\mathtt{x})}, \frac{\mathtt{opt}_{\mathcal{P}}(\mathtt{x})}{\mathtt{m}(\mathtt{x}, \mathtt{y})} \right\}$$

Nótese que, aunque esta es la forma general de definirlo, en problemas de maximización se tendrá

$$R_{\mathcal{P}}(\mathtt{x}, \mathtt{y}) = \frac{\mathtt{opt}_{\mathcal{P}}(\mathtt{x})}{\mathtt{m}(\mathtt{x}, \mathtt{y})}$$

y en los de minimización

$$R_{\mathcal{P}}(\mathtt{x}, \mathtt{y}) = \frac{\mathtt{m}(\mathtt{x}, \mathtt{y})}{\mathtt{opt}_{\mathcal{P}}(\mathtt{x})}$$

Además, el ratio de aproximabilidad es un número perteneciente a $[1, \infty)$, y si $R_{\mathcal{P}}(\mathtt{x}, \mathtt{y}) = 1$ entonces $\mathtt{y} \in \mathtt{Sol}(\mathtt{x})$ es una solución óptima del problema.

Definición 2.12 *Decimos que un algoritmo aproximado* A *para un problema de optimización* $\mathcal{P} = (\mathcal{I}, \mathtt{Sol}, \mathtt{m}, goal)$ *es una función que toma una instancia* $\mathtt{x} \in \mathcal{I}$ *y devuelve una solución factible* $\mathtt{A}(\mathtt{x}) \in \mathtt{Sol}(\mathtt{x})$*.*

Bajo estas definiciones, estamos listos para definir las distintas clases de aproximabilidad.

Definición 2.13 *Sea* $\mathcal{P} = (\mathcal{I}, \mathtt{Sol}, \mathtt{m}, goal)$ *un problema NPO*

- $\mathcal{P}$ *pertenece a la clase* ***Exp-APX*** *si existen un algoritmo polinómico* A *y un polinomio* p *tal que*

$$R_{\mathcal{P}}(\mathtt{x}, \mathtt{A}(\mathtt{x})) \leq 2^{p(|\mathtt{x}|)} \qquad \forall \mathtt{x} \in \mathcal{I}$$

- $\mathcal{P}$ *pertenece a la clase* ***Poly-APX*** *si existen un algoritmo polinómico* A *y un polinomio* p *tal que*

$$R_{\mathcal{P}}(\mathtt{x}, \mathtt{A}(\mathtt{x})) \leq p(|\mathtt{x}|) \qquad \forall \mathtt{x} \in \mathcal{I}$$

- $\mathcal{P}$ *pertenece a la clase* ***Log-APX*** *si existe un algoritmo polinómico* A *tal que*

$$R_{\mathcal{P}}(\mathtt{x}, \mathtt{A}(\mathtt{x})) \in O(log|\mathtt{x}|) \qquad \forall \mathtt{x} \in \mathcal{I}$$

- $\mathcal{P}$ *pertenece a la clase* ***APX*** *si existen un algoritmo polinómico* $\mathtt{A}$ *y una constante* $r > 1$ *tal que*

$$R_{\mathcal{P}}(\mathtt{x}, \mathtt{A}(\mathtt{x})) \leq r \qquad \forall \mathtt{x} \in \mathcal{I}$$

- $\mathcal{P}$ *pertenece a la clase* ***PTAS*** *si para todo* $r > 1$ *existe un algoritmo polinómico* $\mathtt{A}$ *tal que*

$$R_{\mathcal{P}}(\mathtt{x}, \mathtt{A}(\mathtt{x})) \leq r \qquad \forall \mathtt{x} \in \mathcal{I}$$

  Cabe esperar que para valores de r *máx próximos a* 1 *el algoritmo puede aumentar de complejidad, posiblemente de forma exponencial en* $\frac{1}{r-1}$.

- $\mathcal{P}$ *pertenece a la clase* ***FPTAS*** *si existen un algoritmo* $\mathtt{A}(\mathtt{x}, r)$ *polinómico tanto en* $|\mathtt{x}|$ *como en* $\frac{1}{r-1}$ *tal que para todo* $r > 1$

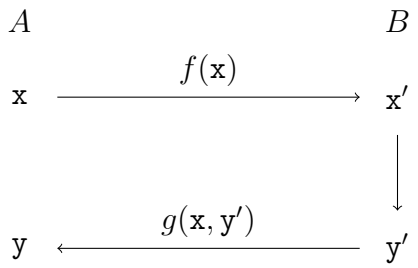
$$R_{\mathcal{P}}(\mathtt{x}, \mathtt{A}(\mathtt{x}, r)) \leq r \qquad \forall \mathtt{x} \in \mathcal{I}$$

Se tiene la siguiente cadena de inclusiones sobre las clases definidas

PO $\subseteq$ **FPTAS** $\subseteq$ **PTAS** $\subseteq$ **APX** $\subseteq$ **Log-APX** $\subseteq$ **Poly-APX** $\subseteq$ **Exp-APX** $\subseteq$ **NPO**

Una pregunta razonable que nos podemos hacer es, igual en el caso de P y NP, si estas inclusiones son o no son estrictas. Existen resultados que afirman que si alguna de estas inclusiones fuese una igualdad, entonces $P = NP$, y por lo tanto $PO = NPO$ y todas estas subclases convergerían en una sola.

Para determinar a qué clase de aproximabilidad pertenece un problema NPO, podemos utilizar de nuevo reducciones desde otro problema. Sin embargo, no siempre nos van a servir las mismas reducciones polinómicas anteriormente vistas, ya que antes sólo buscabamos una respuesta de "sí" o "no", pero ahora la respuesta que buscamos es una solución factible del problema original, como explica el siguiente esquema.



Es decir, si supiéramos resolver el problema B y quisiéramos utilizarlo para resolver el problema A, buscaríamos una función $f : \mathcal{I}_a \longrightarrow \mathcal{I}_b$ que transforme instancias de A, $\mathtt{x} \in \mathcal{I}_A$, en instancias de B, $\mathtt{x}' = f(\mathtt{x}) \in \mathcal{I}_B$.

Así podríamos obtener una solución $\mathtt{y}' \in \mathtt{Sol}(\mathtt{x}')$ del problema B, que necesitaríamos transformar de vuelta a una solución del problema A, mediante otra función $g(\mathtt{x}, \cdot) : \mathtt{Sol}_B(f(\mathtt{x})) \longrightarrow \mathtt{Sol}_A(\mathtt{x})$ que lleva $\mathtt{y}'$ a una solución factible de A, $\mathtt{y} = g(\mathtt{x}, \mathtt{y}')$.

Además, para que la reducción nos sea útil, debe existir una relación entre el ratio de aproximación del problema B (R_B) y el del problema A (R_A) con el objetivo de que nos permita garantizar que si B pertenece a una clase **C** entonces A también pertenece a la misma clase.

Distintos autores han propuesto diversas definiciones de reducciones que preservan la aproximabilidad, teniendo cada una sus ventajas y desventajas. Nosotros, para el contexto de este problema, vamos a tomar tan solo una de ellas, la reducción AP, que preserva la pertenencia a todas las clases [5].

Definición 2.14 *Dados los problemas de optimización* $A = (\mathcal{I}_A, \mathtt{Sol}_A, \mathtt{m}_A, goal_A)$ *y* $B = (\mathcal{I}_B, \mathtt{Sol}_B, \mathtt{m}_B, goal_B)$ *decimos que* A *es AP-reducible a* B*, y escribimos* $A \leq_{AP} B$ *si existen funciones* f *y* g*, y una constante* $c \geq 1$ *tales que:*

- *Si* $\mathtt{x} \in \mathcal{I}_A$ *entonces* $\mathtt{x}' = f(\mathtt{x}) \in \mathcal{I}_B$ *y* f *se puede computar en tiempo polinómico.*
- *Si* $\mathtt{x} \in \mathcal{I}_A$, $\mathtt{y}' \in \mathtt{Sol}_B(f(\mathtt{x}))$ *entonces* $g(\mathtt{x}, \mathtt{y}') \in \mathtt{Sol}_A(\mathtt{x})$ *y* g *se puede computar en tiempo polinómico.*
- *Para toda instancia* $\mathtt{x} \in \mathcal{I}_A$*, para todo* $r > 1$ *y para toda solución de la instancia transformada* $\mathtt{y}' \in \mathtt{Sol}_B(f(\mathtt{x}))$ *se cumple, denotando* $\mathtt{x}' = f(\mathtt{x}) \in \mathcal{I}_B$, $\mathtt{y} = g(\mathtt{x}, \mathtt{y}') \in \mathtt{Sol}_A(\mathtt{x})$*, que*

$$R_B(\mathtt{x}', \mathtt{y}') \leq r \quad \text{implica} \quad R_A(\mathtt{x}, \mathtt{y}) \leq 1 + c(r-1)$$

Nótese que, con la notación utilizada, $\mathtt{x}$ e $\mathtt{y}$ son la instancia y solución del problema A, mientras que $\mathtt{x}'$ e $\mathtt{y}'$ lo son del problema B.
A continuación podemos dar la definición de dureza y completitud en una clase de aproximabilidad. Esta definición depende del tipo de reducción elegida.

Definición 2.15 *Sea* $\boldsymbol{C}$ *una clase de problemas NPO, y* $\mathcal{P}$ *un problema de optimización, decimos que*

- $\mathcal{P}$ *es* $\boldsymbol{C}$*-duro (bajo la reducción AP) si para todo problema* $\mathcal{P}' \in \boldsymbol{C}$ *se tiene* $\mathcal{P}' \leq_{AP} \mathcal{P}$*.*
- $\mathcal{P}$ *es* $\boldsymbol{C}$*-completo (bajo la reducción AP) si es* $\boldsymbol{C}$*-duro y* $\mathcal{P} \in \boldsymbol{C}$*.*

El siguiente lema, que no demostraremos, es el que nos permitirá clasificar en que clase de complejidad se encuentran las distintas variantes de nuestro problema.

Lema 2.1 *Sea $\boldsymbol{C}$ una clase de problemas NPO, y A, B dos problemas de optimización tales que $A \leq_{AP} B$.*

- *Si $B \in \boldsymbol{C}$ entonces $A \in \boldsymbol{C}$.*
- *Si A es $\boldsymbol{C}$-duro, entonces B también lo es.*

Para demostrar la pertenencia a una clase no es necesario recurrir a este lema, puesto que será más fácil por lo general construir explícitamente un algoritmo que tenga el ratio de aproximabilidad deseado. Sin embargo, utilizaremos esto para probar la dureza en una clase.

Además, se puede comprobar que las versiones completas de las clases vistas anteriormente son disjuntas dos a dos a menos que $P = NP$. Por lo tanto la pertenencia a una clase $\mathbf{C}$-completa garantiza que no pertenece a otra clase $\mathbf{C}'$-completa distinta y clasifica completamente a un problema.

Para comprobar este hecho, suponemos que tenemos dos clases de las anteriormente definidas $\mathbf{C}_1 \subseteq \mathbf{C}_2$, para las que sabemos que $\mathbf{C}_1 \neq \mathbf{C}_2$ a menos que $P = NP$. Supongamos que existe un problema $\mathcal{P}$ que es al mismo tiempo $\mathbf{C}_1$-completo y $\mathbf{C}_2$-completo.
Entonces en particular $\mathcal{P} \in \mathbf{C}_1$, y por otro lado para todo problema $\mathcal{P}' \in \mathbf{C}_2$ se tiene que $\mathcal{P}' \leq_{AP} \mathcal{P}$.
Puesto que, de acuerdo a lo que hemos comentado, la reducibilidad AP preserva la permanencia en todas las clases definidas, entonces $\forall \mathcal{P}' \in \mathbf{C}_2$ obtenemos también que $\mathcal{P}' \in \mathbf{C}_1$, es decir, $\mathbf{C}_2 \subseteq \mathbf{C}_1$, resultando por lo tanto en que $\mathbf{C}_1 = \mathbf{C}_2$, lo que hemos visto que implica $P = NP$.

Es decir, $\mathbf{C}_1$-completo $\cap$ $\mathbf{C}_2$-completo $= \varnothing$ a menos que $P = NP$.

Capítulo 3

Definición del Problema y sus Variantes

A lo largo de este capítulo definiremos el problema a estudiar, y diversas variaciones en el mismo, con el objetivo de estudiar en los próximos capítulos la complejidad de estas distintas versiones.

El problema simula los movimientos de un conjunto de personas a lo largo del tiempo por un espacio determinado. Diversos testigos enuncian afirmaciones sobre ellos, indicando cuál es su ubicación en la representación espacial a una cierta hora, dando lugar a una amplia variedad de testimonios que describiremos con más detalle a continuación. El objetivo será determinar cuántos testigos pueden estar diciendo la verdad a la vez, y cuántos de ellos están mintiendo necesariamente.

Comenzamos describiendo la representación del espacio que emplearemos, muy simple, pero a la vez versátil: un grafo dirigido y valorado, donde el valor de cada arista indica el tiempo que se tarda en recorrerla.

Notación 1 *Tomamos un grafo $G = (V, E)$ dirigido y valorado, donde $V = \{v_1, \ldots, v_n\}$ son los distintos lugares donde pueden estar nuestros testigos, y cada arista $(u, v) \in E$ tiene un coste asociado no nulo $d(u, v) \in \mathbb{N}^+$.*

Dados dos vértices u, v cualesquiera del grafo, definimos $\overline{d}(u, v)$ como la distancia mínima entre ambos vértices, es decir:

$$\overline{d}(u,v) = min\left\{\sum_{k=2}^{s} d(w_{k-1}, w_k) \mid w_1, \ldots, w_s \text{ es un camino que une } u \text{ y } v\right\}$$

Si el conjunto fuera vacío decimos que $\overline{d}(u, v) = \infty$.

Las soluciones factibles del problema consistirán en dar la ruta que llevó a cabo cada actor, en forma de un camino en el grafo con posibles vértices repetidos, al que añadimos la posibilidad

de quedarse durante un tiempo parado en un vértice, puesto que si tomáramos la definición habitual de camino, estaríamos obligando a que los actores tengan que abandonar un vértice nada mas llegar a él. Por simplicidad en la representación, los actores no se podrán detener en mitad de una calle (arista) y siempre tardan un tiempo fijo en recorrerla.

Definición 3.1 *Sea un grafo $G = (V, E)$, diremos que una* ***ruta*** *es una sucesión finita de pares vértice-tiempo, con vértices adyacentes:*

$$R = \big\{(w_k, t_k) \mid k \in \{1, \ldots, s\}\big\}$$

donde cada $w_k \in V$, $t_k \in \mathbb{N}$, siendo t_k el tiempo que se espera en cada vértice, y los vértices consecutivos son adyacentes, es decir $\forall k \in \{2, \ldots, s\}$ $(w_{k-1}, w_k) \in E$. Muchas veces escribiremos las rutas simplemente como:

$$R = (w_1, t_1)(w_2, t_2) \cdots (w_s, t_s)$$

Definimos también la hora de llegada a cada vértice $first_R()$, y la hora de salida $last_R()$, de forma mútuamente recursiva:

$$\begin{cases} first_R(1) = 0 \\ last_R(k) = first_R(k) + t_k \\ first_R(k) = last_R(k-1) + d(w_{k-1}, w_k) \end{cases}$$

Definimos además una función auxiliar $W_R : \mathbb{N} \longrightarrow V$ que nos indicará en qué vértice del grafo se encuentra la ruta R en un determinado instante del tiempo t, es decir, $W_R(t) = w_k$ si se cumple que $t \in [first_R(k), last_R(k)]$.

Nótese que W_R puede no estar definido para todos los valores de entrada. Concretamente no estará definido si la persona se encuentra en una arista del grafo, trasladándose a otro vértice, o si ya ha finalizado su ruta. Sin embargo, cuando la función está definida, toma un solo valor. Esto se debe a que las distancias de las aristas son positivas, y por lo tanto $last_R(k-1) < first_R(k) \quad \forall k \in \{2, \ldots, s\}$.
Vamos a demostrar tres lemas auxiliares sobre rutas que resultarán útiles para las demostraciones posteriores.

Lema 3.1 *Dada una ruta $R = (w_1, t_1)(w_2, t_2) \cdots (w_s, t_s)$ podemos obtener expresiones explícitas para las funciones $first_R()$ y $last_R()$ mediante las siguientes igualdades $\forall k \in \{1, \ldots, s\}$:*

$$first_R(k) = \sum_{i=2}^{k} d(w_{i-1}, w_i) + \sum_{i=1}^{k-1} t_i$$

$$last_R(k) = \sum_{i=2}^{k} d(w_{i-1}, w_i) + \sum_{i=1}^{k} t_i$$

Demostración: Procedemos por inducción sobre k.

- El caso base $k = 1$ se cumple, ya que

$$first_R(1) = \sum_{i=2}^{1} d(w_{i-1}, w_i) + \sum_{i=1}^{0} t_i = 0 + 0 = 0$$

$$last_R(1) = \sum_{i=2}^{1} d(w_{i-1}, w_i) + \sum_{i=1}^{1} t_i = 0 + t_1 = first_R(1) + t_1$$

- A continuación demostramos el paso inductivo, asumiendo que la hipótesis es cierta para k y demostrándola para $k + 1$.

$$first_R(k+1) \stackrel{\text{def}}{=} d(w_k, w_{k+1}) + last_R(k) =$$

$$\stackrel{\text{h.i.}}{=} d(w_k, w_{k+1}) + \sum_{i=2}^{k} d(w_{i-1}, w_i) + \sum_{i=1}^{k} t_i =$$

$$= \sum_{i=2}^{k+1} d(w_{i-1}, w_i) + \sum_{i=1}^{k} t_i$$

Y consecuentemente para la otra fórmula

$$last_R(k+1) \stackrel{\text{def}}{=} first_R(k+1) + t_{k+1} =$$

$$= \sum_{i=2}^{k+1} d(w_{i-1}, w_i) + \sum_{i=1}^{k} t_i + t_{k+1} =$$

$$= \sum_{i=2}^{k+1} d(w_{i-1}, w_i) + \sum_{i=1}^{k+1} t_i \qquad \square$$

Lema 3.2 *Dada una ruta* $R = (w_1, t_1)(w_2, t_2) \cdots (w_s, t_s)$ *y dos índices* $1 \leq k_1 < k_2 \leq s$ *se cumple la desigualdad*

$$first_R(k_2) - last_R(k_1) \geq \overline{d}(w_{k_1}, w_{k_2})$$

Es decir, el tiempo que un actor tarda en llegar de un vértice a otro no puede ser menor que la distancia mínima entre dichos vértices.

Demostración: La demostración se basa en las igualdades del lema anterior

$$first_R(k_2) - last_R(k_1) =$$

$$= \sum_{i=2}^{k_2} d(w_{i-1}, w_i) - \sum_{i=2}^{k_1} d(w_{i-1}, w_i) + \sum_{i=1}^{k_2-1} t_i - \sum_{i=1}^{k_1} t_i =$$

$$= \sum_{i=k_1+1}^{k_2} d(w_{i-1}, w_i) + \sum_{i=k_1+1}^{k_2-1} t_i \geq$$

$$\geq \sum_{i=k_1+1}^{k_2} d(w_{i-1}, w_i) \geq \overline{d}(w_{k_1}, w_{k_2})$$

siendo la primera desigualdad cierta ya que cada t_i es no negativo, y la segunda ya que $w_{k_1} w_{k_1+1} \cdots w_{k_2}$ es un camino entre w_{k_1} y w_{k_2}, por lo tanto su distancia es mayor o igual que la mínima. □

Lema 3.3 *Sean* $G = (V, E)$ *un grafo dirigido y valorado,* $u_1, \ldots, u_s \in V$ *vértices del grafo y* $a_1, \ldots, a_s \in \mathbb{N}$ *tiempos tales que*

$$\overline{d}(u_k, u_{k+1}) \leq a_{k+1} - a_k \quad \forall k \in \{1, \ldots, s-1\}$$

Entonces existe una ruta R *que pasa por cada vértice* u_k *a la hora* a_k*, es decir,* $W_R(a_k) = u_k$ $\forall k \in \{1, \ldots, s\}$.

Demostración: Vamos a construir explícitamente dicha ruta en G. Para cada par de vértices consecutivos u_k y u_{k+1}, puesto que $\overline{d}(u_k, u_{k+1}) < \infty$, existirá un camino en G que conecte los vértices u_k y u_{k+1} con distancia mínima. A dicho camino le vamos a denotar como una secuencia de términos con subíndices consecutivos, partiendo de $p_1 = 1$, de la siguiente forma:

$$w_{p_k}\ w_{p_k+1}\ w_{p_k+2} \cdots\ w_{p_{k+1}} \qquad \text{donde } w_{p_k} = u_k \text{ y } w_{p_{k+1}} = u_{k+1}$$

y definimos además los tiempos de espera en dichos vértices de la siguiente manera:

$$t_{p_k+1} = t_{p_k+2} = \cdots = t_{p_{k+1}-1} = 0$$

$$t_{p_{k+1}} = a_{k+1} - a_k - \overline{d}(u_k, u_{k+1})$$

La elección de estos tiempos se debe a que el camino tarda $\overline{d}(u_k, u_{k+1})$ en ir de un vértice a otro, pero nosotros queremos que tarde $a_{k+1} - a_k$. Por lo tanto hacemos que la persona espere el tiempo correspondiente en el último vértice del camino.

Nos queda por definir el valor de t_1, para el cuál tomaremos $t_1 = a_1$. Tras estas definiciones, podemos definir la ruta R como sigue:

$$R = (w_1, t_1)(w_2, t_2)\cdots(w_{p_s}, t_{p_s})$$

A continuación comprobaremos que $last_R(p_k) = a_k$, con lo que quedaría demostrado que $W_R(a_k) = w_{p_k} = u_k$, ya que $a_k \in [first_R(p_k),\ last_R(p_k)]$. Hacemos inducción sobre k:

- Caso base $k = 1$:
  Está definido por separado explícitamente que $t_1 = a_1$, y por lo tanto

  $$last_R(p_1) = last_R(1) = first_R(1) + t_1 = 0 + t_1 = a_1$$

- Paso inductivo. Suponemos que es cierto para k y lo comprobamos para $k+1$. Aplicando el lema 3.1:

  $$last_R(p_{k+1}) - last_R(p_k) =$$
  $$= \sum_{r=2}^{p_{k+1}} d(w_{r-1}, w_r) + \sum_{r=1}^{p_{k+1}} t_r - \sum_{r=2}^{p_k} d(w_{r-1}, w_r) - \sum_{r=1}^{p_k} t_r =$$
  $$= \sum_{r=p_k+1}^{p_{k+1}} d(w_{r-1}, w_r) + \sum_{r=p_k+1}^{p_{k+1}} t_r =$$
  $$= \overline{d}(w_{p_k}, w_{p_{k+1}}) + 0 + t_{p_{k+1}} =$$
  $$= \overline{d}(w_{p_k}, w_{p_{k+1}}) + a_{k+1} - a_k - \overline{d}(u_k, u_{k+1}) =$$
  $$= a_{k+1} - a_k$$

  Habiendo obtenido por lo tanto que

  $$last_R(p_{k+1}) - last_R(p_k) = a_{k+1} - a_k$$

  Como por hipótesis de inducción se tiene que $last_R(p_k) = a_k$, concluimos que $last_R(p_{k+1}) = a_{k+1}$ como deseábamos. □

A continuación definimos los dos conjuntos de personas que forman parte de una instancia del problema. Por una parte, los actores del problema, que se moverán libremente de vértice a vértice en el grafo siguiendo una ruta; por otra, los testigos, encargados de decir afirmaciones espacio-temporales sobre estos actores. Los dos grupos no son necesariamente disjuntos, puede haber actores que sean a la vez testigos.

Definición 3.2 *Denotaremos por $\mathcal{T} = \{P_1, \ldots, P_r\}$ el conjunto de testigos del problema, y por $\mathcal{A} = \{A_1, \ldots, A_m\}$ el conjunto de actores, tales que la intersección entre $\mathcal{T}$ y $\mathcal{A}$ puede no ser vacía.*

A continuación vamos a definir la sintaxis y semántica de los testimonios del problema, para lo que debemos considerar que los testigos podrían no acordarse con exactitud de todos los detalles.
Por ello, por un lado, debemos permitir declaraciones que no especifiquen la persona a la que vieron, como por ejemplo "Vi a **alguien** en v a las 7" o "Vi a A_1 **o a** A_2 en v a las 4". Debemos también permitir testimonios que no especifiquen el lugar en el que sucedieron, como "Vi a A a las 3 **(en algún lugar)**", o "Vi a A **en** v_1 **o en** v_2 a las 3".
Finalmente, y más importante, añadir la posibilidad de dar un margen de tiempo: "Vi a A **(en algún momento)**", "Vi a A en v **entre las 4 y las 8**", indicando que dicha persona pasó por ese vértice en algún momento entre las 4 y las 8.
También tenemos que admitir las negaciones de estos testimonios, como por ejemplo "A no pasó por v a las 6", "Estuve en v de 3 a 5 y nadie más pasó por allí".

Definición 3.3 *Dados conjuntos $\mathcal{T}$ y $\mathcal{A}$ de testigos y actores, y un testigo $P \in \mathcal{T}$, diremos que $P : T$ es un testimonio dicho por el testigo P, donde $T = (S, L, I)$ tal que:*

$S \subseteq \mathcal{A}$ *es el conjunto de actores de los habla el testigo.*

$L \subseteq V$, $L \neq \varnothing$ *es el conjunto de vértices donde pudo estar alguno de los actores.*

$I = [a, b] := \{a, a + 1, \ldots, b\} \subset \mathbb{N}, a \leq b$, *representa un intervalo de tiempo.*

Finalmente, dado un testimonio $P : (S, L, I)$, diremos que $P : \neg(S, L, I)$ es también un testimonio, al que nos referiremos como testimonio negado.

Para dar una definición sobre la semántica de un testimonio, entenderemos que un testimonio es válido, si alguno de los actores en S estuvo en alguno de los lugares de L en algún momento entre las a y las b (incluidas). La negación de un testimonio querrá decir que ninguna de las personas de S estuvo en ninguno de los lugares de L en ningún momento durante el intervalo I, lo cual equivale a la negación lógica de lo anterior.

Sin embargo, en el caso de que el testigo sea también un actor, también deberemos comprobar que los testimonios que él mismo ha dicho cuadran con su propia ruta. En el caso de testimonios no negados, exigiremos que uno de los actores de S haya coincidido con P en alguno de los vértices de L a la misma hora.

Para definir el testimonio negado dicho por un testigo que es actor, asumiremos que si alguien nos dice que "nadie ha estado en u o en v durante el intervalo $[a, b]$", es porque él se encontraba en alguno de los lugares mencionados durante todo el intervalo de tiempo, sin

recordar en cuál de los dos, y que nadie más estuvo en ese lugar durante todo $[a, b]$. En este caso no nos servirá la negación lógica del significado del testimonio.

Por ejemplo, si un testigo P que también es actor dice "A estuvo en v entre las 6 y las 8", lo que representamos como $P : (\{A\}, \{v\}, [6, 8])$, significa que ambos P y A coincidieron a la misma hora en v, o bien a las 6 o a las 7 o a las 8. Tomar la negación lógica de esto significa que no coincidieron, permitiendo así una solución en la que ambos visiten dicho vértice en momentos diferentes del intervalo, por ejemplo si P llega a las 6, se va inmediatamente, y A llega a las 8. Sin embargo, no es este el significado que queremos darle a nuestros testimonios negados, sino el siguiente: Si P dice "A no pasó por v de 6 a 8", lo cuál expresaremos como $P : \neg(\{A\}, \{v\}, [6, 8])$, significa que el propio testigo P estuvo en v de 6 a 8, y no vió a A pasar por allí en ningún momento en ese intervalo.

Definición 3.4 *Sean $G = (V, E)$ un grafo dirigido y valorado, conjuntos de testigos y actores $\mathcal{T}$, $\mathcal{A} = \{A_1, \ldots, A_m\}$, $R_1, \ldots, R_m$ un conjunto de rutas de los actores en el grafo, y $P : T$ un testimonio. Decimos que $P : T$ es cierto para $R_1, \ldots, R_m$ si:*

- *Si $P \notin \mathcal{A}$, $T = (S, L, I)$:* $\quad \exists A_i \in S,\ \exists u \in L,\ y\ \exists t \in I$ *tal que* $W_{R_i}(t) = u$

  Es decir, si alguno de los actores de S estuvo en alguno de los vértices del testimonio en algún momento del intervalo I.

- *Si $P \notin \mathcal{A}$, $T = \neg(S, L, I)$:* $\quad \forall A_i \in S,\ \forall u \in L,\ y\ \forall t \in I$ *se cumple que* $W_{R_i}(t) \neq u$

  Es decir, si ninguno de los actores de S estuvo en niguno de los vértices del testimonio durante el intervalo I.

- *Si $P = A_k$, $T = (S, L, I)$:* $\quad \exists A_i \in S,\ \exists u \in L,\ y\ \exists t \in I$ *tal que* $W_{R_i}(t) = u$ *y* $W_{R_k}(t) = u$

  Es decir, si alguno de los actores de S coincidió con A_k en alguno de los vértices del testimonio en algún momento del intervalo I. Nótese que si $S = \{P\}$, este tipo de testimonios permiten a un actor testificar que él mismo estuvo en un determinado lugar.

- *Si $P = A_k$, $T = \neg(S, L, I)$:* $\quad \exists u \in L$ *tal que* $\forall A_i \in S,\ \ y\ \forall t \in I$ *se tiene* $W_{R_i}(t) \neq u$ *y* $W_{R_k}(t) = u$

  Es decir, si el actor A_k estuvo en uno de los vértices mencionados durante todo el intervalo I y ninguno de los actores de S pasó por allí en ese tiempo. Si $S = \varnothing$, estos testimonios permiten a un actor expresar que estuvo en un lugar durante todo el periodo de tiempo $[a, b]$.

Ejemplo 3.1

Vemos a continuación algunos ejemplos de testimonios con su significado en lenguaje natural.

No vi a nadie en v a las 7	$\neg(\mathcal{A}, \{v\}, [7,7])$
A estuvo en $v1$ o en $v2$ a las 3	$(\{A\}, \{v_1, v_2\}, [3,3])$
Vi a A_1 o A_2 en v entre las 4 y las 8	$(\{A_1, A_2\}, \{v\}, [4,8])$
A no pasó por v entre las 4 y las 8	$\neg(\{A\}, \{v\}, [4,8])$
A estuvo en un lugar distinto de v a las 4	$(\{A\}, V - \{v\}, [4,4])$
Yo (A) pasé por v entre las 4 y las 8	$A : (\{A\}, \{v\}, [4,8])$
Yo (A) estuve en v de 4 a 8	$A : \neg(\varnothing, \{v\}, [4,8])$

En los últimos dos ejemplos vemos como, aunque los testimonios están definidos para situaciones más generales, permiten que un actor que es además testigo indique su propia ruta.

El objetivo del problema consistirá en determinar el máximo número de testigos que pueden estar diciendo la verdad simultáneamente, y para ello deben existir una ruta sobre el grafo para cada actor, de tal manera que verifiquen lo dicho en los testimonios. Estamos en condiciones de definir el problema *WPO*.

Definición 3.5 *Una instancia* $\mathtt{x}$ *del problema de los testigos WPO (Optimization Witness Problem) consiste en un grafo* $G = (V, E)$*, unos conjuntos de actores* $\mathcal{A} = \{A_1, \ldots, A_m\}$ *y testigos* $\mathcal{T} = \{P_1, \ldots, P_r\}$*, y un conjunto de testimonios* $\Delta \subseteq \{(P : T) \mid P \in \mathcal{T}, P : T \text{ es un testimonio }\}$*.*

Una solución factible $\mathtt{y} \in \mathtt{Sol(x)}$ *de una instancia del problema con* m *actores* $\mathcal{A} = \{A_1, \ldots, A_m\}$ *es un conjunto de* m *rutas* $\mathtt{y} = R_1, \ldots, R_m$*, una por cada actor del problema.*

La función a maximizar por el problema es el número de testimonios que dicen la verdad:

$$\mathtt{m(x,y)} = \left|\{P \in \mathcal{T} \mid \ \forall (P:T) \in \Delta,\ P:T \text{ es cierto para } \mathtt{y}\}\right|$$

El problema de decisión de los testigos, WP, queda definido de la siguiente forma: Dada una instancia $\mathtt{x}$ *de WPO, determinar si existe una solución* $\mathtt{y} \in \mathtt{Sol(x)}$ *que haga ciertos a todos los testimonios, es decir, de tal forma que* $\mathtt{m(x,y)} = |\mathcal{T}|$*.*

WP-K es el problema de decisión asociado al problema de optimización WPO, definido como: Dada una instancia de WPO y un número $K \in \mathbb{N}$, determinar si al menos K testigos están diciendo la verdad simultáneamente, es decir, si existe $\mathtt{y} \in \mathtt{Sol}(\mathtt{x})$ tal que $\mathtt{m}(\mathtt{x}, \mathtt{y}) \geq K$.

WPO(1) es la versión del problema en la que sólamente hay un actor.

rest-WPO(1) es la versión restringida del problema de optimización, en la que solo hay un actor que no es testigo, cada testigo dice un único testimonio, y no admite testimonios negados. La condición de que solo haya un testimonio por testigo puede ser interpretada como que el problema consiste en maximizar el número de testimonios ciertos, en lugar del número de testigos fiables.

Notación 2 *En ciertas ocasiones, conocido el conjunto de actores de un problema $\mathcal{A} = \{A_1, \ldots, A_m\}$, y sus correspondientes rutas $R_1, \ldots, R_m$, con el objetivo de simplificar las demostraciones de algún teorema, utilizaremos la siguiente notación para indicar la ubicación del actor A_i en el tiempo t.*

$$W(A_i, t) = W_{R_i}(t)$$

Esto nos será útil cuando el símbolo utilizado para referirnos a un actor $A \in \mathcal{A}$ no lleve asociado el subíndice i que se corresponde con su ruta. Así $W(A, t)$ nos permitirá hacer afirmaciones sobre donde estuvo A a cierta hora.

Capítulo 4

Complejidad del Problema de Decisión *WP*

A lo largo de este capítulo vamos a estudiar diferentes versiones del problema *WP*, el problema de decisión en el que nuestro objetivo es determinar si todos los testimonios pueden ser ciertos a la vez. Analizaremos bajo qué restricciones el problema puede ser resuelto en tiempo polinómico, y cuáles son las características del problema suficientes para que éste sea *NP*-completo.

Comenzamos la sección probando la pertenencia del problema *WP* a la clase *NP*, quedando también demostradas, debido a la similaridad en el razonamiento, la pertenencia de *WP-K* también a *NP*, y la pertenencia del problema de optimización *WPO* a la clase *NPO*.

4.1 Pertenencia a *NP*

Comenzamos demostrarendo que los problemas de decisión definidos *WP* y *WP-K* pertenecen a la clase *NP*, así como que *WPO* se trata de un problema *NPO*. A diferencia de lo que sucede en muchos otros problemas, esto no se puede demostrar de manera trivial en nuestro caso, debido a que el tamaño de las soluciones factibles de una instancia (conjuntos de rutas) no está acotado superiormente, por lo que existen soluciones de longitud exponencial en el tamaño de la instancia, que tendremos que probar innecesarias para determinar si una instancia es cierta o no, es decir, que el problema se puede resolver observando solo soluciones de tamaño polinómico. Remitiéndonos a la definición de pertenencia a *NP*, debemos demostrar que para cada instancia `x` para la que exista una solución, se puede encontrar una solución `y` acotada por un cierto polinomio en el tamaño de la instancia. El siguiente lema nos proporciona la cota polinómica que utilizaremos.

Notación 3 *Dada una ruta* $R = (w_1, t_1) \cdots (w_s, t_s)$ *definimos su tamaño como* $|R| = s$*, su longitud como el tiempo que tarda en ser recorrida:* $time(R) = last_R(s)$*, y su final como el último vértice visitado* $final(R) = w_s$*.*

Lema 4.1 *Sea* $\mathtt{x}$ *una instancia de WPO, formada por un grafo* $G = (V, E)$*, actores* $\mathcal{A} = \{A_1, \ldots, A_m\}$*, testigos* $\mathcal{T} = \{P_1, \ldots, P_r\}$ *y testimonios* $\Delta = \{T_1, \ldots, T_N\}$*, y sea* $\mathtt{y} = R_1, \ldots, R_m$ *una solución formada por las rutas de todos los actores, que hace ciertos a todos los testimonios del conjunto* $\Sigma \subseteq \Delta$*. Sea un actor* A_i *cuya ruta es* R_i*, entonces existe una ruta* R_i' *que cumple las siguientes condiciones:*

(i) Si cambiamos la ruta R_i *por* R_i' *en la solución, tomando la solución alternativa* $y' = R_1, \ldots, R_{i-1}, R_i', R_{i+1}, \ldots, R_m$*, se siguen cumpliendo todos los testimonios de* Σ*.*

(ii) El tamaño de la nueva ruta está acotado polinómicamente por $|R_i'| \leq |\Sigma||V| + 1$*.*

(iii) Ambas rutas terminan en el mismo vértice y a la misma hora, es decir, $final(R_i) = final(R_i')$ *y* $time(R_i) = time(R_i')$*.*

Demostración: Sea $R_i = (w_1, t_1) \cdots (w_s, t_s)$, vamos a proceder por inducción fuerte sobre el número de testimonios $|\Sigma|$.

Caso base: Si no hay testimonios que cumplir ($|\Sigma| = 0$), entonces se puede sustituir por la ruta que estuvo en el último vértice durante todo el tiempo:

$$R_i' = (w_s, time(R_i))$$

Claramente esta ruta cumple las tres condiciones del enunciado del teorema:

(i) El primer punto es trivial al no haber testimonios que cumplir.

(ii) $|R_i'| = 1 \leq 0 \cdot |V| + 1$

(iii) Ambas rutas terminan en w_s a la hora $time(R_i)$.

Paso inductivo: Suponemos que para cualquier número de testimonios menor que $|\Sigma|$ el resultado se cumple, y probamos que también se cumple para $|\Sigma| \geq 1$ testimonios. Distinguimos en los siguientes casos:

(a) La ruta R_i no repite vértices, es decir: $\forall j, k \in \{1, \ldots, s\}$ con $j \neq k$ se tiene $w_j \neq w_k$.

En este caso, el tamaño de la ruta es necesariamente menor o igual que el número de vértices en el grafo, por lo tanto basta con tomar $R_i' = R_i$ para que se cumpla la desigualdad $|R_i'| \leq |V| < |\Sigma||V| + 1$.

(b) La ruta repite algún vértice y denotamos con los índices $k \neq j$ los últimos vértices repetidos:

$$k = max\,\{q \in \{1, \ldots, s\} \mid \exists j > q \text{ tal que } w_q = w_j\}$$

$$j \text{ es el único índice } j > k \text{ tal que } w_k = w_j.$$

El objetivo de nuestra demostración será eliminar las repeticiones de vértices en la medida de lo posible, por lo que analizamos en qué casos no se pueden eliminar:

(b1) Existe un testimonio $\overline{T}$ que hace que la ruta tenga que repetir vértice, es decir, que si el actor A_i se quedara esperando en w_k en lugar de visitar los vértices $w_{k+1} \dots w_{j-1}$ dicho testimonio no sería cierto.

En este caso, tomamos la ruta truncada en k, $Q = (w_1, t_1) \cdots (w_k, t_k)$, y consideramos el subconjunto $\Sigma' \subseteq \Sigma - \{\overline{T}\}$ formado por los testimonios que son ciertos al sustituir en $\mathtt{y}$ la ruta R_i por Q.

Como el número de testimonios en Σ' es estrictamente menor que los de Σ, podemos aplicar la hipótesis de inducción sobre él y garantizar que existe una ruta alternativa $Q' = (u_1, h_1) \cdots (u_{s'}, h_{s'})$ para el actor A que cumple las tres condiciones del enunciado del lema para los testimonios de Σ'. Tomemos la ruta

$$R'_i = (u_1, h_1) \cdots (u_{s'}, h_{s'})(w_{k+1}, t_{k+1}) \cdots (w_s, t_s)$$

y comprobemos que cumple los tres requisitos pedidos para Σ.

(i) Vamos a probar que todos los testimonios de Σ siguen siendo ciertos con la nueva ruta. Observamos que podemos expresar la función $W_{R'_i}$ en función de las otras rutas:

$$W_{R'_i}(t) = \begin{cases} W_{Q'}(t) & \text{si} \quad t \leq time(Q) \\ \\ W_{R_i}(t) & \text{si} \quad t > time(Q) \end{cases}$$

1. Sea un testimonio no negado $(P : T) \in \Sigma$, tal que $T = (S, L, [a, b])$ y $P \neq A$. Si $T \in \Sigma'$ entonces la solución modificada al sustituir R_i por Q' cumple el testimonio, por lo que existe un actor $B \in S$ que visita algún vértice de L durante $[a, b]$. Si el actor es distinto de A, trivialmente el testimonio sigue siendo cierto y si $B = A$ entonces existe $t \in [a, b]$ tal que $W_{Q'}(t) \in L$. Además esta t debe ser menor o igual que $time(Q')$, porque si no la función $W_{Q'}$ no estaría definida. Por lo tanto $W_{R'_i}(t) = W_{Q'}(t) \in L$ y el testimonio T se verifica.

   Si $T \notin \Sigma'$, entonces, la ruta truncada Q no verifica el testimonio, pero sin embargo la ruta original R_i si que lo hace. Por lo tanto el actor que cumplía ese testimonio debe ser A, así que existe $t \in [a, b]$ tal que $W_{R_i}(t) \in L$, y este t debe ser mayor que $time(Q)$ (si fuese menor, la ruta Q también verificaría el testimonio). Por lo tanto $W_{R'_i}(t) = W_{R_i}(t) \in L$.

2. Sea un testimonio negado $(P : T) \in \Sigma$, tal que $T = \neg(S, L, [a, b])$ y $P \neq A$. Si $A \notin S$ el testimonio sigue siendo cierto trivialmente, así que asumimos que $A \in S$. Suponemos que el actor A no verifica el testimonio en la nueva ruta R'_i y llegaremos a una contradicción: Existe $t \in [a, b]$ tal que $W_{R'_i}(t) \in L$.

   Si $t \leq time(Q)$, entonces $W_{Q'}(t) \in L$, por lo que Q' no cumple el testimonio, así que Q tampoco lo hace (Q' cumple todos los testimonios que cumple Q). Existe por lo tanto t_0 tal que $W_Q(t_0) \in L$, y como esta ruta coincide con los primeros vértices de R_i, se cumple $W_{R_i}(t_0) \in L$, llegando a que la ruta original R_i estaría incumpliendo uno de los testimonios y obteniendo así la contradicción.

   En caso de que $t > time(Q)$, tenemos directamente que $W_{R_i}(t) = W_{R'_i}(t) \in L$, obteniendo la misma contradicción de antes.

3. Además, si el actor es un testigo, también debemos comprobar que se sigan cumpliendo los testimonios que él haya dicho, $(A : T) \in \Sigma$. Si el testimonio no es negado, $T = (S, L, [a, b])$, la comprobación de que el testimonio sigue siendo cierto es análoga a la hecha en (1).

   Si se trata de un testimonio negado, $T = \neg(S, L, [a, b])$, entonces equivale a que exista un vértice $u \in L$ tal que A se encuentre en u durante todo el intervalo $[a, b]$, y ningún otro actor de S haya pasado por u en ese tiempo. Nótese que no puede darse el caso en que $a \leq time(Q) < b$, ya que $W_{R_i}(time(Q)) = w_i$ y al instante de tiempo siguiente la ruta abandona dicho vértice: $W_{R_i}(time(Q) + 1) \neq w_i$. Por lo tanto tenemos de nuevo 2 casos:

   - Si $a \leq b \leq time(Q)$, entonces la ruta Q verifica el testimonio T, por lo tanto Q' también lo verifica, así que existe $u \in L$ tal que $\forall t \in [a, b] \quad W_{R'_i}(t) = W_{Q'}(t) = u$, y $\forall A \in S$ se tiene $W(A, t) \neq u$, resultando en que T también es cierto en la ruta R'_i.

   - Si $time(Q) < a \leq b$, entonces como la ruta R_i verifica el testimonio, $\forall t \in [a, b]$ t es mayor que $time(Q)$ y por tanto $W_{R'_i}(t) = W_{R_i}(t) = u$, además $\forall A \in S$ se tiene $W(A, t) \neq u$.

(ii) Comprobamos que el tamaño de la ruta R'_i cumple la cota exigida. Como entre los vértices $w_{k+1}, \ldots, w_s$ no hay repetidos, tenemos que $|(w_{k+1}, t_{k+1}) \cdots (w_s, t_s)| \leq |V|$, y por tanto

$$|R'| \leq |Q'| + |V| \overset{\text{h.i.}}{\leq} \big((|\Sigma| - 1)|V| + 1\big) + |V| = |\Sigma||V| + 1$$

(iii) Ambas rutas acaban en el vértice w_s, además, utilizando que por hipótesis de inducción Q y Q' acaban a la misma hora y $w_k = final(Q) = final(Q') = u_{s'}$, también obtenemos que acaban a la misma hora:

$$time(R'_i) = h_1 + \sum_{q=2}^{s'} \big(d(u_{q-1}, u_q) + h_q\big) + \sum_{q=i+1}^{s} \big(d(w_{q-1}, w_q) + t_q\big) =$$

$$= time(Q') + \sum_{q=k+1}^{s} \big(d(w_{q-1}, w_q) + t_q\big) =$$

$$\overset{\text{h.i.}}{=} time(Q) + \sum_{q=k+1}^{s} \big(d(w_{q-1}, w_q) + t_q\big) =$$

$$= t_1 + \sum_{q=2}^{k} \big(d(w_{q-1}, w_q) + t_q\big) + \sum_{q=k+1}^{s} \big(d(w_{q-1}, w_q) + t_q\big) = time(R_i)$$

(b2) El último caso es que existan vértices w_k y w_j repetidos, y al sustituir la ruta R_i por la nueva ruta $R'_i = (w_1, t_1) \cdots (w_{k-1}, t_{k-1})(w_k, last_{R_i}(j) - first_{R_i}(k))(w_{j+1}, t_{j+1}) \cdots (w_s, t_s)$, que espera el tiempo correspondiente en el vértice repetido $w_k = w_j$ en lugar de recorrer $w_{k+1}, \ldots, w_{j-1}$, se sigan cumpliendo todos los testimonios. De esta forma conseguimos una ruta de tamaño menor, así que podemos repetir este proceso con la ruta R'_i y eventualmente, al ir iterando, llegaremos al caso (a) o (b1). □

Por lo tanto, dada una instancia $\mathtt{x}$ de *WPO*, y una solución $\mathtt{y} = R_1, \ldots, R_m$ formada por las rutas de todos los actores, que hace ciertos a todos los testimonios del conjunto $\Sigma \subseteq \Delta$, podemos encontrar una solución del problema $\mathtt{y}'$ aplicando el lema anterior repetidas veces, una sobre cada actor, de tal forma que el tamaño de la solución sea menor o igual que $m(N|V| + 1)$, donde $m = |\mathcal{A}|$ y $N = |\Delta|$.

A continuación, comentamos cómo se puede calcular en tiempo polinómico el valor de la función $\mathtt{m}(\mathtt{x}, \mathtt{y})$, que dadas una instancia del problema $\mathtt{x}$ y una solución factible $\mathtt{y}$ como conjunto de rutas de los actores, evalúa el número de testigos que dicen la verdad. Para ello determinamos para cada testimonio por separado si es o no es cierto conocidas las rutas, y el resultado será la cantidad de testigos tales que todos sus testimonios son ciertos.

Analizamos individualmente cada uno de los tipos de testimonio. No se pretende dar el metodo más eficiente para hacerlo, sino mostrar una forma sencilla de hacerlo en tiempo polinómico.

- Dado un testimonio $T = (S, L, [a, b])$, que ha sido dicho por un testigo que no forma parte de los actores, podemos comprobar si es cierto o no en tiempo polinómico con el siguiente código, que simula la ruta de cada actor que aparece en el testimonio para comprobar si cumple las condiciones exigidas:

```
fun verificar(T = (S, L, [a, b]))
    for A_i ∈ S :
        Si la ruta del A_i es  R_i = (w_1, t_1)···(w_s, t_s)
        for k = 1,...,s :
            if [first(k), last(k)]∩[a, b]≠∅ and w_k ∈ L :
                return true
    return false
```

- Un testimonio negado $T = \neg(S, L, [a, b])$ dicho por un testigo que no es actor es cierto si y solo si no lo es $(S, L, [a, b])$, podemos reducirlo por tanto al caso anterior.

- Un testimonio no negado dicho por un actor $A : T$, $T = (S, L, [a, b])$, se puede comprobar viendo por separado para cada actor de S si coinciden, de la siguiente forma:

```
fun verificar(A : (S, L, [a, b]))
    for A_i ∈ S :
        Si la ruta de A_i es  R_i = (w_1, t_1)···(w_s, t_s)
        y la ruta de A es  R = (u_1, h_1)···(u_s, h_p)
        for k = 1,...,s :
            for j = 1,...,p :
                if [first(k), last(k)]∩[first(j), last(j)]∩[a, b]≠∅
                    and w_k ∈ L :
                    return true
    return false
```

- Finalmente, un testimonio negado dicho por un actor $A : T$, $T = \neg(S, L, [a, b])$, es cierto si el actor A estuvo en algún vértice $u \in L$ durante todo el intervalo $[a, b]$ y ningún actor de S pasó por ese mismo vértice durante todo el intervalo, siendo esto último equivalente al testimonio $\neg(S, \{u\}, [a, b])$.

```
fun verificar(A : ¬(S, L, [a, b]))
    Si la ruta del A es  R = (w_1, t_1)···(w_s, t_s)
    for k = 1,...,s :
        if [a, b]⊆[first(k), last(k)] and w_k ∈ L :
            return true and not verificar(¬(S, {w_k}, [a, b]))
    return false
```

Con todo esto, estamos en condiciones de ver que los problemas que hemos definido pertenecen correspondientemente a las clases *NP* y *NPO*.

Teorema 2 *Los problemas de decisión WP y WP-K pertenecen a la clase NP. El problema WPO de optimización es un problema NPO.*

Demostración: El problema *WP*, donde nos preguntan si todos los testimonios son ciertos, es sólo un caso particular del problema *WP-K*, donde la pregunta es si hay más de K testigos diciendo la verdad simultáneamente. Nos sirve por lo tanto ver que este segundo problema está en *NP*.

Tenemos que comprobar que se cumplen los puntos (1) y (2) de la definición de pertenencia a *NP*. Por el lema 4.1 demostrado previamente, tenemos que se cumple la condición del punto (1), que exige la existencia de una solución `y` de tamaño polinómico respecto al tamaño de la instancia. Hemos visto además que la veracidad o falsedad de una solución se puede computar en tiempo polinómico, lo que prueba también el punto (2) de la definición, demostrando que el problema pertenece a *NP*.

Para demostrar que *WPO* es un problema *NPO*, comprobamos que se cumplen los cuatro puntos de la definición de *NPO*. Los puntos (1) y (2) se cumplen trivialmente, porque reconocer una instancia y una solución se puede hacer claramente en tiempo polinómico. El lema 4.1 de esta sección garantiza la existencia de soluciones que se pueden obtener en tiempo polinómico (3), y los comentarios sobre cómo calcular la función `m` en tiempo polinómico prueban el punto (4). □

4.2 Análisis de la Complejidad de Distintas Versiones

A continuación, estudiamos con detalle una serie de versiones restringidas y simplificadas del problema, decidiendo para cada una de ellas si es *NP*-completa o si se puede resolver en tiempo polinómico, con el objetivo de determinar cuáles son los elementos de la definición del problema que provocan la intratabilidad. Tomaremos como punto de partida una versión muy restringida del problema, fácil de resolver en tiempo polinómico, que iremos poco a poco relajando al eliminar ciertas restricciones, y realizando las modificaciones necesarias sobre el algoritmo.

Comenzamos estudiando versiones del problema en las que el conjunto de actores y testigos son disjuntos. Por lo tanto realizamos una primera simplificación en la notación, ya que debemos notar que, a efectos de decidir si todos los testimonios son compatibles o no, es irrelevante quién sea el testigo. Es por esto que en lugar de referirnos a los testimonios con la notación definida, $P : T$, evitaremos nombrar al testigo que lo dijo y llamaremos testimonio simplemente a una terna $T = (S, L, I)$, puesto que es irrelevante quién sea el testigo.

Nótese que esto mismo no se puede hacer en el problema de optimización, aunque también sean disjuntos los conjuntos $\mathcal{A}$ y $\mathcal{T}$, puesto que en ese caso buscamos a testigos que siempre digan la verdad, así que quién haya dicho cada testimonio sí que es influyente en la solución. Tampoco se podrá hacer cuando los conjuntos $\mathcal{A}$ y $\mathcal{T}$ tengan elementos en común, ya que debemos recordar que, si el testigo es un actor, la definición de veracidad de un testimonio es distinta.

4.2.1 Un solo actor, lugar y hora concretos

En la primera versión que consideraremos sólo hay un actor, digamos $\mathcal{A} = \{A\}$, que además no forma parte de los testigos del problema, $A \notin \mathcal{T}$. Tenemos una serie de testimonios, $T_1, \ldots, T_N$, donde $T_j = (S_j, L_j, I_j)$ con $j \in \{1, \ldots, N\}$ de tal forma que $S_j = \{A\}$, $L_j = \{u_j\}$, e $I_j = [a_j, a_j]$. Es decir, solo admitimos testimonios que concreten persona, lugar y hora.

Vamos a suponer que los testimonios están ordenados cronológicamente, de tal forma que $a_1 \leq a_2 \leq \cdots \leq a_N$. En caso de no cumplirse esta propiedad, se pueden ordenar en tiempo polinómico para cumplirla.

Lo único que necesitamos comprobar para saber si todos los testimonios son ciertos es que se pueda llegar al vértice u_j a la hora a_j saliendo desde el vértice anterior u_{j-1} a las a_{j-1}, $\forall j = 2, ..., N$. Para ello podemos recurrir al algoritmo de Dijkstra, que calcula el camino más corto desde un vértice a todos los demás, o al algoritmo de Floyd-Warshall que calcula los caminos más cortos entre todo par de vértices. Presentamos el pseudocódigo que lo resuelve, de manera muy sencilla, ya que nos servirá como estructura base sobre la que realizaremos cambios en las siguientes versiones del problema. En el código asumimos que existe una función `dist` que dados dos vértices obtiene la distancia de su camino más corto.

```
for j = 2,...,N
    if dist(u_{j-1},u_j) > a_j − a_{j-1}
        return false
return true
```

En caso de haber utilizado el algoritmo de Floyd previamente para calcular todas las distancias, se obtiene un orden de complejidad de $O(|V|^3 + N)$, donde N es el número de testimonios. Si optáramos por utilizar el algoritmo de Dijkstra cada vez que se llama a la función `dist`, el orden sería de $O(N|E|log|V|)$, tratándose en cualquiera de los dos casos de algoritmos polinómicos.

4.2.2 Un actor, intervalos de tiempo arbitrarios

Ahora consideramos la variante del problema en la que mantenemos todas las restricciones antes citadas, salvo la restricción sobre los intervalos, es decir, permitimos que los testigos den intervalos de tiempo de cualquier tamaño $I_j = [a_j, b_j]$. Demostramos a continuación que, pese a todas las restricciones exigidas, la libertad a la hora de dar el intervalo de tiempo en los testimonios es suficiente para que el problema sea NP-completo, mediante una reducción desde el problema del camino hamiltoniano.

Definición 4.1 *Sea $G = (V, E)$ un grafo no dirigido, decimos que un camino $w_1 w_2 \ldots w_s$ es hamiltoniano si visita cada vértice del grafo exactamente una vez. Esto equivale a cumplir las siguientes dos condiciones:*

- $V = \{w_1, \ldots, w_s\}$
- $|V| = s$

El problema de determinar si un grafo contiene un camino hamiltoniano es *NP*-completo [7].

Teorema 3 *Sea $\mathcal{A} = \{A\}$, el problema de determinar si todos los testimonios $T_1, \ldots, T_N$ de la forma $T_j = (A, u_j, [a_j, b_j])$ $\forall j \in \{1, \ldots, N\}$ sobre un grafo $G = (V, E)$ son compatibles o no, es NP-duro.*

Demostración: Construiremos una reducción polinómica desde el problema del camino hamiltoniano.

Sea $G = (V, E)$ una instancia del problema del camino hamiltoniano, consideramos la instancia de WP con las siguientes características:

- Un sólo actor, $\mathcal{A} = \{A\}$.
- El mismo grafo G, asociando a cada arista distancia 1.

$$d(u, v) = 1 \quad \forall (u, v) \in E$$

- Si $V = \{v_1, \ldots, v_n\}$ consideramos los siguientes n testimonios:

$$T_j = (A, v_j, [0, n-1]) \quad \forall j \in \{1, \ldots, n\}$$

Al querer hacer ciertos todos los testimonios, estamos exigiendo al actor A que visite todos los vértices del grafo en un tiempo donde solo podrá visitar n vértices, por lo que será imposible que repita ninguno de ellos.
Comprobamos por doble implicación que G tiene un camino hamiltoniano si y solo si todos los testimonios pueden ser ciertos simultáneamente.

$\Rightarrow$ Primero suponemos que existe un camino hamiltoniano $w_1 w_2 \cdots w_n$.

Entonces basta con tomar la ruta $R = (w_1, 0)(w_2, 0) \cdots (w_n, 0)$, y utilizando el lema 3.1 sobre rutas tenemos que $\forall k \in \{1, \ldots, n\}$

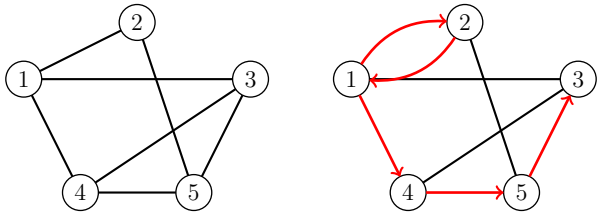
$$first_R(k) = \sum_{i=2}^{k} d(w_{i-1}, w_i) + \sum_{i=1}^{k-1} t_i = \sum_{i=2}^{k} 1 + \sum_{i=1}^{k-1} 0 = k - 1$$

$$last_R(k) = \sum_{i=2}^{k} d(w_{i-1}, w_i) + \sum_{i=1}^{k} t_i = k - 1$$

Por lo tanto $\forall k \in \{1, \ldots, n\}$ $W_R(k-1) = w_k$.

Tomando un testimonio cualquiera $T_j = (A, v_j, [0, n-1])$, como el camino es hamiltoniano, visita el vértice v_j, entonces existe $k \in \{1, \ldots, n\}$ tal que $w_k = v_j$. Tenemos entonces que $W_R(k-1) = w_k = v_j$, $k-1 \in [0, n-1]$ y por lo tanto el testimonio es cierto. Como esto es válido para cualquier testimonio, todos los testimonios pueden ser ciertos simultáneamente.

$\Leftarrow$ Supongamos ahora que existe una ruta $(w_1, t_1)(w_2, t_2)\cdots(w_s, t_s)$ que hace ciertos a todos los testimonios. En principio la ruta podría no corresponderse con un camino hamiltoniano, porque una solución de nuestro problema puede pasar varias veces por el mismo vértice. Nuestro trabajo será comprobar que, gracias a las restricciones temporales, se garantiza que de la ruta podemos extraer un camino hamiltoniano. En la siguiente imagen, se muestran un grafo y una ruta sobre él, que no es ciclo hamiltoniano. Lo que demostraremos a continuación es que rutas de este tipo no hacen ciertos a todos los testimonios, ya que no llegan a tiempo al último.



Por una parte, para que todos los testimonios sean ciertos, la ruta debe pasar por todos los vértices, por lo que $s \geq n$. Por otra, recurriendo de nuevo al lema 3.1 sobre rutas

$$first_R(k) = k - 1 + \sum_{i=0}^{k-1} t_i \geq k - 1 \quad \text{y} \quad last_R(k) = k - 1 + \sum_{i=0}^{k} t_i \geq k - 1$$

Con esto tenemos que en concreto el vértice w_n de la ruta es alcanzado a una hora mayor o igual que $n-1$, siendo el último vértice de la ruta alcanzable dentro del rango $[0, n-1]$ como exigen los testimonios. Es decir, los vértices $w_{n+1}, w_{n+2}, \ldots$, si existieran, no sirven para verificar nigún testimonio, por lo tanto la ruta tiene que haber visitado todos los n vértices del grafo al llegar a w_n.

Es decir, $\{v_1, \ldots, v_n\} = \{w_1, \ldots, w_n\}$, lo que prueba que $w_1 \cdots w_n$ es un camino hamiltoniano. $\square$

4.2.3 Un actor, intervalos de tiempo disjuntos

En esta tercera versión buscamos una postura intermedia entre las dos anteriores, permitiendo intervalos de la forma $I_j = [a_j, b_j]$, pero exigiendo una condición para poder garantizar que el problema puede ser resuelto en tiempo polinómico. Dicha condición es que todos los

intervalos sean disjuntos dos a dos. Mantenemos de momento las restricciones previas, es decir, $\mathcal{A} = \{A\}$, y los testimonios $T_j = (S_j, L_j, I_j)$ cumplen $S_j = \{A\}$, $L_j = \{u_j\}$. Vamos a explicar cómo puede ser resuelta esta variante en tiempo polinómico.

Suponemos, de nuevo, que los testimonios vienen dados en orden cronológico:

$$a_1 \leq b_1 < a_2 \leq b_2 < \cdots < a_r \leq b_r$$

La solución es similar a la versión inicial. En este caso tenemos que comprobar que se puede llegar al vértice u_j antes de b_j, saliendo desde u_{j-1} a la hora $h_{j-1} \in [a_{j-1}, b_{j-1}]$ a la que hayamos llegado a este. La mejor hora de llegada a u_j es $h_{j-1} + \bar{d}(u_{j-1}, u_j)$, y se puede calcular con Floyd-Warshall o Dijkstra. Sin embargo, podría darse el caso en el que la mejor hora para llegar a un vértice ocurra antes de a_j. Entonces, si saliéramos a por el siguiente vértice inmediatamente, no se verificaría el testimonio que exigía estar ahí en $[a_j, b_j]$.
La solución que tomaremos consiste en hacer que el actor "pierda el tiempo" en cualquier vértice del camino hasta que sean las a_j. Por lo tanto, para calcular h_j tomamos el máximo entre la hora de llegada y a_j.

```
h_1 = a_1
for j = 2,...,N :

    h_j = max(h_{j-1} + dist(u_{j-1},u_j), a_j)

    if h_j > b_j :
        return false

return true
```

El coste computacional del algoritmo es igual al de la versión 4.2.1.

4.2.4 Un actor, testimonios con varios lugares

Consideramos una nueva variante que conserva todas las restricciones anteriores, salvo la de $|L_j| = 1$, permitiendo que los testigos no concreten el lugar en el que vieron al actor, sino que nos den un conjunto de lugares $L_j = \{u_1^j, \ldots, u_{n_j}^j\}$ donde este pudo haber estado. Mantenemos $S_j = \{A\}$ y $I_j = [a_j, b_j]$, siendo los intervalos disjuntos dos a dos, y asumimos como siempre que los testimonios están en orden cronológico: $a_1 \leq b_1 < a_2 \leq b_2 < \cdots < a_r \leq b_r$.

Para nuestra solución tenemos que adentrarnos ligeramente en el funcionamiento del algoritmo de Dijkstra, que resumimos brevemente:
Este algoritmo toma como entrada un grafo y un vértice `v` del mismo. Como dijimos antes, calculará la mínima distancia en el grafo desde dicho vértice a todos los demás.
Para ello, usa un array `dist` en el que marca la mejor forma de llegar a cada vértice. Este array comienza con el valor ∞ en todos los vértices, menos en `v`, donde pone el valor 0, indicando el tiempo que tardaríamos en llegar al vértice `v` (en el que ya nos encontramos).

También crea otro array `visitados` en el que indica los vértices para los cuales ya conocemos el mejor camino. Inicialmente, todos los valores son `false` salvo para el vértice `v`, que es `true` (indicando que ya hemos llegado a él).
En cada paso del algoritmo, calcula el vértice `u` al que llegamos en el menor tiempo de entre los vértices que aún no han sido visitados. Marca `u` como visitado y analiza todos sus vértices adyacentes, actualizando `dist` si viajando desde `u` se consigue un camino mejor que el anteriormente conocido.
De esta forma consigue el mejor camino de `v` a todos los vértices del grafo, por lo tanto podremos calcular con una sola ejecución el mejor camino a todos los vértices $u_1^j, \ldots, u_{n_j}^j$.

Modificaremos la fase inicial del algoritmo, de tal forma que el origen no sea un sólo vértice, sino que sean varios (todos los vértices entre $u_1^{j-1}, \ldots, u_{n_{j-1}}^{j-1}$ a los que hallamos podido llegar a tiempo en el paso anterior). Para ello, indicamos que en la tabla `visitados` el valor de todos estos vértices es inicialmente `true`, y en la tabla `dist` marcamos la mejor forma de llegar a ellos, que recordemos de la variante anterior que se calculaba como el máximo entre la hora de llegada y a_j.

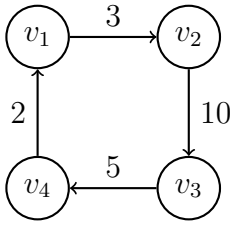
```
dist = {∞,...,∞}
//Personalizamos la tabla inicial, para partir desde todos los vertices
for i = 1,...,n_1 :
    dist[u_i^1] = a_1
    visited[u_i^1] = true

for j = 2,...,N :
    dijkstra(dist, visited) //Usando la tabla y cola ya creadas
    dist_aux = {∞,...,∞}
    //Personalizamos la siguiente tabla inicial, para partir desde todos
   los vertices, cada uno desde cuando fue alcanzado.
    visited = {false,...,false}
    for i = 1,...,n_j :
        if dist[u_i^j] <= b_j : //Marcamos los vertices alcanzados a tiempo
            dist_aux[u_i^j] = max(dist[u_i^j],a_j)
            visited[u_i^j] = true
    if visited == {false,...,false} :
        //Si no hemos alcanzado ningun vertice, no hay solucion
        return false
    dist = dist_aux
return true
```

Ejemplo 4.1

Supongamos que tenemos los siguientes testimonios sobre el grafo de la imagen:

$$(A, \{v_1, v_3\}, [0, 0]) \quad (A, \{v_2, v_4\}, [3, 4]) \quad (A, \{v_1, v_3\}, [5, 7])$$



El algoritmo comienza desde los vértices v_1 y v_3, y calcula el camino más corto desde ellos a los demás.
Consigue llegar a tiempo a v_2, pero no a v_4. Para la siguiente iteración toma como punto de partida solamente el vértice v_2, y por lo tanto no consigue alcanzar v_1 ni v_3. Aunque desde v_4 sí que se puede llegar a v_1, el algoritmo hace bien en descartarlo, ya que esa ruta no cumple los tres testimonios.

4.2.5 Un actor, con testimonios negados y con varios lugares

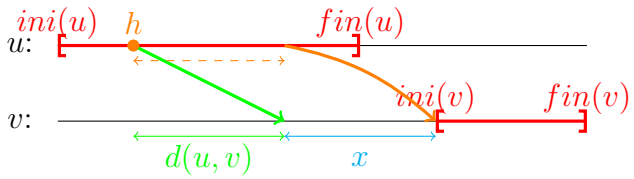
La siguiente versión, que resolveremos también en tiempo polinómico, será igual que la anterior pero añadiendo las negaciones de testimonios.
Separaremos los testimonios en función del tipo que sean, teniendo los testimonios no negados $T_1, \ldots, T_{N_1}$ de la forma $T_j = (\{A\}, \{u_1^j, \ldots, u_{n_j}^j\}, [a_j, b_j])$ con $j \in \{1, \ldots, N_1\}$, y los testimonios negados $T_{N_1+1}, \ldots, T_{N_2}$ donde $T_j = \neg(\{A\}, \{u_1^j, \ldots, u_{n_j}^j\}, [a_j, b_j])$ con $j \in \{N_1 + 1, \ldots, N_2\}$. Estos testimonios negados equivalen, en lenguaje natural, a decir que A no estuvo en ninguno de los lugares $u_1^j, \ldots, u_{n_j}^j$ en ningún momento entre a_j y b_j.

Primero vamos a realizar una modificación en el algoritmo anterior: cada vértice u debe llevar una hora de apertura $ini(u)$ y de cierre $fin(u)$. No se puede estar en dicho vértice ni antes de su apertura, ni después de su cierre. El algoritmo de Dijkstra se puede modificar para cumplir este requisito sin problemas:

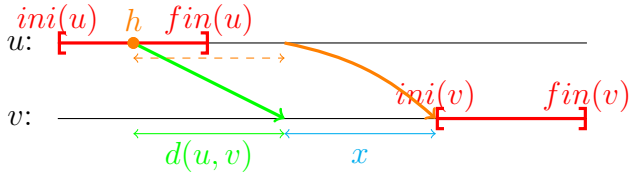
Si estamos en el vértice u a la hora h, y miramos la arista $(u, v) \in E$, debemos comprobar que la hora de llegada al vértice v sea válida.

- Si $h + d(u, v) > fin(v)$, entonces el vértice v no se puede alcanzar desde u y no cambiamos nada.
- Si $ini(v) \leq h + d(u, v) \leq fin(v)$ podemos alcanzar el vértice v a hora $h + d(u, v)$.

- Si $h + d(u,v) \leq ini(v)$, hay dos posibilidades, como ilustran los dibujos a continuación:
  - Podemos esperar en u un tiempo y, tras este, partir hacia v.
  - No podemos esperar el tiempo requerido porque se cierra la ventana de tiempo en la que se puede estar en u.



En la figura de arriba observamos como el tiempo que hay que esperar para llegar al vértice v a tiempo es la distancia entre $ini(v)$ y $h + d(u,v)$, es decir, $x = ini(v) - (h + d(u,v))$. Para poder viajar de u a v, necesitamos que se cumpla la siguiente condición: $h + x \leq fin(u)$, equivalente a $ini(v) - d(u,v) \leq fin(u)$.



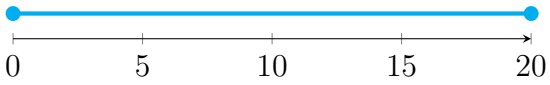
En esta segunda imagen vemos la necesidad de que se cumpla la condición previa. Como en este caso $fin(u)$ es menor, no podemos quedarnos el tiempo suficiente esperando en el vértice u.

Mostramos a continuación como queda el pseudocódigo del algoritmo de Dijkstra con las modificaciones comentadas. En la práctica se implementaría con una cola de prioridad.

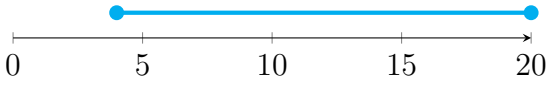
```
1  void dijkstra(src, dist, visited) :
2    while ∃u = V tal que  visited[u] = false and dist[u] < ∞:
3        Tomamos  u ∈ V  con las condiciones anteriores que minimice  dist[u]
4        visited[u] = false
5      for  v ∈ V  tal que  (u, v) ∈ E  :
6              h = dist[u] // la mejor hora en la que se puede llegar a u
7              if  ini(v) ≤  h  +  d(u, v) ≤ fin(v): // Caso 1
8                   h' = h + d(u, v)
9              else if  h  +  d(u, v)  <  fin(v)  and  ini(v) − d(u, v) ≤ fin(u): // Caso 2
10                  h' = ini(v)
11             // Actualizamos la tabla si mejora la hora
12             if (not visitado[v]) and h' < dist[v] :
13                  dist[v] = h'
```

Este cambio no soluciona todavía el problema con las negaciones, pero a continuación explicaremos como podemos usarlo a nuestro favor.
Dada una instancia del problema con testimonios negados y no negados, vamos a transformarla en otra instancia, que sólo conserva las restricciones no negadas, pero recoge las condiciones exigidas por las negaciones mediante un nuevo grafo modificado, cuyos vértices tendrán hora de apertura y de cierre. Esta nueva instancia podrá ser resuelta con el algoritmo que solucionaba la variante anterior utilizando el código de Dijkstra que acabamos de ver.

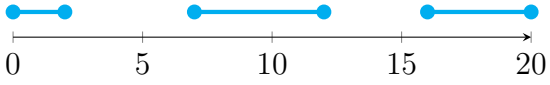
Nuestro objetivo será asignar a cada vértice una hora de apertura y otra de cierre, en función de los instantes de tiempo en los que pueden ser visitados acorde a las negaciones. En un vértice del que no se habla en ninguna negación podremos estar a cualquier hora, digamos entonces que $ini(v) = 0, fin(v) = \infty$:



Si tuviésemos negaciones que nos impiden visitar el vértice en intervalos de tiempo solapados con el principio o el final, también lo podemos hacer. Por ejemplo, un testimonio que nos impide estar de 0 a 3, daría como resultado $ini(v) = 4$.



El problema aparece cuando los instantes de tiempo en los que el actor puede visitar un determinado vértice no constituyen un intervalo, sino que queda dividido en varios intervalos disjuntos. En este caso no podemos asignar unos valores adecuados para $ini(v)$ y $fin(v)$. Por ejemplo, si un testigo nos dice que no estuvo en este vértice de 3 a 6, y otro que no estuvo de 13 a 15, se nos queda el siguiente diagrama:



Tras procesar todos los testimonios negados, habremos obtenido para cada vértice v_i del grafo una lista de intervalos $J_1^i, \ldots, J_{k_i}^i$ (asociados al vértice) indicando las horas en las que se puede visitar. En nuestro ejemplo serían $[0, 2], [7, 12], [16, \infty)$. A continuación damos una definición formal de estos subintervalos:

Definición 4.2 *Dado un conjunto $X \subset \mathbb{N}$ finito definimos su* ***representación en intervalos*** *como el conjunto de intervalos en los numeros naturales $J_1, \ldots, J_k$ tales que:*

$$X = J_1 \cup \ldots \cup J_k$$

donde cada J_p, $p \in \{1, \ldots, k\}$ es de la forma $J_p = [x_p, y_p]$, $x_p \leq y_p$, y es maximal en X, es decir, no se puede ampliar ni por la izquierda ni por la derecha:

$$x_p - 1 \notin X \ , \ y_p + 1 \notin X$$

Definimos adicionalmente las siguientes funciones sobre dichos intervalos:

$$ini(J_p) = x_p \quad y \quad fin(J_p) = y_p$$

Entonces, para definir los intervalos $J_1^i, \ldots, J_{k_i}^i$ del vértice v_i, partiremos del conjunto $X_i = [0, \infty)$ representando todos los instantes de tiempo en los que A pudo haber visitado v_i.
Si existe algún testimonio negado $T_j = \neg(\{A\}, \{u_1^j, \ldots, u_{n_j}^j\}, [a_j, b_j])$ que indique que v_i no fue visitado, es decir, $v_i \in \{u_1^j, \ldots, u_{n_j}^j\}$, entonces restaremos a X_i el intervalo $[a_j, b_j]$.

Finalmente tomaremos la representación en intervalos del conjunto X_i resultante. Como un conjunto tiene que ser finito para que exista su representación en intervalos, supondremos que $M \in \mathbb{N}$ es un número lo suficientemente grande, en concreto $M \geq b_j \ \forall j = 1, \ldots, N_1$, y definimos $J_1^i, \ldots, J_{k_i}^i$ del vértice v_i formalmente como la representación de intervalos de

$$X_i = [0, M] - \bigcup \left\{ [a_j, b_j] \mid j \in \{N_1 + 1, \ldots, N_2\},\ v_i \in \{u_1^j, \ldots, u_{n_j}^j\} \right\}$$

Construimos un nuevo grafo cuyos vértices son estos intervalos, de la siguiente forma:

$$G' = (V', E')$$

$$V' = \{J_p^i \mid i \in \{1, \ldots, n\},\ p \in \{1, \ldots, k_i\}\}$$

$$E' = \{(J_{p_1}^{i_1}, J_{p_2}^{i_2}) \mid (v_{i_1}, v_{i_2}) \in E, p_1 \in \{1, \ldots, k_{i_1}\}, p_2 \in \{1, \ldots, k_{i_2}\}\}$$

$$d(J_{p_1}^{i_1}, J_{p_2}^{i_2}) = d(v_{i_1}, v_{i_2})$$

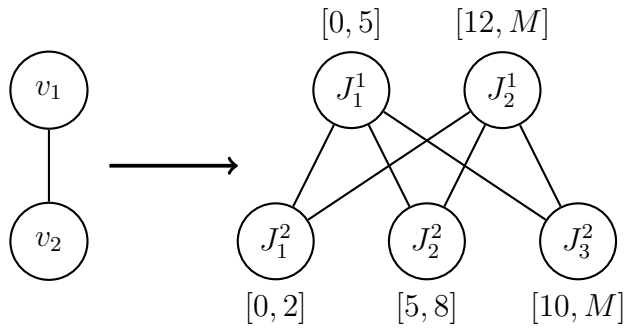
Es decir, estamos replicando cada vértice del grafo original, para dividirlo en nuevos vértices con hora de apertura y de cierre.

Ejemplo 4.2

Si tenemos los siguientes testimonios:

$$\neg(\{A\}, v_1, [6, 11]) \quad \neg(\{A\}, v_2, [3, 4]) \quad \neg(\{A\}, v_2, [9, 9])$$

A raíz del grafo de la izquierda, obtendríamos el siguiente grafo:

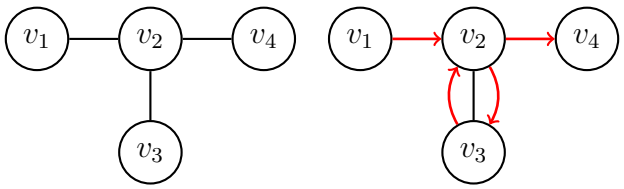


Además al replicar vértices de esta forma admitimos la posibilidad de que en una sola ejecución de Dijkstra se pueda pasar más de una vez por un mismo vértice del grafo G, siempre que se pase en distintos intervalos de tiempo, ya que se corresponderán con vértices distintos del grafo G'. Esto es de gran importancia, como ilustra el siguiente ejemplo.

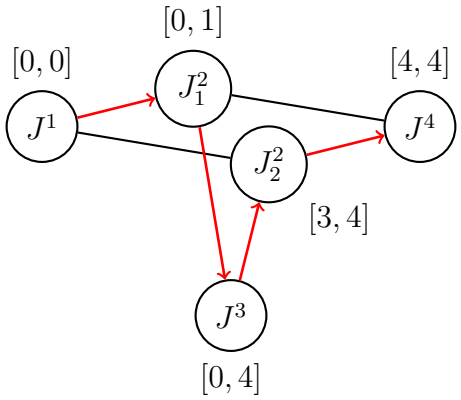
Ejemplo 4.3

Supongamos que tenemos los siguientes testimonios sobre el grafo de la imagen, donde todas las aristas tienen distancia 1.

$$(\{A\}, v_1, [0, 0]) \quad (\{A\}, v_4, [0, 4]) \quad \neg(\{A\}, v_1, [1, 4]) \quad \neg(\{A\}, v_2, [2, 2]) \quad \neg(\{A\}, v_4, [0, 3])$$



La imagen de la derecha muestra el recorrido que debe hacer A para verificar los 5 testimonios. Comienza en v_1 a la hora 0, y debe trasladarse a v_2 para verificar la primera negación. Inmediatamente debe salir de allí hacia v_3 para no incumplir la segunda negación. Finalmente volverá a v_2 por ser el único camino hacia v_4. La siguiente imagen muestra como quedaría el grafo G' para este mismo problema



Al haber duplicado el vértice v_2, el camino en el nuevo grafo no repite vértice, por lo tanto se puede obtener mediante el algoritmo de Dijkstra.

Finalmente, nos falta adaptar los testimonios no negados $T_1, \ldots, T_{N_1}$ de G al nuevo grafo G'. Visitar un vértice v_i en G será equivalente a visitar alguno de los vértices $J_1^i, \ldots, J_{k_i}^i$, por lo tanto, para cada testimonio $T_j = (\{A\}, \{u_1^j, \ldots, u_{n_j}^j\}, [a_j, b_j])$ definimos el correspondiente testimonio sobre el nuevo grafo:

$$T'_j = \left(\{A\}, \left\{J_1^i, \ldots, J_{k_i}^i \mid v_i \in \{u_1^j, \ldots, u_{n_j}^j\}\right\}, [a_j, b_j]\right)$$

Con todo esto, hemos transformado una instancia del problema con testimonios negados a una instancia del problema sin negaciones pero con horas de apertura y cierre en cada vértice, que ya sabemos resolver en tiempo polinómico empleando el algoritmo explicado anteriormente. El siguiente resultado garantiza que este método funciona.

Teorema 4 *Existe una ruta R en $G = (V, E)$ que satisface todos los testimonios $T_1, \ldots, T_{N_2}$, si y solo si existe una ruta R' en G' que satisface todos los testimonios $T'_1, \ldots, T'_{N_1}$ y además cumple las restricciones temporales, es decir, si $W_{R'}(t) = J_p^i$ entonces $ini(J_p^i) \leq t \leq fin(J_p^i)$.*

Demostración: Procedemos por doble implicación.

$\Rightarrow$ Primero suponemos que existe una ruta $R = (w_1, t_1) \cdots (w_s, t_s)$ en G que verifica todos los testimonios $T_1, \ldots, T_{N_2}$.
Construimos la ruta R' correspondiente sobre el grafo G', $R' = (w'_1, t_1) \cdots (w'_s, t_s)$. Los tiempos de espera $t_1, \ldots, t_s$ serán los mismos que en la ruta original, y cada vértice w'_k, $k = 1, \ldots, s$ queda definido como sigue:

$$\text{Si } w_k = v_i \quad \text{entonces } w'_k = J_p^i \quad \text{con } p \text{ tal que} \quad first_R(k) \in J_p^i$$

- **La ruta está bien definida:**
  Tenemos que demostrar la existencia de algún valor de $p \in \{1, \ldots, k_i\}$ que cumpla la condición de que $first_R(k) \in J_p^i$.
  En caso de existir tal valor de p es claro que es único, porque los distintos intervalos $J_1^i, \ldots, J_{k_i}^i$ son disjuntos dos a dos.

  Supongamos que no existiera dicho p, y que por lo tanto $first_R(k) \notin J_p^i \ \forall p = 1, \ldots, k_i$. Entonces, por la definición de los intervalos J_p^i:

  $$first_R(k) \notin J_1^i \cup \cdots \cup J_{k_i}^i = X_i$$

  $$first_R(k) \notin [0, M] - \bigcup \left\{ [a_j, b_j] \mid j \in \{N_1 + 1, \ldots, N_2\},\ v_i \in \{u_1^j, \ldots, u_{n_j}^j\} \right\}$$

  Como para M lo suficientemente grande podemos asumir que $first_R(k) \in [0, M]$, entonces

  $$first_R(k) \in \bigcup \left\{ [a_j, b_j] \mid j \in \{N_1 + 1, \ldots, N_2\},\ v_i \in \{u_1^j, \ldots, u_{n_j}^j\} \right\}$$

  así que existe j comprendido entre $N_1 + 1$ y N_2 tal que

  $$first_R(k) \in [a_j, b_j] \quad \text{y} \quad v_i \in \{u_1^j, \ldots, u_{n_j}^j\}$$

  Entrando en contradicción con el testimonio $T_j = \neg(\{A\}, \{u_1^j, \ldots, u_{n_j}^j\}, [a_j, b_j])$, que dice precisamente que el actor no estuvo en dicho vértice a esa hora.

  Además, para que la ruta esté bien definida, necesitamos ver que los vértices que aparecen consecutivos en ella sean aristas del grafo G'. Esto es claro a partir de la definición de E', ya que si $(w_k, w_{k+1}) \in E$, entonces $(w'_k, w'_{k+1}) \in E'$.

  Con esto tenemos la ruta R' bien definida, y como los tiempos y las distancias entre vértices coinciden, se tiene

  $$first_R(k) = first_{R'}(k) \quad \text{y} \quad last_R(k) = last_{R'}(k)$$

  por lo tanto

  $$\text{si } W_R(t) = w_k \quad \text{entonces} \quad W_{R'}(t) = w'_k$$

- **Se cumplen las restricciones temporales:**
  Vamos a comprobar que si $W_{R'}(t) = J_p^i$ entonces $ini(J_p^i) \leq t \leq fin(J_p^i)$.

  Sea k tal que $J_p^i = w'_k$ y $t \in [first_{R'}(k), last_{R'}(k)] = [first_R(k), last_R(k)]$ (Nótese que podría haber distintos valores de k tales que $J_p^i = w'_k$, porque nada nos impide

revisitar el mismo vértice, pero sin embargo sólo uno de ellos visita el vértice en el tiempo t).
De la definición de w'_k deducimos que $first_R(k) \in J_p^i = [ini(J_p^i), fin(J_p^i)]$, por lo tanto

$$ini(J_p^i) \leq first_R(k) \leq t$$

Para comprobar la otra desigualdad, $t \leq fin(J_p^i)$, vamos a suponer por reducción al absurdo que $t > fin(J_p^i)$. Entonces

$$first_R(k) \leq fin(J_p^i) + 1 \leq t \leq last_R(k)$$

y por lo tanto $W_R(fin(J_p^i) + 1) = w_k = v_i$. Es decir, a la hora $t' := fin(J_p^i) + 1$, la ruta R estaba en el vértice v_i, pero vamos a comprobar que esto es imposible.

Como J_p^i forma parte de la representación en intervalos del conjunto X_i, deducimos de su definición que $t' = fin(J_p^i) + 1 \notin X_i$, por lo tanto

$$t' \in \bigcup \left\{ [a_j, b_j] \mid j \in \{N_1 + 1, \ldots, N_2\},\ v_i \in \{u_1^j, \ldots, u_{n_j}^j\} \right\}$$

por lo tanto existe un índice $j \in \{N_1 + 1, \ldots, N_2\}$ tal que

$$t' \in [a_j, b_j] \quad \text{y} \quad v_i \in \{u_1^j, \ldots, u_{n_j}^j\}$$

y por lo tanto $W_R(t') = v_i$ entra en contradicción con el testimonio

$$T_j = \neg(\{A\}, \{u_1^j, \ldots, u_{n_j}^j\}, [a_j, b_j])$$

- **Se satisfacen los testimonios** Nos falta comprobar que, como queremos, la ruta creada satisface los testimonios $T'_1, \ldots, T'_{N_1}$.

  Sea T'_j un testimonio cualquiera en este rango. Debido a que el correspondiente T_j es cierto, sabemos que existen un cierto $v_i \in \{u_1^j, \ldots, u_{n_j}^j\}$ y un valor de $t \in [a_j, b_j]$ tales que $W_R(t) = v_i$.
  Sea $k \in \{1, \ldots, s\}$ el índice de la ruta R tal que $v_i = w_k$ y $first_R(k) \leq t \leq last_R(k)$, tenemos que $W_{R'}(t) = w'_k = J_p^i$ para un cierto p. Finalmente, si nos vamos a la definición de los testimonios T' comprobamos que

  $$J_p^i \in L'_j = \left\{ J_1^i, \ldots, J_{k_i}^i \mid v_i \in \{u_1^j, \ldots, u_{n_j}^j\} \right\}$$

  Por lo tanto la ruta R' pasa por uno de los vértices del testimonio T'_j a la hora t exigida por este, así que se verifica el testimonio T'_j.

$\Leftarrow$ Ahora suponemos la existencia de una ruta $R' = (w'_1, t_1) \cdots (w'_s, t_s)$ en G' que satisface las condiciones del enunciado del teorema.
Definimos la ruta $R = (w_1, t_1) \cdots (w_s, t_s)$ en G, donde si $w'_k = J^i_p$ entonces $w_k = v_i$.
Según la definición de E', $(J^{i_1}_{p_1}, J^{i_2}_{p_2}) \in E'$ si y solo si $(v_{i_1}, v_{i_2}) \in E$, por lo tanto la ruta está bien definida, ya que los vértices consecutivos en ella son aristas de G.
Como antes, es claro ver que si $W_{R'}(t) = w'_k$ entonces $W_R(t) = w_k$.

Primero veamos que se cumplen los testimonios sin negaciones $T_1, \ldots, T_{N_1}$. Sea $j \in \{1, \ldots, N_1\}$, dado que el testimonio T'_j se verifica, existen $i \in V$ y $p = 1, \ldots, k_i$ tales que

$$J^i_p \in L'_j = \left\{ J^i_1, \ldots, J^i_{k_i} \mid v_i \in \{u^j_1, \ldots, u^j_{n_j}\} \right\}$$

y existe $t \in [a_j, b_j]$ de tal forma que $W_{R'}(t) = J^i_p$. Sea k tal que $J^i_p = w'_k$, entonces tenemos que $W_R(t) = w_k = v_i$, haciendo cierto el testimonio $T_j = (A, \{u^j_1, \ldots, u^j_{n_j}\}, [a_j, b_j])$.

A continuación vemos que se cumplen los testimonios negados $T_{N_1+1}, \ldots, T_{N_2}$. Supongamos que alguno de ellos, $T_j = \neg(A, \{u^j_1, \ldots, u^j_{n_j}\}, [a_j, b_j])$, no se cumpliera. Esto significa que existe cierto $v_i \in \{u^j_1, \ldots, u^j_{n_j}\}$ y $t \in [a_j, b_j]$ tal que $W_R(t) = v_i$. Por lo tanto

$$t \in \bigcup \left\{ [a_j, b_j] \mid j = N_1 + 1, \ldots, N_2,\ v_i \in \{u^j_1, \ldots, u^j_{n_j}\} \right\}$$

así que $t \notin X_i = J^i_1 \cup \cdots \cup J^i_{k_i}$, de lo que concluimos que t no pertenece a ningún intervalo J^i_p, $p \in \{1, \ldots, k_i\}$.

Por otra parte, como $W_R(t) = v_i$, tenemos que $W_{R'}(t) = J^i_p$ para cierto valor de p, y según la hipótesis, esto implicaría que $ini(J^i_k) \leq t \leq fin(J^i_k)$, es decir, $t \in J^i_p$, lo cual entra en contradicción con lo anterior.
Concluimos por lo tanto que todos los testimonios $T_1, \ldots, T_{N_2}$ son ciertos para la ruta R. $\square$

4.2.6 m actores que no son testigos, los testimonios hablan de un sólo actor

Es fácil adaptar los algoritmos estudiados hasta ahora a las versiones del problema en las que haya más de un actor, $\mathcal{A} = \{A_1, \ldots, A_m\}$, siempre que no haya actores que a la vez sean testigos ($\mathcal{T} \cap \mathcal{A} = \varnothing$), y mantengamos las condiciones de que para cada testimonio no negado $T_j = (S_j, L_j, I_j)$ se cumpla que $|S| = 1$, es decir, que sólo se habla de un actor a la vez, además de que los intervalos de tiempo que aparezcan en los testimonios no negados sobre una misma persona sean disjuntos dos a dos. Es decir, si T_i y T_j son dos testimonios no negados tales que $S_i = S_j = \{\mathrm{A}\}$, entonces $[a_i, b_i]$ y $[a_j, b_j]$ deben ser disjuntos. Los testimonios negados sin embargo si que pueden tener $|S| > 1$, ya que el testimonio $\neg(\{B_1, \ldots, B_k\}, L, I)$ es equivalente a los testimonios $\neg(B_1, L, I), \cdots, \neg(B_k, L, I)$, por lo que no añade complejidad extra al problema.

En este caso, un testimonio para ser cierto o no sólo dependerá de los otros testimonios que hablen del mismo actor, ya que los actores no interfieren entre sí, por lo que si hay n de ellos, bastaría con utilizar n veces el algoritmo anterior, es decir, partimos la instancia a resolver en varias instancias más pequeñas independientes.

4.2.7 m actores, testimonios sobre conjuntos de actores

Recapitulando, hemos visto que añadir libertad a los testimonios en el intervalo horario hace que el problema sea *NP*-completo, y necesitamos ciertas restricciones sobre este para poder resolverlo en tiempo polinómico. Sin embargo, añadir libertad en el campo de los lugares visitados no nos causa este mismo problema. La siguiente pregunta natural que nos planteamos es qué sucederá si añadimos libertades también en el campo de los actores de los que trata un testimonio.

Concretamente, veremos que basta con admitir testimonios de la forma $T = (S_j, \{u_j\}, 0)$ para que el problema sea *NP*-duro, no siendo necesarios ni los testimonios negados ni las libertades en el intervalo de tiempo o lugar para que el problema se encuentre en esta clase de complejidad. Demostraremos una reducción desde el problema clásico *3-SAT*.

Definición 4.3 *Dado un número natural $K \geq 2$, el problema K-SAT queda definido de la siguiente forma: Dado un conjunto de variables lógicas $\{x_1, \ldots, x_m\}$, y N fórmulas lógicas de tamaño K, $\varphi_1, \ldots, \varphi_N$ donde $\varphi_j = y_1^j \vee y_2^j \vee \cdots \vee y_K^j$ con $y_i^j \in \{x_1, \ldots, x_m, \overline{x}_1, \ldots, \overline{x}_m\}$, el problema consiste en decidir si todas las fórmulas pueden ser ciertas simultáneamente.*
Es decir, si existe una asignación lógica $\tau : \{x_1, \ldots, x_m\} \longrightarrow \{0, 1\}$ de tal forma que $\forall j \in \{1, \ldots, m\}$ existe un índice $i \in \{1, \ldots, K\}$ tal que $\tau(y_i^j) = 1$ (entendiendo que $\tau(\overline{x}) = 1$ sii $\tau(x) = 0$ y viceversa).

El problema *K-SAT* es *NP*-completo si K es mayor o igual que 3, sin embargo el problema *2-SAT* se puede resolver en tiempo polinómico.

Teorema 5 *La versión del problema WP donde los testimonios son de la forma $T = (S_j, \{u_j\}, 0)$ es NP-duro.*

Demostración: Vamos a reducir polinómicamente el problema *3-SAT* al problema *WP*. Como el problema *3-SAT* es *NP*-completo, esto sirve para probar la dureza de nuestro problema.

Tomamos una instancia de *3-SAT* sobre las variables $x_1, \ldots, x_m$, formada por las cláusulas $\varphi_1, \ldots, \varphi_N$, donde $\varphi_j = y_1^j \vee y_2^j \vee y_3^j$, con cada $y_k^j \in \{x_1, \ldots, x_m, \overline{x}_1, \ldots, \overline{x}_m\}$.

Vamos a transformarla en una instancia del problema *WP*. Para ello tomamos un grafo formado por dos vértices, $V = \{\alpha, \beta\}$, y ninguna arista $E = \varnothing$. Este grafo tan sencillo nos servirá para modelar las variables $x_1, \ldots, x_m$ como actores en nuestro problema, de tal forma

que las variables que estén en el vértice α tomarán el valor 1, y las que estén en β tomarán el valor 0.

Definimos el conjunto de actores de nuestro problema como $\mathcal{A} = \{x_1, \ldots, x_m, \overline{x}_1, \ldots, \overline{x}_m\}$, y solo tomamos un testigo $\mathcal{T} = \{P\}$, tal que $P \notin \mathcal{A}$. Dicho testigo dice todos los siguientes testimonios:

$$T_i^{\alpha} = (\{x_i, \overline{x}_i\}, \alpha, 0) \quad \forall i = 1, \ldots, m$$
$$T_i^{\beta} = (\{x_i, \overline{x}_i\}, \beta, 0) \quad \forall i = 1, \ldots, m$$
$$T_j = (\{y_1^j, y_2^j, y_3^j\}, \alpha, 0) \quad \forall j = 1, \ldots, N$$

Los primeros dos grupos de testimonios, T^{α} y T^{β}, sirven para garantizar que una variable y su negada están en vértices distintos, mientras que el tercero para asegurar que cada cláusula φ_j es cierta.
Procedemos a comprobar que existe una asignación $\tau : \{x_1, \ldots, x_m\} \longrightarrow \{0, 1\}$ que hace ciertas todas las cláusulas $\varphi_1, \ldots, \varphi_m$ si y sólo si todos los testimonios son compatibles en la instancia de *WP*.

$\Rightarrow$ Suponemos que tenemos una asignación τ que hace ciertas todas las cláusulas. Entonces, definimos las rutas $R_1, \ldots, R_{2m}$ (cada una formada por un sólo vértice) de la siguiente forma:

$$\text{Si } \tau(x_i) = 1 \quad \text{entonces} \quad R_i = (\alpha, 0) \text{ y } R_{m+i} = (\beta, 0)$$
$$\text{Si } \tau(x_i) = 0 \quad \text{entonces} \quad R_i = (\beta, 0) \text{ y } R_{m+i} = (\alpha, 0)$$

donde las rutas $R_1, \ldots, R_m$ se corresponden con los actores $x_1, \ldots, x_m$ y las rutas $R_{m+1}, \ldots, R_{2m}$ con los actores $\overline{x}_1, \ldots, \overline{x}_m$.

Es claro que se satisfacen los testimonios T_i^{α} y T_i^{β} ya que si si una variable se encuentra en un vértice, su negada se encuentra en el otro.
Además, como cada φ_j es cierta, existe $k \in \{1, 2, 3\}$ tal que $\tau(y_k^j) = 1$, y por lo tanto $W_R(y_k^j, 0) = \alpha$, haciendo cierto el testimonio T_j.

$\Leftarrow$ Suponemos que conocemos rutas para todos los actores, $R_1, \ldots, R_{2m}$, que satisfacen todos los testimonios. Definimos la asignación τ de la siguiente forma:

$$\tau(x_i) = \begin{cases} 1 & \text{si } W(x_i, 0) = \alpha \\ \\ 0 & \text{si } W(x_i, 0) = \beta \end{cases} \qquad \forall i \in \{1, \ldots, m\}$$

Tenemos entonces que

$$\tau(x_i) = 1 \;\Leftrightarrow\; W(x_i, 0) = \alpha$$

y gracias a los testimonios T_i^{α} y T_i^{β}, también tenemos

$$\tau(\overline{x}_i) = 1 \;\Leftrightarrow\; \tau(x_i) = 0 \;\Leftrightarrow\; W(x_i, 0) = \beta \;\Leftrightarrow\; W(\overline{x}_i, 0) = \alpha$$

por lo tanto, para cualquier actor $A \in \{x_1, \ldots, x_m, \overline{x}_1, \ldots, \overline{x}_m\}$

$$\tau(A) = 1 \;\Leftrightarrow\; W(A, 0) = \alpha$$

Ahora veamos que $\forall j = 1, \ldots, m$ la cláusula φ_j es cierta. Como el testimonio T_j lo es, existe y_k^j tal que $W_R(y_k^j, 0) = \alpha$. Entonces $\tau(y_j^i) = 1$, y por lo tanto $\varphi_j = y_1^j \vee y_2^j \vee y_3^j$ es cierta. $\square$

4.2.8 m actores, testimonios sobre todos ellos

Aunque ya hemos visto que el problema con varios actores es en general *NP*-completo, vamos a ver como podemos resolver en tiempo polinómico algunas variantes restringidas del mismo. Vamos a tomar los testimonios más sencillos posibles que admiten a varios actores a la vez: "alguien estuvo en v_i a las a_i", es decir, tenemos una lista de testimonios $T_1, \ldots, T_N$, donde $T_j = (\mathcal{A}, u_j, [a_j, a_j])$, que simplificaremos escribiendo $T_j = (\mathcal{A}, u_j, a_j)$, y como siempre suponemos ordenados de menor a mayor valor de a_j.

Definición 4.4 *Sea $G = (V, E)$ un grado dirigido y valorado, y sean los testimonios $T_1, \ldots, T_k$, de tal forma que para cada $T_j = (S_j, L_j, I_j)$ con $j \in \{1, \ldots, k\}$ se cumple que $L_j = \{u_j\}$ y $I_j = \{a_j\}$. Definimos el* ***grafo ordenado*** *de $T_1, \ldots, T_k$ como $\overrightarrow{G} = (\overrightarrow{V}, \overrightarrow{R})$ donde*

$$\overrightarrow{V} = \{T_1, \ldots, T_k\}$$

$$\overrightarrow{E} = \{(T_i, T_j) \mid i \neq j,\ a_i + \overline{d}(u_i, u_j) \leq a_j\}$$

siendo $\overline{d}(u, v)$ la mínima distancia de u a v en G.

En este grafo, existe una arista entre dos testimonios sólo si un mismo actor puede hacer ciertos ambos. Se trata de un grafo acíclico, ya que, suponiendo los testimonios $T_1, \ldots, T_k$ ordenados cronológicamente, si $i > j$, entonces $a_i > a_j$ y por lo tanto $a_i + d(u_i, u_j) > a_j$ y la arista (T_i, T_j) no pertenece a $\overrightarrow{E}$.

Como demostraremos en el siguiente teorema, comprobar la veracidad de todos los testimonios se puede reducir a encontrar un recubrimiento del grafo ordenado con m caminos.

Definición 4.5 *Dado un grafo $G = (V, E)$, un recubrimiento por caminos consiste en un conjunto formado por caminos sobre el grafo, de tal forma que cada vértice $v \in V$ está contenido en uno de esos caminos.*

El problema Minimum Path Cover consiste en determinar el cardinal mínimo de cualquier recubrimiento por caminos del grafo.

Nótese que en general se trata de un problema *NP*-completo, pues determinar si existe un recubrimiento formado por un solo camino equivale a encontrar un camino hamiltoniano en dicho grafo. Sin embargo, en el caso de grafos acíclicos como el nuestro, existen algoritmos que lo resuelven en tiempo polinómico [8]. En particular, conocer el mínimo número de caminos necesarios, nos permite determinar si existe un recubrimiento formado por exactamente m caminos.

Teorema 6 *Sea $G = (V, E)$ un grafo dirigido y valorado, $\mathcal{A} = \{A_1, \ldots, A_m\}$, sean $T_1, \ldots, T_N$ testimonios de la forma $T_j = (\mathcal{A}, u_j, [a_j, a_j])$, y sea $\overrightarrow{G}$ el grafo ordenado de $T_1, \ldots, T_N$.*
Existe un conjunto de m rutas que hacen ciertos a todos los testimonios en G si y solo si existe un recubrimiento del grafo $\overrightarrow{G}$ formado por m caminos.

Demostración: Procedemos por doble implicación.

$\Rightarrow$ Supongamos que conocemos las m rutas $R_1, \ldots, R_m$ que hacen que sean ciertos todos los testimonios $T_1, \ldots, T_N$. Los caminos correspondientes en $\overrightarrow{G}$ que cubrirán el grafo consisten en la secuencia de testimonios que verifica cada actor (recordemos que los vértices son testimonios).

Entonces para todo $j \in \{1, \ldots, N\}$, dado que los testimonios son de la forma $T_j = (\mathcal{A}, u_j, a_j)$, existe al menos un actor, al que nombraremos $B_j \in \mathcal{A}$, que visitó el vértice u_j a la hora a_j, es decir, $W(B_j, a_j) = u_j$.

Y podemos definir, por lo tanto, cada camino C_i $\forall i \in \{1, \ldots, m\}$, correspondiente a la ruta del actor A_i, como el que pasará por los vértices del conjunto $C_i = \{T_j \mid j \in \{1, \ldots, N\},\ B_j = A_i\}$. Debido a la definición de $\overrightarrow{G}$, esto define de forma única el camino, puesto que (T_{j_1}, T_{j_2}) no es una arista del grafo si $j_1 \geq j_2$.

Lo que sí deberemos comprobar es que se trate efectivamente de un camino según la definición de $\overrightarrow{G}$, es decir, que cada arista (T_{j_1}, T_{j_2}) que se encuentre en el camino sea en efecto una arista de $\overrightarrow{E}$.
Si $T_{j_1}, T_{j_2} \in C_i$ son dos testimonios tales que $j_1 < j_2$, entonces tenemos

$$W(A_i, a_{j_1}) = W_{R_i}(a_{j_1}) = u_{j_1} = w^i_{k_1} \quad W(A_i, a_{j_2}) = W_{R_i}(a_{j_2}) = u_{j_2} = w^i_{k_2}$$

para ciertos $k_1 < k_2$, donde $w^i_1, \ldots, w^i_{s_i}$ son los vértices visitados por la ruta R_i.

Utilizando el lema 3.2 sobre rutas demostrado anteriormente, obtenemos lo siguiente

$$first_{R_i}(k_2) - last_{R_i}(k_1) \geq \overline{d}(w^i_{k_1}, w^i_{k_2}) = \overline{d}(u_{j_1}, u_{j_2})$$

Además, de la definición de la función W_{R_i} obtenemos que $a_{j_1} \in [first_{R_i}(k_1), last_{R_i}(k_1)]$ y además $a_{j_2} \in [first_{R_i}(k_2), last_{R_i}(k_2)]$, en particular:

$$a_{j_1} \leq last_{R_i}(k_1) \quad a_{j_2} \geq first_{R_i}(k_2)$$

Lo cual, junto con la anterior ecuación, nos lleva a la desigualdad

$$a_{j_2} - a_{j_1} \geq first_{R_i}(k_2) - last_{R_i}(k_1) \geq \overline{d}(u_{j_1}, u_{j_2})$$

$$a_{j_1} + \overline{d}(u_{j_1}, u_{j_2}) \leq a_{j_2}$$

por lo tanto, $(T_{j_1}, T_{j_2}) \in \overrightarrow{E}$ como queríamos.

Solo nos falta comprobar que estos caminos cubren todos los vértices de $\overrightarrow{V} = \{T_1, \ldots, T_N\}$. Para ello, tomamos un testimonio cualquiera $T_j \in \overrightarrow{V}$, y como dijimos antes, el actor B_j cumple que $W_R(B_j, a_j) = u_j$. Entonces el vértice T_j queda cubierto por el camino C_i asociado a B_j, es decir, con índice $i \in \{1, \ldots, m\}$ tal que $A_i = B_j$.

$\Leftarrow$ Para la otra implicación vamos a suponer que conocemos los caminos $C_i = \overrightarrow{w}^i_1, \ldots, \overrightarrow{w}^i_{s_i}$ para $i \in \{1, \ldots, m\}$ que cubren el grafo $\overrightarrow{G}$.

Para construir las rutas equivalentes en el grafo G utilizamos el lema 3.3 sobre rutas, que nos garantiza la existencia de una ruta dados los vértices por los que pasa y los tiempos en los que pasa por cada uno. Definimos cada ruta R_i como la ruta que pasa por los vértices del camino C_i, $\left\{u_j \mid T_j = \overrightarrow{w}^i_k \text{ para cierto } k \in \{1, \ldots, s_i\}\right\}$ en los tiempos correspondientes $\left\{a_j \mid T_j = \overrightarrow{w}^i_k \text{ para cierto } k \in \{1, \ldots, s_i\}\right\}$.

Tenemos que comprobar que se cumple la hipótesis de dicho teorema, es decir, que si u_{j_1}, u_{j_2} son vértices consecutivos en una misma ruta R_i tales que $j_1 < j_2$ entonces

$$\overline{d}(u_{j_1}, u_{j_2}) \leq a_{j_2} - a_{j_1}$$

Esto es fácil, dado que u_{j_1} y u_{j_2} se corresponden con vértices consecutivos en el camino C_i, por lo tanto $(T_{j_1}, T_{j_2}) \in \overrightarrow{E}$, y por definición $\overline{d}(u_{j_1}, u_{j_2}) + a_{j_1} \leq a_{j_2}$.

Estas rutas $R_1, \ldots, R_m$, gracias al lema usado, cumplen que para todo $i \in \{1, \ldots, m\}$ y $j \in \{1, \ldots, N\}$, si el camino C_i pasa por el testimonio T_j, entonces la ruta del actor pasa por u_j a las a_j, $W_{R_i}(a_j) = u_j$.

Entonces es directo ver que cada testimonio $T_1, \ldots, T_N$ se verifica para estas rutas. Sea T_j con $j \in \{1, \ldots, N\}$ un testimonio, como $T_j \in \overrightarrow{V}$, existe algún camino C_i que pasa por este vértice, es decir, $T_i = \overrightarrow{w}^i_k$ para cierto k. Por lo tanto $W_{R_i}(a_j) = u_j$, así que el actor A_i visitó dicho vértice a las a_j. $\square$

Finalmente nos introducimos superficialmente en el funcionamiento del algoritmo mencionado, para obtener el coste computacional resultante. Los pasos a dar consisten en computar el grafo $\overrightarrow{G}$, y posteriormente aplicar el algoritmo citado. Para la obtención del grafo, necesitamos las distancias entre muchos pares de vértices distintos, por lo tanto según convenga podemos utilizar el algoritmo de Dijkstra, obteniendo un coste de $O\big(min(N, |V|) \cdot |E|log|V|\big)$, o el de Floyd-Warshall, con coste $O(|V|^3)$, para posteriormente generar el grafo comparando cada par de vértices en tiempo $O(N^2)$.

Para el algoritmo que resuelve el recubrimiento por caminos mínimos en tiempo polinómico en grafos acíclicos, tomamos el algoritmo utilizado en [8], que está explicado más detalladamente en la página web [9].
Su funcionamiento consiste en reducir el problema a un problema de encontrar un emparejamiento de $N - m$ aristas en un grafo bipartito. Para esto emplea un teorema de Fulkerson [10], que muestra la equivalencia entre encontrar un recubrimiento de m caminos en G y encontrar un emparejamiento de tamaño $N - m$ en el grafo bipartito generado de la siguiente forma:

$$V_1 = \{(v, 1) \mid v \in \overrightarrow{V}\} \qquad V_2 = \{(v, 2) \mid v \in \overrightarrow{V}\} \qquad V' = V_1 \cup V_2$$

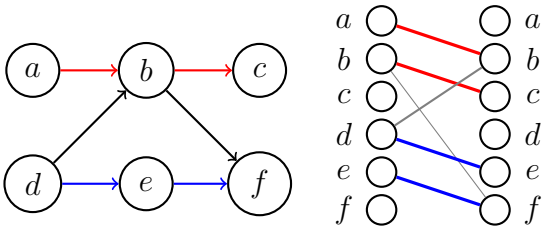
$$E' = \{\big((u, 1), (v, 2)\big) \mid (u, v) \in \overrightarrow{E}\}$$

siendo el coste computacional del algoritmo de emparejamiento de $O\left(\sqrt{|V'|}|E'|\right) =$
$= O\left(\sqrt{N}N^2\right) = O\left(N^{\frac{5}{2}}\right)$

Ejemplo 4.4

A continuación vemos un ejemplo extraido de [8] que ilustra como generar el grafo bipartito a raíz del grafo acíclico, y como se corresponde el emparejamiento máximo encontrado, con el recubrimiento mínimo de caminos.

A la izquierda sale el grafo original, y a continuación el correspondiente grafo bipartito, sobre el cuál calcular un emparejamiento máximo equivale a encontrar caminos que cubren el grafo original.



4.2.9 m actores, testimonios que hablan de 2 actores

Dado que el problema *2-SAT* sí que puede ser resuelto en tiempo polinómico, de hecho en tiempo lineal, si restringimos los testimonios a la forma $T = (\{A^j, B^j\}, u_j, a_j)$, donde cada testimonio sólo menciona a dos actores, no podemos garantizar que el problema sea *NP*-completo. De hecho, vamos a comprobar que no lo es y que se puede resolver en tiempo polinómico, reduciéndolo al propio *2-SAT*.

Definimos las variables lógicas x_j^i, para todo $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, N\}$. De forma intuitiva, si la variable x_j^i toma el valor 1 en una asignación τ de *2-SAT*, se corresponderá en nuestro problema con que el actor A_i verificó el testimonio T_j, es decir, que estuvo en u_j a la hora a_j.

Definiremos dos tipos distintos de cláusulas en el problema *2-SAT*:

- Las cláusulas $\varphi_1, \ldots, \varphi_N$, una por cada testimonio, que garantizarán que o bien el actor A^j o bien B^j hicieron cierto el testimonio.

$$\varphi_j = x_j^p \vee x_j^q \text{ donde } A^j = A_p \text{ y } B^j = A_q$$

- Por otro lado, necesitamos cláusulas que representen que, en ocasiones, si un actor verifica un testimonio, esto impedirá que haga cierto otro testimonio posterior, al no poder llegar a tiempo de un vértice al otro. Para ello definimos $\overrightarrow{G_i} = (\overrightarrow{V_i}, \overrightarrow{E_i})$ como el grafo ordenado de los testimonios $\{T_j \mid A_i \in \{A^j, B^j\},\ j \in \{1, \ldots, N\}\}$ en los que aparece A_i.

  Teniendo esto, definimos la cláusula $\varphi_{j,k}^i = x_j^i \to \neg x_k^i$ para $i \in \{1, \ldots, m\}$ y $j, k \in \{1, \ldots, N\}$ tales que $T_j, T_k \in \overrightarrow{V_i}$ donde $j < k$ y $(T_j, T_k) \notin \overrightarrow{E_i}$. Es decir, dado que dicha arista no se encuentra en el grafo $\overrightarrow{G_i}$, si el actor A_i cumpliera el primer testimonio, no podría cumplir el segundo, lo cual expresamos mediante la condición de *2-SAT* $x_j^i \to \neg x_k^i = \neg x_j^i \vee \neg x_k^i$.

A continuación demostramos el teorema que nos garantiza que aplicar un algoritmo que resuelva la instancia transformada de *2-SAT* nos servirá para encontrar una solución a nuestro problema.

Teorema 7 *Todos los testimonios $T_1, \ldots, T_N$ con forma $T = (\{A^j, B^j\}, u_j, a_j)$ pueden ser ciertos simultáneamente si y solo si existe una asignación*

$$\tau : \left\{x_j^i \mid i \in \{1, \ldots, m\},\ j \in \{1, \ldots, N\}\right\} \longrightarrow \{0, 1\}$$

que satisface todas las cláusulas $\varphi_1, \ldots, \varphi_N$ y las $\varphi_{j,k}^i$ definidas.

Demostración: Vemos las dos implicaciones por separado.

$\Rightarrow$ Suponemos conocidas las rutas $R_1, \ldots, R_m$ que verifican todos los testimonios. Definimos τ de la siguiente forma:

$$\tau(x_j^i) = \begin{cases} 1 & \text{si } W_{R_i}(a_j) = u_j \\ \\ 0 & \text{en caso contrario} \end{cases}$$

es decir, $\tau(x_j^i) = 1$ si el actor A_i cumplió el testimonio T_j.

- Primero vemos que las cláusulas $\varphi_1, \ldots, \varphi_N$ son ciertas. Sea $j \in \{1, \ldots, N\}$, como $T_j = (\{A^j, B^j\}, u_j, a_j)$ es cierto, tenemos la siguiente cadena de equivalencias:

  $$A^j \text{ o } B^j \text{ visitaron el vértice } u_j$$

  $$W_{R_p}(a_j) = u_j \text{ o } W_{R_q}(a_j) = u_j \text{ (donde } A_p = A^j \text{ y } A_q = A^j)$$

  $$\tau(x_j^p) = 1 \text{ o } \tau(x_j^q) = 1$$

  $$\varphi_j = x_j^p \vee x_j^q \text{ es cierta con la asignación } \tau$$

- Veamos que también son ciertas las cláusulas $\varphi_{j,k}^i = \neg x_j^i \vee \neg x_k^i$. Supongamos que $\tau(x_j^i) = 1$ y $\tau(x_k^i) = 1$ simultáneamente y llegaremos a una contradicción.

  Por la definición de las cláusulas de este tipo, sabemos que $(T_j, T_k) \notin \overrightarrow{E_i}$, y por lo tanto es imposible llegar al vértice u_k a tiempo saliendo desde el vértice u_j a las a_j:

  $$a_j + \overline{d}(u_j, u_k) > a_k$$

  Sin embargo, como $\tau(x_j^i) = 1$ y $\tau(x_k^i) = 1$, nuestra ruta si pasaría a tiempo por los dos vértices. La contradicción la obtenemos a partir del lema 3.2:

  Tenemos que $W_{R_i}(a_j) = u_j = w_{p_1}$ y $W_{R_i}(a_k) = u_k = w_{p_2}$ para ciertos $p_1 < p_2$ donde $w_1, \ldots, w_s$ son los vértices por los que pasa la ruta R_i.

  $$a_j \leq last_{R_i}(p_1)$$

  $$a_k \geq first_{R_i}(p_2)$$

  y aplicando el lema 3.2:

  $$a_k - a_j \geq first_{R_i}(p_2) - last_{R_i}(p_1) \geq \overline{d}(w_{p_1}, w_{p_2}) = \overline{d}(u_j, u_k)$$

  Concluimos por lo tanto que las cláusulas $\varphi_{j,k}^i$ también son ciertas con esta asignación.

$\Leftarrow$ Suponemos que existe una asignación τ que satisface todas las cláusulas del problema transformado. Vamos a construir rutas de forma muy similar a la demostración del teorema anterior, recurriendo al lema 3.3. Para ello, comenzamos tomando todos los testimonios que debe cumplir cada actor A_i.

Sean $x^i_{j_1}, x^i_{j_2}, \ldots, x^i_{j_p}$ todas las variables de la forma x^i_j tales que $\tau(x^i_j) = 1$, y suponemos que se encuentran ordenadas de tal forma que $j_1 < j_2 < \cdots < j_p$ (y por lo tanto $a_{j_1} \leq a_{j_2} \leq \cdots < a_{j_p}$).
El lema 3.3 nos garantiza la existencia de una ruta R_i que pasa por todos los vértices $u_{j_1}, \ldots, u_{j_p}$ a las horas $a_{j_1}, \ldots, a_{j_p}$.

Para la existencia de dicha ruta, necesitamos comprobar que $a_{j_{k+1}} - a_{j_k} \geq \overline{d}(u_{j_k}, u_{j_{k+1}})$ para todo $k \in \{1, \ldots, p-1\}$. Esto se tiene gracias a que las variables $x^i_{j_k}$ y $x^i_{j_{k+1}}$ son ciertas simultáneamente, lo que impica que la cláusula $\neg x^i_{j_k} \vee \neg x^i_{j_{k+1}}$ no forma parte del problema, y que por lo tanto $(T_{j_k}, T_{j_{k+1}}) \in \overrightarrow{E_i}$, de donde obtenemos precisamente como queríamos la desigualdad

$$a_{j_k} + \overline{d}(u_{j_k}, u_{j_{k+1}}) \leq a_{j_{k+1}}$$

Tenemos definida por lo tanto, para cada actor A_i, una ruta R_i que cumplen que $\forall i \in \{1, \ldots, m\}$ y $\forall j \in \{1, \ldots, N\}$

$$\text{si } \tau(x^i_j) = 1 \text{ entonces } W_{R_i}(a_j) = u_j$$

Comprobamos que estas rutas hacen ciertos a todos los testimonios. Sea el testimonio T_j con $j \in \{1, \ldots, N\}$:

$$\varphi_j = x^p_j \vee x^q_j \text{ es cierta con la asignación } \tau \text{ (donde } \{A_p, A_q\} = \{A^j, B^j\})$$

$$\Rightarrow \tau(x^p_j) = 1 \text{ o } \tau(x^q_j) = 1$$

$$\Rightarrow W_{R_p}(a_j) = u_j \text{ o } W_{R_q}(a_j) = u_j$$

$$\Rightarrow A^j \text{ o } B^j \text{ visitaron el vértice } u_j \ \square$$

Existen algoritmos que resuelven *2-SAT* en tiempo lineal en el número de cláusulas, en concreto este [11] reduce el problema a encontrar las componentes fuertemente conexas sobre un grafo dirigido. Por lo tanto, suponiendo que utilizamos el algoritmo de Floyd-Warshall para obtener las distancias $\overline{d}$ entre todos los pares de vértices, obtenemos un algoritmo de orden $O(|V|^3 + N^2)$, ya que el número de cláusulas creado es de orden N^2 en el caso peor.

4.2.10 Actores que a la vez son testigos

Finalmente, el último elemento que nos queda por añadir al problema, es la posibilidad de que los actores digan testimonios sobre otros actores, es decir, que $\mathcal{A} \cap \mathcal{T}$ no sea necearimente vacío como veníamos asumiendo hasta ahora. En concreto estudiaremos una versión simple, con las siguientes características:

- Actores $\mathcal{A} = \{A_1, \ldots, A_m\}$
- Testigos $\mathcal{T} = \{A_1, \ldots, A_r, \overline{P}\}$ con $0 \leq r \leq m$, y $\overline{P} \notin \mathcal{A}$. Es decir, los primeros r actores son también testigos, y hay un testigo extra $\overline{P}$ que no es actor. Nos podemos permitir tener un sólo testigo extra, ya que nuestro objetivo es comprobar si todos los testimonios son ciertos o no, por lo tanto, al igual que hemos hecho en las versiones anteriores, si un testimonio ha sido dicho por un testigo que no es actor, da igual quien sea dicho testigo.
- Los testimonios especifican actor y vértice, es decir, son de la forma $P : T$ donde $T = (\{A\}, \{u\}, [a, b]) = (A, u, [a, b])$.

  Vamos a agrupar los testimonios en función de los actores a los que se refieren. Para cada actor A_i, unimos los testimonios que él ha dicho, junto con los que otro testigo haya dicho sobre el:

  $$\texttt{test}(A_) := \{(A : T) \in \Delta\} \cup \{(P' : T) \in \Delta \mid T = (A, u, I)\}$$

  Al igual que en las versiones anteriores, para que una instancia sea válida necesitamos que todos los intervalos que aparezcan en los testimonios sobre un mismo actor, $\texttt{test}(A_i)$, sean disjuntos dos a dos.
  Nótese que un testimonio dicho por un actor, $A_1 : (A_2, u, I)$, aparecerá tanto en $\texttt{test}(A_1)$ como en $\texttt{test}(A_2)$.

Ejemplo 4.5

A continuación damos un ejemplo en el que vemos como podemos interpretar gráficamente el problema. Supongamos que tenemos el conjunto de actores $\mathcal{A} = \{A_1, A_2, A_3\}$, testigos $\mathcal{T} = \{A_1, A_2, \overline{P}\}$, y los siguientes testimonios:

$$\overline{P} : (\{A_1\}, u_1, [2, 5]) \qquad \overline{P} : (\{A_1\}, u_2, [6, 6]) \qquad \overline{P} : (\{A_2\}, u_3, [0, 3]) \qquad \overline{P} : (\{A_3\}, u_4, [1, 2])$$

$$\overline{P} : (\{A_3\}, u_5, [8, 10]) \qquad A_1 : (\{A_2\}, u_6, [7, 10]) \qquad A_2 : (\{A_3\}, u_7, [4, 6])$$

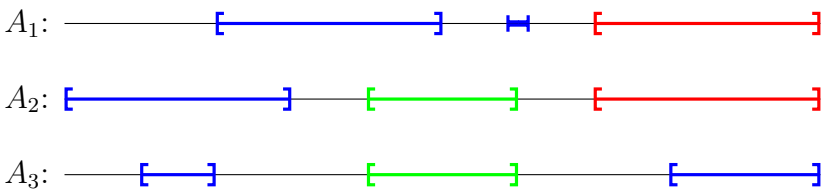
Entonces los conjuntos $\texttt{test}(A_i)$ ordenados quedan:

$\texttt{test}(A_1)$: $\overline{P} : (\{A_1\}, u_1, [2,5])$ $\overline{P} : (\{A_1\}, u_2, [6,6])$ $A_1 : (\{A_2\}, u_6, [7,10])$

$\texttt{test}(A_2)$: $\overline{P} : (\{A_2\}, u_3, [0,3])$ $A_2 : (\{A_3\}, u_7, [4,6])$ $A_1 : (\{A_2\}, u_6, [7,10])$

$\texttt{test}(A_3)$: $\overline{P} : (\{A_3\}, u_4, [1,2])$ $A_2 : (\{A_3\}, u_7, [4,6])$ $\overline{P} : (\{A_3\}, u_5, [8,10])$

Y los visualizaremos como en el siguiente dibujo, representando en azul los testimonios de P, y en otros colores los testimonios en común entre dos actores.

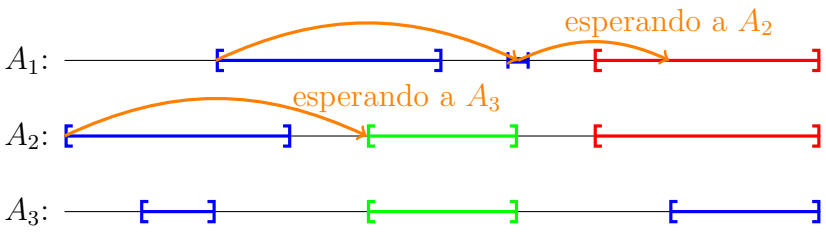


Para solucionar esta versión en tiempo polinómico, vamos a inspirarnos en el paralelismo: haremos que cada actor vaya ejecutando individualmente un algoritmo muy similar al de la versión con un actor e intervalos de tiempo disjuntos. La principal idea utilizada consiste en que los testimonios $A : (B, u, I)$ y $B : (A, u, I)$ son equivalentes, según la definición: Ambos actores tienen que coincidir en el vértice u a la misma hora.

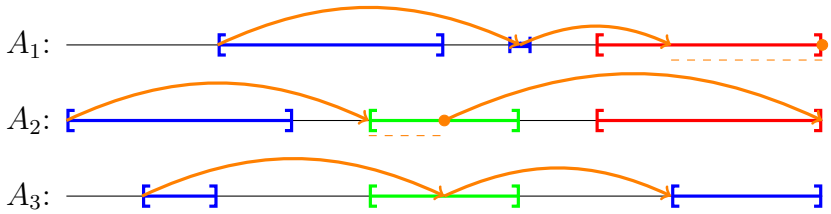
Lo que haremos será, por tanto, que cuando el actor A_i alcance un testimonio compartido con otro actor A_k, sin importar cual de los dos sea el testigo y sobre cual hable el testimonio, esperará en ese mismo vértice hasta que A_k también lo alcance, ya que ambos actores tienen que coincidir.
Entonces, si A_i puede llegar a las h_i, y A_k puede llegar a las h_k, el que llegue antes de los dos esperará el tiempo correspondiente en dicho vértice hasta que llegue el otro, de tal forma que ambos abandonan el vértice a la misma hora : $max(h_i, h_k)$.

Ejemplo 4.6

Ilustramos como se ejecutaría el algoritmo con los testimonios antes vistos. El testimonio representado en verde decía que A_2 y A_3 tienen que coincidir en un vértice de 4 a 6. Supongamos que A_2 puede llegar a las 4, y A_3 a las 5, y que el código del actor A_1 se ejecuta antes que el de A_2, por lo que se queda pausado hasta que llegue dicho actor. La siguiente imagen muestra una ejecución a medias, en la que A_1 está esperando a A_2, que está a su vez esperando a A_3.



En el momento en que A_3 alcanza el testimonio verde, A_2 reanuda su ejecución, teniendo que esperar una unidad de tiempo en el vértice correspondiente para coincidir con A_3 a la misma hora, ya que este llega más tarde.
La misma situación sucede posteriormente entre los actores A_1 y A_2.



A continuación mostramos el pseudocódigo que haría funcionar esta solución del problema, utilizando la misma notación empleada en la versión 2.3, ejecutándose el siguiente fragmento simultáneamente para todos los actores $A_1, \ldots, A_m$.

```
1  h_1 = a_1
2  for j = 2,...,N :
3
4      h_j = max(h_{j-1} + dist(u_{j-1},u_j), a_j) // Hora de llegada a u_j
5
6      if el testimonio T_j es compartido con A_k :
7          esperar(A_k)
8          Sea h' la hora a la que llega A_k
9          h_j = max(h_j,h')
10
11     if h_j > b_j: // Si no llega a tiempo, no cumple todos los testimonios
12         return false
13
14 return true
```

Nótese que, aunque estamos utilizando paralelismo porque surge de forma natural al haber varios actores moviéndose en un grafo, el algoritmo sigue siendo polinómico si solo se ejecuta un hilo a la vez. Además, no se produce interbloqueo, ya que si el actor A_i está esperando al actor A_k, necesariamente la hora en la que se encuentra el segundo actor es menor que la hora del primero, $h_k < h_i$. Por lo tanto, el actor con menor hora de entre todos los actores nunca puede estar bloqueado esperando a otro actor, así que siempre hay un hilo en ejecución.

Capítulo 5

Complejidad del Problema de Optimización *WPO*

En esta sección abordamos el problema de optimización *WPO* desde el punto de vista teórico, buscando demostrar la pertenencia y dureza en alguna de las clases de aproximabilidad antes mencionadas. En concreto, veremos en el primer apartado de la sección que la versión restringida del problema *rest-WPO(1)* es APX-dura. En el segundo apartado proponemos distintos algoritmos voraces para obtener soluciones aproximadas.

5.1 Pertenencia de *rest-WPO(1)* a APX-duro

Comenzamos con la demostración de que la versión restringida del problema *rest-WPO(1)*, donde sólo hay un actor, cada testigo dice un único testimonio, y no permitimos testimonios negados, es dura en la clase APX. Para ello, construiremos una reducción AP desde el problema de optimización *MAX3SAT*, definido como sigue:

Definición 5.1 *Dada una instancia del problema de decisión 3-SAT, es decir, un conjunto de variables lógicas* $\{x_1, \ldots, x_m\}$*, y* N *fórmulas lógicas de* 3 *variables* $\varphi_1, \ldots, \varphi_N$ *de la forma* $\varphi_j = a_j \vee b_j \vee c_j$ *con* $a_j, b_j, c_j \in \{x_1, \ldots, x_m, \overline{x}_1, \ldots, \overline{x}_m\}$*, el problema de optimización MAX3SAT consiste en encontrar una asignación* $\tau : \{x_1, \ldots, x_m\} \longrightarrow \{0, 1\}$ *que maximice el número de fórmulas ciertas.*

Teorema 8 *El problema rest-WPO(1) es APX-duro.*

Demostración: Vamos a comprobar que existe una reducción AP desde el problema *MAX3SAT* a nuestro problema *rest-WPO(1)*, *MAX3SAT* $\leq_{AP}$ *rest-WPO(1)*. Como *MAX3SAT* es APX-duro, y la reducción AP preserva la dureza en la clase APX, esto confirmará que *rest-WPO(1)* es también APX-duro.

Tomamos una instancia Φ del problema *MAX3SAT*, con n variables $x_1, \ldots, x_n$, y m fórmulas $\varphi_1, \ldots, \varphi_N$, de la forma $\varphi_j = a_j \vee b_j \vee c_j$, con $a_j, b_j, c_j \in \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$

Definimos la función f, que transforma instancias de *MAX3SAT* en instancias de *rest-WPO(1)* de la siguiente forma:

- $\mathcal{A} = \{A\}$ es el único actor del problema.
- $\mathcal{T} = \{P_1, \ldots, P_N\}$ son los testigos del problema, tantos como fórmulas lógicas $\varphi_1, \ldots, \varphi_N$, y tales que $A \notin \mathcal{T}$.
- Los vértices del grafo serán las variables de Φ, negadas y no negadas, y las aristas entre todo par de variables con índice consecutivo

$$V = \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$$

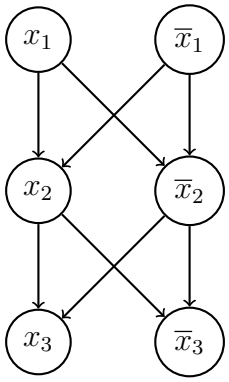
$$E = \{(x_{i-1}, x_i), (\overline{x}_{i-1}, x_i), (x_{i-1}, \overline{x}_i), (\overline{x}_{i-1}, \overline{x}_i) |\ i = 2, \ldots, n\}$$

  todas ellas con distancia 1

$$d(u, v) = 1 \quad \forall (u, v) \in E$$

  Intuitivamente, una ruta en este grafo dirigido se podrá transformar en una asignación de valores para las variables $x_1, \ldots, x_n$, ya que el sospechoso o bien pasa por x_i o bien pasa por $\overline{x}_i$, pero nunca pasará por ambas (también puede que no pase por ninguna, en cuyo caso tomaremos un valor arbitrario para $\tau(x_i)$).

  Por ejemplo, si tenemos sólo las 3 variables x_1, x_2, x_3 el grafo correspondiente será



- Finalmente transformamos cada cláusula $\varphi_j = a_j \vee b_j \vee c_j$ en el testimonio $T_j = (A, \{a_j, b_j, c_j\}, [0, n-1])$, dicho por la persona P_j. Un testimonio es cierto si A ha visitado alguno de los 3 lugares que aparecen en él, lo que implicará que en la asignación τ correspondiente alguna de las 3 variables a_j, b_j, c_j sea cierta.

Vamos a comenzar comprobando que si *MAX3SAT* admite una solución con M cláusulas ciertas, entonces la instancia transformada de *rest-WPO(1)* tiene una solución con al menos M testimonios ciertos.

Sea $\tau : \{x_1, \ldots, x_n\} \longrightarrow \{0, 1\}$ dicha asignación, definimos $\forall k \in \{1, \ldots, n\}$

$$w_k = \begin{cases} x_k & si \quad \tau(x_k) = 1 \\ \\ \overline{x}_k & si \quad \tau(x_k) = 0 \end{cases}$$

Tomamos la siguiente ruta para el sospechoso A, que garantizará que se cumplan M testimonios:

$$R = (w_1, 0)(w_2, 0) \cdots (w_n, 0)$$

Debido al lema 3.1 sobre rutas, tenemos que $\forall k \in \{1, \ldots, n\}$

$$first_R(k) = \sum_{i=2}^{k} d(w_{i-1}, w_i) + \sum_{i=1}^{k-1} t_i$$

$$= \sum_{i=2}^{k} 1 + \sum_{i=1}^{k-1} 0 = k - 1$$

Por lo tanto $W_R(k-1) = w_k \quad \forall k \in \{1, \ldots, n\}$.

Sea $\varphi_j = a_j \vee b_j \vee c_j$ una cláusula cierta con las asignaciones de τ, podemos suponer sin pérdida de generalidad que $\tau(a_j) = 1$ (entendiendo que $\tau(\overline{x}) = 1$ equivale a $\tau(x) = 0$).

- Si $a_j = x_k$, con $k \in \{1, \ldots, n\}$, entonces $W_R(k-1) = w_k = x_k = a_j$, haciendo al testimonio $T_j = (A, \{a_j, b_j, c_j\}, [0, n-1])$ cierto, ya que uno de sus vértices, a_j, fue visitado a tiempo.
- Análogamente si $a_j = \overline{x}_k$, tenemos que $\tau(x_k) = 0$, por lo tanto $W_R(k-1) = w_k = \overline{x}_k = a_j$, haciendo al testimonio $T_j = (A, \{a_j, b_j, c_j\}, [0, n-1])$ cierto.

Esto hace que R sea una solución de la instancia $f(\Phi)$ donde al menos M testimonios son ciertos, por lo que se cumple que:

$$\texttt{opt}_{\textit{rest-WPO(1)}}(f(\Phi)) \geq \texttt{opt}_{MAX3SAT}(\Phi)$$

A continuación definimos la función g, que transforma soluciones factibles de $f(\Phi)$ en soluciones factibles de Φ. Dada una ruta R solución de $f(\Phi)$, defimos que $g(\Phi, R) = \tau$ tal que:

$$\tau(x_i) = \begin{cases} 1 & \text{si } R \text{ pasa por } x_i \\ \\ 0 & \text{si } R \text{ pasa por } \overline{x}_i \end{cases}$$

Nótese que la función no está completamente definida, pues la ruta podría no pasar por ninguno de los dos vértices, sin embargo, si la ruta no pasa ni por x_i ni por $\overline{x}_i$ el valor que le demos a τ va a ser irrelevante por lo que podemos definirlo a cualquier valor.
Comprobamos que, si R es una solución de $f(\Phi)$ donde hay M testimonios ciertos, entonces $g(\Phi, R)$ será una solución de Φ con al menos M cláusulas ciertas.

Si $T_j = (s, \{a_j, b_j, c_j\}, [0, n-1])$ es un testimonio cierto, entonces existen un vértice del conjunto $d \in \{a_j, b_j, c_j\}$ y un tiempo $t \in [0, n-1]$ tales que la ruta visita dicho vértice en ese tiempo, $W_R(t) = d$, por lo que $\tau(d) = 1$, lo que hace que la cláusula $\varphi_j = a_j \vee b_j \vee c_j$ sea cierta.

Con esto queda demostrada la siguiente desigualdad:

$$\mathtt{m}(f(\Phi), R) \leq \mathtt{m}(\Phi, g(\Phi, R))$$

Y como caso particular tendremos que $\mathtt{opt}$*rest-WPO(1)*$(f(\Phi)) \leq \mathtt{opt}_{MAX3SAT}(\Phi)$, que juntándola con la desigualdad obtenida previamente nos garantiza la igualdad de las soluciones óptimas.

Finalmente, para ver que se trata de una reducción AP, demostramos que la constante $c = 1$ garantiza se cumplen las condiciones sobre el ratio de la reducción AP:

$$\text{si} \quad R_{\textit{rest-WPO(1)}}(f(\Phi), R) =\leq r$$

$$\text{entonces} \quad R_{MAX3SAT}(\Phi, g(\Phi, R)) = \frac{\mathtt{opt}_{MAX3SAT}(\Phi)}{\mathtt{m}(\Phi, g(\Phi, R))} = \frac{\mathtt{opt}_{\textit{rest-WPO(1)}}(f(\Phi))}{\mathtt{m}(\Phi, g(\Phi, R))} \leq$$

$$\leq \frac{\mathtt{opt}_{\textit{rest-WPO(1)}}(f(\Phi))}{\mathtt{m}(f(\Phi), R))} = R_{\textit{rest-WPO(1)}}(f(\Phi), R) \leq r = 1 + c(r-1) \quad \square$$

5.2 Algoritmos Voraces

Hasta ahora hemos demostrado que la versión del problema *rest-WPO(1)*, que tiene un sólo actor, y admite sólamente un testimonio por testigo y sin testimonios negados, es dura en la clase APX. Sin embargo, desconocemos si pertenece en efecto a la clase APX o no. En esta sección exploramos distintos algoritmos voraces con el objetivo inicial de encontrar alguno de ellos que obtenga soluciones aproximadas con un ratio acotado por una constante. Desafortunadamente, para cada algoritmo propuesto mostramos también un contraejemplo que sirve para probar que el ratio obtenido no cumple la cota que deseamos.

Por lo tanto, en el plano teórico no vamos a obtener ningún resultado, ni positivo ni negativo, más que comprobar que algunos métodos voraces intuitivos no consiguen el resultado deseado. Veremos sin embargo como en la práctica la mayoría de los algoritmos sí que resultan

ser eficaces, encontrando soluciones generalmente cercanas a la óptima.

En este apartado exponemos los algoritmos en cuestión junto a sus contraejemplos, justificando que el ratio alcanzado no se puede acotar por una constante. En el siguiente capítulo del trabajo utilizamos estos algoritmos junto a un algoritmo genético para obtener y analizar resultados sobre el funcionamiento de los algoritmos en la práctica.

5.2.1 Voraz por hora inicial

El primer algoritmo voraz que expondremos consiste en ordenar todos los testimonios de la instancia $T_1, \ldots, T_N$, siendo $T_j = (A, L_j, [a_j, b_j])$ con $j \in \{1, \ldots, N\}$, de tal forma que las horas iniciales a las que se debe estar en cada testimonio queden ordenadas de menor a mayor:

$$a_1 \leq a_2 \leq \cdots \leq a_N$$

Este algoritmo voraz, $\texttt{A}_1$, dará por cierto el primer testimonio de todos en esta lista, situando al actor A en un vértice arbitrario v_1 del conjunto L_1 a la hora a_1.

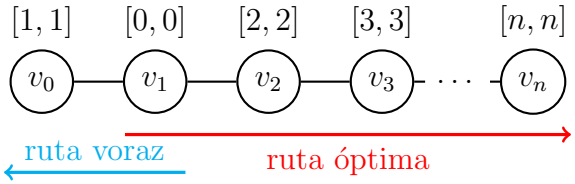
Entonces el algoritmo ejecuta un bucle que recorre cada testimonio T_j de la lista ordenada $T_2, \ldots, T_N$. Si el actor puede llegar a algún vértice de L_j a tiempo, es decir, entre las a_j y las b_j, entonces traslada al actor desde donde estuviera hasta el vértice $v \in L_j$ que minimice la hora de llegada, añadiendo a la ruta todos los vértices intermedios del camino más corto para llegar a v. Si fuera imposible que el actor tomara este testimonio, descarta T_j y procede a mirar el siguiente, T_{j+1}.

```
tiempo = a_1
vertice = Vértice arbitrario de L_1
for j = 2,...,N :
    u = vértice de L_j que minimiza d(vertice,u)
    if tiempo + d(vertice,u) ≤ b_j :
        tiempo = max(a_j, tiempo + d(vertice,u))
        vertice = u
        modificar_ruta(vertice, u, tiempo)
```

A continuación mostramos un contraejemplo que nos sirve para probar que el ratio alcanzado por este algoritmo puede llegar a ser, en el caso peor, polinómico.

Contraejemplo 5.1

Tomamos el grafo de la siguiente imagen, formado por $n + 1$ vértices, donde las distancias de todas las aristas valen 1.



La solución óptima en este grafo se puede conseguir con la ruta $(v_1, 1)(v_2, 0)\cdots(v_n, 0)$, alcanzando un total de n testimonios ciertos. Sin embargo, el algoritmo voraz toma primero el testimonio $(A, v_1, [0, 0])$, posteriormente toma $(A, v_0, [1, 1])$, y a partir de este punto el resto de testimonios son inalcanzables, obteniendo una solución de valor 2. Por lo tanto el ratio para el algoritmo $\mathtt{A}_1$ en esta instancia es:

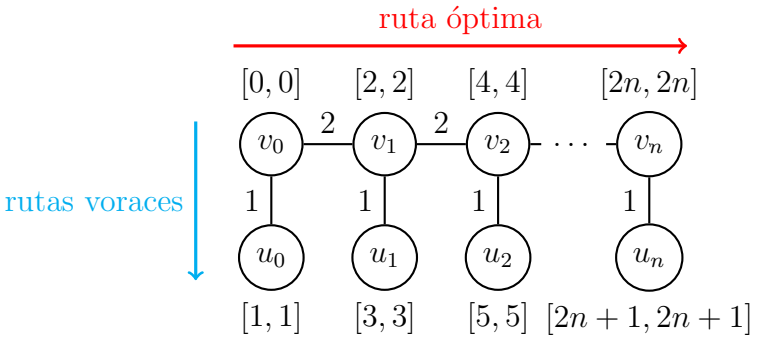
$$R(\mathtt{x}, \mathtt{A}_1(\mathtt{x})) = \frac{\mathtt{opt}(\mathtt{x})}{\mathtt{m}(\mathtt{x}, \mathtt{A}_1(\mathtt{x}))} = \frac{n}{2}$$

no pudiendo ser acotado por una función inferior a polinómica.

Sin embargo, nos planteamos qué sucedería si probáramos a repetir este mismo algoritmo voraz, cambiando la fase inicial, porque en este mismo ejemplo se obtendría un valor mucho mejor si el algoritmo tomara como testimonio inicial el que indica que el actor está en v_2. Llamaremos *algoritmo persistente de* $\mathtt{A}_1$ a este nuevo algoritmo $\mathtt{A}'_1$, que realiza N ejecuciones consecutivas de $\mathtt{A}_1$, iniciando cada vez en un testimonio distinto. Desafortunadamente también existe un contraejemplo en el que el algoritmo voraz obtiene un ratio necesariamente polinómico.

Contraejemplo 5.2

La idea parte de crear un grafo con una solución óptima lineal, y para cada vértice de esta solución colocar un vértice de cebo, de tal forma que el voraz lo considere mejor, pero no lo sea en realidad. Creamos el grafo como muestra la siguiente imagen:



Los testimonios de la instancia son:

$$(A, v_k, [2k, 2k]) \qquad (A, u_k, [2k+1, 2k+1]) \qquad \forall k \in \{0, \ldots, n\}$$

Observamos que en el grafo del dibujo podemos obtener una ruta que cumpla el máximo número de testimonios mediante la secuencia de vértices $v_0 v_1 \ldots v_n u_n$, obteniendo una solución óptima con $n + 2$ testimonios ciertos.
Sin embargo, si ejecutamos el algoritmo voraz, hay dos opciones posibles:

- Si comienza en un testimonio de vértice u_k, $k \in \{0, \ldots, n\}$, entonces no podemos llegar a tiempo a cumplir ningún otro testimonio, y por lo tanto la solución obtenida alcanza un valor de 1.

- Comenzando en el testimonio $(A, v_k, [2k, 2k])$, el siguiente testimonio a considerar por el algoritmo, al estar ordenados en función de la hora inicial, es $(A, u_k, [2k+1, 2k+1])$, el cuál es alcanzable desde v_k, pero una vez llegado a u_k no se puede cumplir ningún otro testimonio de la instancia, dando lugar a una solución de valor 2.

Por lo tanto el algoritmo voraz $\mathtt{A}'_1$, que ejecuta $\mathtt{A}_1$ comenzando desde cada vértice del grafo y se queda con el mejor valor obtenido, consigue un ratio para esta instancia de

$$R(\mathtt{x}, \mathtt{A}'_1(\mathtt{x})) = \frac{\mathtt{opt(x)}}{\mathtt{m}(\mathtt{x}, \mathtt{A}'_1(\mathtt{x}))} = \frac{n+2}{2}$$

5.2.2 Voraz por hora final

El siguiente algoritmo voraz $\mathtt{A}_2$ que comentamos es, en esencia, igual que el anterior, pero en lugar de ordenar los testimonios $T_1, \ldots, T_N$ por la hora inicial, los ordenamos por la hora final, es decir

$$b_1 \leq b_2 \leq \cdots \leq b_N$$

Como cabe esperar, el funcionamiento de este algoritmo es igual de malo que el anterior, sirviéndonos de hecho los mismos contraejemplos expuestos antes para comprobar que el ratio alcanzado puede llegar a ser polinómico, tanto en el caso normal como en el caso persistente $\mathtt{A}'_2$.

5.2.3 Voraz por tiempo

Otro método voraz que nos planteamos, de forma natural, es tomar en cada paso del algoritmo el testimonio que podamos alcanzar en el menor tiempo posible. Dados un vértice $v \in V$ y un tiempo $t \in \mathbb{N}$, definimos la siguiente métrica sobre un testimonio $T = (A, L, [a, b])$, que nos indica el tiempo más pequeño en el que puede ser alcanzado :

$$llegada_{v,t}(T) = t + min\{\overline{d}(v, u) \mid u \in L\}$$

$$tiempo_{v,t}(T) = \begin{cases} \infty & \text{si} \quad llegada_{v,t}(T) > b \\ a & \text{si} \quad llegada_{v,t}(T) < a \\ llegada_{v,t}(T) & \text{si} \quad llegada_{v,t}(T) \in [a, b] \end{cases}$$

El algoritmo $\mathtt{A}_3$ consistirá en, partiendo de un vértice del grafo arbitrario a las 0, tomar por cierto a cada paso un testimonio, de los que no han sido elegidos aún, que minimice la métrica $tiempo_{v,t}$. De forma similar a los algoritmos anteriores, podemos considerar su versión persistente $\mathtt{A}'_3$ que consiste en aplicar $\mathtt{A}_3$ repetidas veces, comenzando cada vez desde un vértice del grafo distinto.

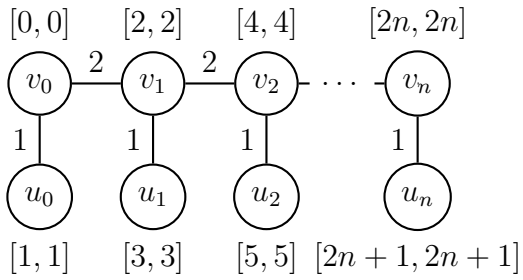
```
voraz_tiempo(v, t) :
    testimonios = T_1,...,T_N
    while ∃ T ∈ testimonios : tiempo_{v,t}(T) < ∞
        Sea T = (A, L, [a, b]) = un testimonio que minimiza tiempo_{v,t}(T)
        u = vértice de L que minimiza d̄(v,u)
        t = tiempo_{v,t}(T)
        v = u // Nos desplazamos a dicho v rtice a la hora correspondiente
        modificar_ruta(v, u, t)
        testimonios = testimonios\{T}
```

Como la métrica $tiempo_{\mathtt{v},\mathtt{t}}(T)$ se calcula con coste $O(|L|) \subseteq O(|V|)$ si hemos calculado las distancias entre cada par de vértices previamente (por ejemplo con el algoritmo de Floyd), como encontrar el testimonio que minimiza dicha métrica se puede hacer en tiempo lineal en N, el algoritmo $\mathtt{A}_3$ tiene un coste en $O(|V|N^2)$, y su versión persistente $\mathtt{A}'_3$ tiene por lo tanto un coste computacional de $O(|V|^2N^2)$.

El ratio de aproximación de este algoritmo es también polinómico en el tamaño de la instancia en el caso peor, pudiendo utilizar de nuevo el contraejemplo antes visto.

Contraejemplo 5.3

Tomando el grafo de la figura:



Al igual que antes, si el algoritmo comienza en un testimonio de vértice u_k, es incapaz de hacer cierto ningún testimonio más. Si empieza desde un testimonio de vértice v_k, puede hacer cierto el testimonio del vértice u_k en tiempo 1, mientras que el testimonio de vértice v_{k+1} en tiempo 2 (y el resto de testimonios en tiempos mayores), por lo que el algoritmo toma el vértice u_k, quedándose atascado. Por lo tanto, $\mathtt{m}(\mathtt{x}, \mathtt{A}'_3(\mathtt{x})) = 2$.

$$R(\mathtt{x}, \mathtt{A}'_3(\mathtt{x})) = \frac{n+2}{2}$$

5.2.4 Voraz por Testimonios Alcanzables

El último método voraz con el que nos planteamos resolver el problema es el más sofisticado de todos ellos, así como el más costoso computacionalmente. Sin embargo que en este caso el contraejemplo que proponemos tan solo demuestra que el ratio de aproximación puede

llegar a ser proporcional a la raíz cúbica del tamaño de la instancia, lo cual, aún siendo una cota inferior a las anteriores, es suficiente para ver que este algoritmo tampoco nos sirve para comprobar si el problema pertenece a APX.

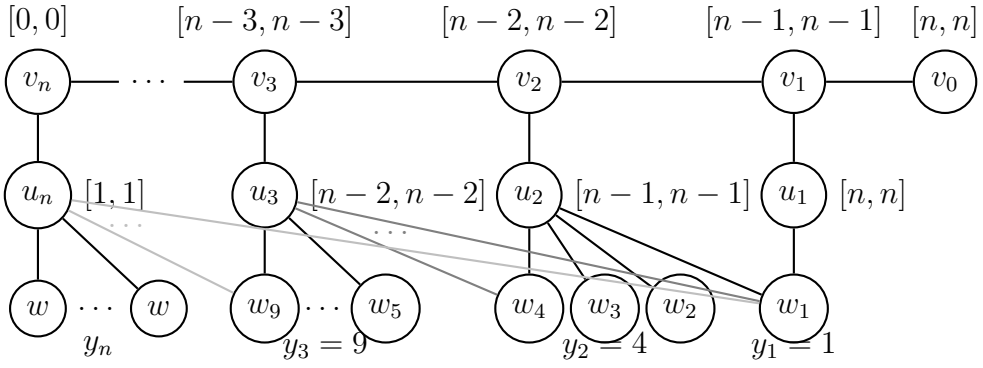
El algoritmo $\mathtt{A}_4$ en cuestión parte de un primer testimonio que da por cierto, y a cada paso selecciona el testimonio, de entre los que no hayan sido seleccionados aún, desde el cuál se pueda seguir alcanzando un número mayor de testimonios. El algoritmo $\mathtt{A}'_4$ es la versión persistente del mismo. Esta medida queda recogido en la siguiente definición:

Definición 5.2 *Sean $v \in V$ un vértice del grafo, $t \in \mathbb{N}$ la hora de salida de dicho vértice, y $T = (A, L, [a, b])$ un testimonio. Sean $h = tiempo_{v,t}(T)$ la menor hora a la que se puede cumplir el testimonio, y $u \in L$ el vértice del testimonio alcanzable a dicha hora.*

$$alcanzables_{v,t}(T) = \left|\{T' \in \Delta \mid T' \textit{se puede cumplir saliendo desde el vértice } u \textit{ a la hora } h\}\right|$$

Contraejemplo 5.4

El siguiente grafo parte de la misma idea que los anteriores, donde para cada vértice del camino óptimo, tenemos un vértice cebo que mejora la métrica *alcanzables*, pero sin embargo nos aleja de la solución óptima. Estos nodos cebo consisten en nodos con muchos vértices adyacentes, cada uno con un testimonio, de tal forma que desde u_k sean alcanzables todos los testimonios, pero una ruta solo pueda tomar uno de ellos. En concreto, el nodo u_k deberá tener tantos vértices como testimonios sean alcanzables desde v_2. De esta forma, cuando el algoritmo se encuentre en el vértice v_k a la hora t, la métrica *alcanzables* considerará mejor (por una unidad) desplazarse hacia u_k que hacia v_{k-1}. Renombraremos los vértices en el orden contrario respecto a los anteriores ejemplos por simplicidad en el argumento, y asumimos que la distancia de todas las aristas es 1.



Formalizando, los testimonios que tenemos sobre este grafo son:

$$(A, v_k, [n-k, n-k]) \quad \forall k \in \{0, \ldots, n\}$$

$$(A, u_k, [n-k+1, n-k+1]) \quad \forall k \in \{1, \ldots, n\}$$

Además, desde cada vértice u_k, añadimos $y_k - y_{k-1} \in \mathbb{N}$ vértices trampa (asumiendo $y_0 = 0$) cada uno de ellos con un testimonio:

$$(A, w_i, [n-k+2, n-k+2]) \quad \forall i \in \{y_{k-1}+1, \ldots, y_k\}$$

A continuación discutimos los valores que deben tomar los índices y_k, que indican el número de aristas a vértices trampa que salen desde u_k. Si nos encontramos en el vértice v_k, al deplazarnos hacia el v_{k-1}, son alcanzables los vértices $v_{k-1}v_{k-2}\cdots v_0$, los vértices $u_{k-1}u_{k-2}\cdots u_1$, y todos los vértices $w_{y_{k-1}}, \ldots, w_1$, sumando una cantidad de $y_{k-1}+k+(k-1)$. Por lo tanto, el valor de y_k tendrá que ser igual a esa suma, para que la métrica alcanzables del vértice u_k sea justo $y_k + 1$ (al contar también el testimonio del propio u_k). Estos índices quedan definidos mediante la sucesión recursiva

$$\begin{aligned} y_1 &= 1 \\ y_k &= k + k - 1 + y_{k-1} \end{aligned}$$

y se comprueba fácilmente que $y_k = k^2$.

A diferencia de los contraejemplos anteriores, el parámetro n no es proporcional al tamaño de la instancia del problema. Esta instancia consta de un número de aristas superior a $y_1 + y_2 + y_3 + \cdots + y_n$ (las que unen cada u_k con cada vértice $w_1 \cdots w_{y_k}$). Tenemos por lo tanto que el tamaño de nuestra instancia pertenece a $O(n^3)$:

$$y_1 + y_2 + y_3 + \cdots + y_n = 1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} \in O(n^3)$$

Comprobamos la calidad de la solución obtenida por el algoritmo $\mathtt{A}_4$ según cual sea su testimonio inicial.

- Si comienza en un testimonio $(A, w_k, [n+2-k, n+2-k])$, viendo el dibujo es fácil ver que, por la hora a la que ha de estar en él, no se puede alcanzar ningún otro testimonio, y por lo tanto la solución devuelta es 1.
- Si comienza en un testimonio u_k a la hora $t = n+1-k$, solamente se podrán alcanzar a tiempo los testimonios de los vértices $w_1 w_2 \cdots w_{y_k}$, el algoritmo elige uno de ellos arbitrariamente, y a continuación no puede hacer más testimonios ciertos, por lo que se detiene con una solución de valor 2.
- Finalmente, si comienza en un testimonio v_k a la hora $t = n-k$, el algoritmo comprueba que seleccionar el testimonio T_1 del vértice v_{k-1} evalúa a $alcanzables_{v_k,t}(T_1) = y_k$, mientras que seleccionar el testimonio T_2 asociado al vértice u_k evalúa a $alcanzables_{v_k,y}(T_2) = y_k + 1$, siendo el que alcanza el máximo también respecto al resto de testimonios. Por lo tanto traslada al actor al vértice u_k, desde donde hemos visto que obtiene una solución de valor 2, por lo que en total, alcanza una solución de valor 3.

Sin embargo, la solución óptima consiste en visitar los vértices $v_n \cdots v_2 v_1 u_1 w_1$, teniendo tamaño $n+2$. Se alcanza el siguiente ratio:

$$R(\mathtt{x}, \mathtt{A}'_4(\mathtt{x})) = \frac{n+2}{3}$$

Como el tamaño de la instancia es $O(n^3)$, en este caso $\frac{n+2}{3}$ no es polinómico en $|\mathtt{x}|$. Lo que sí nos garantiza este ejemplo es que existen instancias tales que el ratio obtenido por este algoritmo es del orden de magnitud de la raíz cúbica del tamaño de la instancia, $\frac{n+2}{3} \in O(n) \subseteq O(\sqrt[3]{|\mathtt{x}|})$. De igual forma, queda demostrado que el ratio (para este algoritmo en concreto) no está acotado por una constante.

Finalmente, mencionar que modificando adecuadamente las distancias de las aristas del grafo anterior, y las horas de los testimonios, podemos obtener una instancia del problema en la que dan malos resultados los cuatro algoritmos voraces estudiados, por lo tanto combinar varios de ellos no mejorará el orden del ratio de aproximación.

Capítulo 6

Resolución práctica

En este capítulo comenzamos resumiendo el funcionamiento general de los algoritmos genéticos, y posteriormente explicamos como han sido utilizados para resolver de forma aproximada instancias del problema *WPO(1)*, la versión del problema con un sólo actor. La principal fuente de donde hemos obtenido la información teórica es [12].

Finalmente, realizamos una comparación estadística de los resultados obtenidos por estos algoritmos y los algoritmos oraces de la sección previa.

6.1 Introducción a los Algoritmos Genéticos

Los algoritmos genéticos son un tipo de algoritmos comunmente utilizados para la resolución aproximada de problemas de optimización y búsqueda, inspirados en el funcionamiento de la selección natural. Consiste en, partiendo de una población inicial formada por posibles soluciones del sistema, realizar un proceso de **selección** para quedarse con los individuos más aptos, y dar lugar a una nueva generación, creada a partir de ellos mediante funciones de **cruce** y **mutación**.

Explicamos con más detalle los distintos elementos a ser implemetados:

- **Cromosomas:** La representación de una solución del problema recibe el nombre de cromosoma, que son los elementos que conforman una generación de individuos. Normalmente son cadenas formadas por 0s y 1s de longitud fija, pues esto simplifica el resto de funciones a implementar.
- **Población inicial:** Se genera una población de cromosomas aleatoriamente, que normalmente consta de cientos o miles de individuos. También se pueden incluir en la población inicial algunas soluciones generadas con métodos mejores, como estrategias voraces.
- **Fitness:** Una función objetivo que evalúa lo buena o mala que es una solución, con el objetivo de determinar la calidad de un cromosoma como solución del problema. Puede

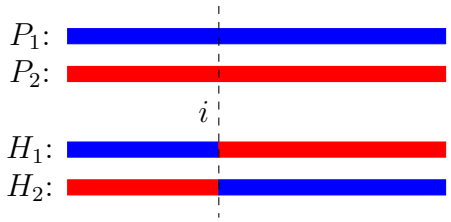
tomarse directamente la función objetivo del problema de optimización a resolver, pero en muchas ocasiones se modifica con la intención de penalizar soluciones que no son factibles o son muy negativas.

- **Selección:** Se encarga de seleccionar a los individuos más aptos de una población para dar lugar a la siguiente generación de individuos a partir de ellos. Para decidir qué individuos son mejores que otros se utiliza la función fitness.

  Nosotros utilizaremos en concreto el método de selección por torneo, que consiste en seleccionar varios grupos aleatorios de k individuos, y dentro de cada grupo, elegir el individuo con mejor valor de fitness con probabilidad p, el siguiente individuo con probabilidad $(1-p)p$, el siguiente con $(1-p)^2p\ldots$

- **Cruce:** Este operador genético toma dos cromosomas de una población para ser utilizados como padres, y genera dos hijos a partir de ellos, que normalmente comparten muchas características en común con los padres, para formar parte de la siguiente generación.

  Uno de los métodos de cruce más frecuentemente utilizados se trata de la recombinación en un punto, y funciona particularmente bien en problemas cuyos cromosomas tienen longitud fija n. Consiste en que dadas las cadenas de los dos padres $P_1 = x_1x_2\cdots x_n$ y $P_2 = y_1y_2\cdots y_n$, se selecciona un índice aleatorio $i \in \{1,\ldots,n\}$, y se crean dos hijos partiendo a los padres en el índice i, y mezclando las dos mitades: $H_1 = x_1\cdots x_iy_{i+1}\cdots y_n$, $H_2 = y_1\cdots y_ix_{i+1}\cdots x_n$.



- **Mutación:** El operador mutación es necesario para mantener la diversidad en la población, y consiste en ejecutar, con probabilidad p_{mut}, un cambio aleatorio sobre un cromosoma de una población. Un tipo de mutación común, en problemas donde los cromosomas están codificados como cadenas de 1s y 0s, consiste en cambiar un 1 por un 0 o viceversa en una posición del cromosoma seleccionada aleatoriamente.

El algoritmo genético que usaremos sigue la siguiente estructura.

1. Generar una población inicial de n individuos, usando la función **población inicial**.
2. Repetir los siguientes pasos.
   - Calcular el valor fitness para cada miembro y de la población.

- Elegir mediante el proceso de **selección** $n \cdot p_{padre}$ padres, siendo p_{padre} un parámetro que indica la proporción de individuos de una población que pasan a ser padres en la siguiente. Seleccionar también $n \cdot (1 - p_{padre})$ individuos que pasarán directamente a la siguiente generación.
- Aplicar la función **cruce** sobre los indiviudos seleccionados como padres, de dos en dos, para dar lugar a la misma cantidad de hijos.
- Aplicar la mutación a dichos hijos recién generados, así como a los individuos que pasan directamente a la siguiente generación, con probabilidad p_{mut} de que se efectúe el cambio. Juntando ambos grupos, volvemos a contar con una población de tamaño n.

3. El algoritmo finaliza tras un determinado número de pasos dado como parámetro, y devuelve el individuo con mayor valor de fitness encontrado hasta el momento.

6.2 Algoritmo Genético para el Problema *WPO(1)*

En este apartado, comentaremos cómo hemos aplicado el algoritmo genético sobre el problema *WPO(1)*, la versión con un sólo actor del problema WPO, y las distintas opciones que planteamos para algunas de las funciones requeridas. Recordemos que, el problema consiste en deteminar el máximo número de testigos que pueden estar diciendo la verdad simultáneamente, en instancias en las que sólo hay un actor A.

- **Cromosomas**: La codificación escogida para un cromosoma será la propia ruta $R = (w_1, t_1) \cdots (w_s, t_s)$ del único actor A, de igual forma que hicimos en el plano teórico, ya que se trata de una representación sencilla al mismo tiempo que no muy costosa en memoria, y nos será fácil implementar el resto de las funciones con ella.

- **Población inicial**: Compararemos dos funciones distintas de población inicial, por un lado $\texttt{ini}_1$, que genera n rutas completamente aleatorias, y por otro, $\texttt{ini}_2$, que toma las soluciones de los ocho algoritmos voraces estudiados en el capítulo previo (los cuatro métodos $\texttt{A}_1, \texttt{A}_2, \texttt{A}_3$ y $\texttt{A}_4$ y sus versiones persistentes), junto a $n - 8$ rutas aleatorias. Estos algoritmos funcionaban para la versión *rest-WPO(1)* del problema, así que solo admiten un testimonio por testigo, y no admiten testimonios negados. Por ello, utilizaremos la versión $\texttt{ini}_2$ para generar la población inicial sólo cuando estemos tratando con instancias del problema *rest-WPO(1)*.

  Respecto a la forma de creación de rutas aleatorias, han sido tomadas las siguientes decisiones:

  1. Primero seleccionamos uniformemente un vértice aleatorio del grafo v.

2. Luego seleccionamos un vértice final u aleatorio también, teniendo en cuenta que u debe ser alcanzable desde v en un tiempo menor o igual que M, donde M es la hora más tardía que aparece en algún testimonio de la instancia.

3. Posteriormente, se genera una ruta aleatoria que comience en v a las 0, y termine en u a las M. Para ello, a la hora de seleccionar un vértice w_k que añadir a la ruta, y el tiempo t_k de espera en dicho vértice, el algoritmo se asegura de que w_k y t_k son tales que permiten llegar a u en el tiempo requerido. Nótese que estamos obligando a que la duración de todas las rutas de la población sea la misma, M, lo cual implica, como veremos en el siguiente ejemplo, que el algoritmo puede perderse soluciones óptimas. Esto no supone un gran problema porque se puede arreglar como explicamos también en el ejemplo siguiente:

   Ejemplo 6.1

   Sea un grafo $G = (V, E)$, un conjunto de un actor $\mathcal{A} = \{A\}$ y un conjunto de testigos $\mathcal{T}$. Si tenemos una instancia en la que hay un testimonio de la forma $\neg(A, V, [1, M])$, diciendo que el actor no estuvo en ningún lugar desde la 1 hasta la última hora del día M, entonces todas las rutas consideradas por el algoritmo dan este testimonio por falso, ya que a las M se encuentran siempre en algún vértice.

   En general, el hecho de permitir rutas de duración estrictamente menor que M puede dar lugar a soluciones mejores que no hacerlo, debido a la existencia de testimonios negados. Sin embargo, si tenemos la sospecha de que estamos teniendo este problema en una instancia, podemos hacer el siguiente truco:

   Añadimos un vértice $\overline{v}$ al grafo, que simbolice no estar en ningún vértice, y lo unimos con distancia 1 al resto de vértices.

   $$G' = (V', E') \qquad V' = V \cup \{\overline{v}\} \qquad E' = E \cup \{(v, \overline{v}) \mid v \in V\}$$

   $$d(\overline{v}, v) = d(v, \overline{v}) = 1 \quad \forall v \in V$$

   De esta forma, existe una equivalencia entre una ruta $R = (w_1, t_1) \cdots (w_s, w_s)$ sobre el grafo G de distancia menor estricta que M, y una ruta de distancia exactamente M, $R' = (w_1, t_1) \cdots (w_s, w_s)(\overline{v}, M - time(R))$ sobre el grafo G', que recorre R y al terminar pasa el tiempo que le queda en el vértice nuevo $\overline{v}$, así que añadiendo un sólo vértice podemos resolver el problema original.

4. Finalmente, para las elección aleatoria de los tiempos de espera en un vértice, no hemos querido utilizar una distribución uniforme, porque daría pie a rutas que visitan muy pocos vértices distintos, y pasan gran parte del tiempo esperando en el mismo vértice. En su lugar, hemos empleado una distribución geométrica, de probabilidad p ajustable como parámetro.

Esta distribución está definida sobre todos los naturales, y da mayor peso a los números más pequeños, siendo la probabilidad de cada número la siguiente

$$P(Z = k) = p(1-p)^k$$

Podemos ver que la suma de probabilidades sobre todos los naturales es 1.

$$\sum_{k=0}^{\infty} p(1-p)^k = p \sum_{k=0}^{\infty} (1-p)^k = p \, \frac{1}{1-(1-p)} = p \, \frac{1}{p} = 1$$

utilizando la fórmula de las series geométricas:

$$\sum_{k=0}^{\infty} r^k = \frac{1}{1-r} \qquad \text{si } r < 1$$

- **Fitness**: Como función fitness hemos tomado la propia función objetivo del problema: el número de testigos que dicen la verdad.

- **Selección**: Como proceso de selección hemos tomado la selección por torneo, con torneos de $k = 4$ participantes.

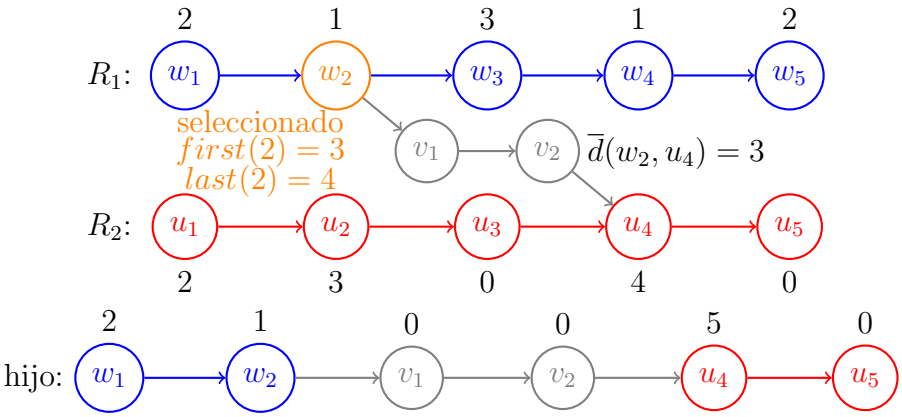
- **Cruce**: Proponemos dos alternativas distintas para el cruce de dos rutas padres $R_1 = (w_1, t_1) \cdots (w_{s_1}, t_{s_1})$ y $R_2 = (u_1, h_1) \cdots (u_{s_2}, h_{s_2})$, ambas inspiradas en la información de [13], pero personalizadas y adaptadas al contexto nuestro problema.

  Primer método de cruce: `cruce`$_1$. Este método intenta adaptar el concepto de la recombinación en un punto a nuestro modelo de cromosomas, ya que en nuestro caso no puede hacerse exactamente como describimos anteriormente puesto que la ruta resultante podría contener vértices consecutivos no adyacentes en el grafo, o podría tener duración distinta de M. Para ello lo que haremos será seleccionar un vértice w_i de la primera ruta, donde comenzará el cambio, y buscamos el primer vértice u_j de la segunda ruta que sea alcanzable a tiempo desde w_i, es decir, que $\overline{d}(w_i, u_j) \leq first_{R'}(j) - last_R(i)$.

Ejemplo 6.2

En la siguiente imagen mostramos un ejemplo de cruce de dos rutas, donde los nodos representan los vértices w_k visitados por cada ruta, y los números asociados a cada nodo representan los tiempos de espera t_k, además asumimos que todas las distancias valen 1.
En el ejemplo, el vértice seleccionado aleatoriamente para comenzar el cruce ha sido w_2, y la ruta R_1 abandona este vértice a las $last_R(2) = 4$. Suponemos también que,

a través de otros vértices del grafo, v_1 y v_2, llegamos en distancia 3 al vértice u_4 de la otra ruta padre, y que este es el primer vértice de la segunda ruta al que se puede llegar a tiempo. Como la ruta R_2 abandona dicho vértice a las $last(4) = 12$, nos tenemos que asegurar que la ruta hijo creada también lo haga a la misma hora, para ello modificamos el tiempo de estancia en u_4 en la ruta hijo.



Para formalizar este método, introducimos el concepto de concatenación de rutas.

Definición 6.1 *(Concatenacion de rutas) Sean* $R_1 = (w_1, t_1) \cdots (w_s, t_s)$ *y* $R_2 = (u_1, h_1) \cdots (u_p, h_p)$ *dos rutas sobre un mismo grafo, tales que* $w_s = u_1$*, definimos su concatenación como la ruta que recorre* R_2 *a continuación de* R_1*, fusionando el vértice común para que sea uno solo, con la suma de tiempos:*

$$R_1 \circ R_2 = (w_1, t_1) \cdots (w_{s-1}, t_{s-1})(u_1, t_s + h_1)(u_2, h_2) \cdots (u_p, h_p)$$

De esta forma, la ruta hija de R_1 y R_2 puede quedar definida como sigue:

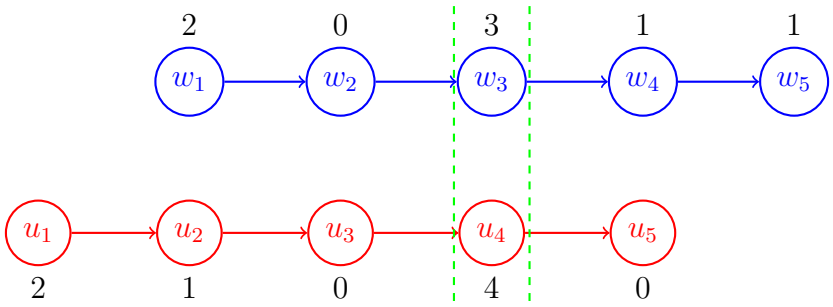
$$(w_1, t_1) \cdots (w_i, t_i) \circ \overline{R} \circ (u_j, h_j) \cdots (u_{s_2}, h_{s_2})$$

donde $\overline{R}$ es la ruta que recorre el camino óptimo de w_i hasta u_j (que tenemos calculado previamente aplicando el algoritmo de Floyd sobre el grafo) esperando el tiempo necesario en el último vértice de la misma.

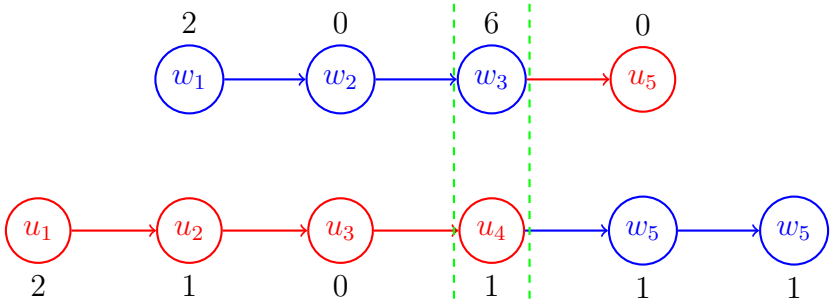
Primer método de cruce: `cruce`$_2$**.** La otra modalidad de cruce de dos rutas que tomaremos consiste en buscar un vértice por el que ambas rutas pasen a la misma hora, y realizar el cambio en ese punto.

Ejemplo 6.3

En el siguiente ejemplo ilustramos los sucesores que surgen a partir de dos rutas con el segundo modo de cruce, donde asumimos que la distancia de cada arista es 1.



Ambas rutas coinciden en el vértice $w_3 = u_4$, por lo tanto en este punto cruzaremos a los hijos. La ruta R_1 está en dicho vértice durante el intervalo $[4, 7]$, mientras que la ruta R_2 está durante $[6, 10]$. Por lo tanto, uno de los hijos llegará allí a las 4 y no saldrá hasta las 10, mientras que el otro llegará a las 6 y se irá a las 7. Los hijos obtenidos son:



Para definir este cruce formalmente, suponemos que existen índices $i \in \{1, \ldots, s_1\}$ y $j \in \{1, \ldots, s_2\}$ tales que $w_i = u_j$ y $[first_R(i), last_R(i)] \cap [first_{R'}(j), last_{R'}(j)] \neq \varnothing$. Entonces los sucesores de R_1 y R_2 quedan definidos como sigue:

$$(w_1, t_1) \cdots (w_{i-1}, t_{i-1})(w_i, last_{R'}(j) - first_R(i))(u_{j+1}, h_{j+1}) \cdots (u_{s_2}, h_{s_2})$$

$$(u_1, h_1) \cdots (u_{j-1}, h_{j-1})(u_j, last_R(i) - first_{R'}(j))(w_{i+1}, t_{i+1}) \cdots (w_{s_1}, t_{s_1})$$

Cabe esperar que esta condición sea bastante frecuente en grafos pequeños pero infrecuente en instancias con grafos grandes, por lo que finalmente hacemos un algoritmo que intente aplicar este segundo cruce, y en caso de no ser posible ejecute el primero.

- **Mutación**: Para realizar las mutaciones, contamos con un parámetro n_{mut} que representa el número máximo de vértices de una ruta que cambian tras una mutación. Si la ruta es $(w_1, t_1) \cdots (w_s, t_s)$, el algoritmo selecciona aleatoriamente dos índices $i, j \in \{1, \ldots, s + n_{mut}\}$ de tal forma que $i < j$ y $j - i \leq n_{mut}$, y pueden darse dos casos:

– Si $j < s$, el algoritmo genera una ruta aleatoria $(u_1, h_1) \cdots (u_{s'}, h_{s'})$ que va desde el vértice w_i hasta el vértice w_j en tiempo exactamente $last_R(j) - first_R(i)$, utilizando el mismo método que la función población inicial, devolviendo por resultado la ruta mutada que simplemente sustituye los vértices $w_i \cdots w_j$ por los nuevos vértices aleatorios:

$$(w_1, t_1) \cdots (w_i, t_i) \circ (u_1, h_1) \cdots (u_{s'}, h_{s'}) \circ (w_j, t_j) \cdots (w_s, t_s)$$

– Si $j \geq s$, entonces el algoritmo cambia el comienzo y el final de la ruta, generando una ruta $(u_1, h_1) \cdots (u_{s_1}, h_{s_1})$ que va desde un vértice origen aleatorio hasta $w_{j \mod s}$ para sustituir por el comienzo de R, y genera la ruta $(u'_1, h'_1) \cdots (u'_{s_2}, h'_{s_2})$ que va desde el vértice w_i hasta un vértice final aleatorio, para reemplazar el final R. El resultado es:

$$(u_1, h_1) \cdots (u_{s_1}, h_{s_1}) \circ (w_j, t_j) \cdots (w_i, t_i) \circ (u'_1, h'_1) \cdots (u'_{s_2}, h'_{s_2})$$

El motivo por el que tomamos esta función de mutación, dividiéndola en dos casos, es para que todos los vértices tengan igual probabilidad de ser mutados. Si cambiáramos tan solo intervalos de la forma $w_i w_{i+1} \cdots w_j$ con $i < j$, los vértices cercanos a los extremos de la ruta tienen menor probabilidad de ser modificados.

6.3 Resultados Obtenidos

En esta sección del capítulo comentamos los resultados obtenidos en la práctica al ejecutar los distintos algoritmos sobre instancias del problema generadas aleatoriamente. Comenzamos probando los algoritmos en instancias pequeñas del problema *rest-WPO(1)*, para las que podemos calcular la solución óptima, así como en instancias basadas en el último contraejemplo de los propuestos en la anterior sección, para comprobar como efectivamente los algoritmos voraces fallan en algunos casos concretos.

Posteriormente, como en soluciones más grandes no podemos calcular la solución óptima del problema, emplearemos el método estadístico de Kruskal-Wallis para comparar los algoritmos entre sí. Todas estas pruebas se realizan ejecutando cada algoritmo una única vez por instancia, en numerosas instancias distintas, para extraer conclusiones sobre el comportamiento de forma general en cualquier instancia del problema.

En el tercer y último apartado de la sección, realizamos varias ejecuciones de los mismos algoritmos genéticos sobre instancias de *WPO(1)* (en este caso las voraces no aplican) para estudiar la estabilidad que presentan estos algoritmos al repetirse sobre una misma instancia.

6.3.1 Comparativa con la Solución Óptima

Comenzamos probando todos los algoritmos sobre 100 instancias pequeñas del problema *rest-WPO(1)*, con hasta 100 testigos, en grafos de 8 vértices o menos, y con una duración de

10 unidades temporales, es decir, todos los testimonios $(a, L, [a, b])$ cumplen $a, b \leq 10$. Como estas instancias son pequeñas, podemos calcular su solución óptima para comparar con ella. Si un algoritmo $\mathtt{A}$ devuelve las soluciones $s_1, \ldots, s_{100}$, y las soluciones óptimas para dichas instancias son $\overline{s}_1, \ldots, \overline{s}_{100}$, los ratios obtenidos para cada instancia son:

$$r_i = R_{\mathcal{P}}(\mathtt{x}_i, \mathtt{A}(\mathtt{x}_i)) = \frac{\overline{s}_i}{s_i} \qquad \forall i \in \{1, \ldots, 100\}$$

Sobre estos ratios, calculamos para cada algoritmo su máximo, mínimo, media y varianza, recogidos en las siguientes tablas, donde además incluimos el número de veces que el algoritmo ha sido capaz de alcanzar la solución óptima. En la primera tabla aparecen todos los algoritmos voraces, y en la segunda las cuatro combinaciones de algoritmos genéticos, mezclando las funciones $\mathtt{ini}_1, \mathtt{ini}_2, \mathtt{cruce}_1$ y $\mathtt{cruce}_2$, ejecutadas con una población de tamaño 250 y realizando 1000 iteraciones.

	Cantidad soluciones óptimas	Máximo	Mínimo	Media	Varianza
$\mathtt{A}_1$	26	2.307	1.0	1.291	0.0898
$\mathtt{A}_2$	33	2.6	1.0	1.200	0.0886
$\mathtt{A}_3$	29	2.064	1.0	1.276	0.0794
$\mathtt{A}_4$	24	5.2	1.0	1.645	0.6886
$\mathtt{A}'_1$	54	1.615	1.0	1.061	0.0095
$\mathtt{A}'_2$	59	1.294	1.0	1.041	0.0044
$\mathtt{A}'_3$	61	1.304	1.0	1.053	0.0076
$\mathtt{A}'_4$	43	2.304	1.0	1.281	0.0925

Algoritmos Genéticos		Cantidad soluciones óptimas	Mínimo	Máximo	Media	Varianza
$\mathtt{ini}_1$	$\mathtt{cruce}_1$	48	1.0	1.333	1.089	0.0105
	$\mathtt{cruce}_2$	46	1.0	1.363	1.098	0.0121
$\mathtt{ini}_2$	$\mathtt{cruce}_1$	72	1.0	1.177	1.019	0.0014
	$\mathtt{cruce}_2$	71	1.0	1.162	1.018	0.0012

Vemos cómo entre los algoritmos voraces, son los algoritmos persistentes $\mathtt{A}_1, \mathtt{A}_2$ y $\mathtt{A}_3$ los que mejores resultados obtienen, alcanzando un ratio promedio muy cercano a 1, y con una varianza pequeña, por lo que para estas instancias pequeñas han demostrado ser bastante eficaces. Debemos notar que, aunque el grafo de la instancia sea pequeño, el número de testimonios es 100, por lo que las diferencias entre distintas soluciones sí que resultan apreciables, como podemos ver en la ineficacia obtenida por los otros algoritmos.
En particular, para nuestra sorpresa, el algoritmo $\mathtt{A}_4$, que es el que más prometía desde un punto de vista teórico porque el contraejemplo que conseguíamos para él era menos malo que los otros contraejemplos, es el que peores resultados está dando, así como su versión persistente también da malos resultados en comparación con el resto de algoritmos persistentes.

Respecto a los algoritmos genéticos, observamos que al ser iniciados con instancias aleatorias ($\texttt{ini}_1$) los resultados obtenidos son algo peores que los de los algoritmos voraces, pero al iniciarlos con las soluciones de los algoritmos voraces ($\texttt{ini}_2$) mejoran consistentemente su eficacia, batiendo, como es de esperar, a los voraces. No se percibe una diferencia significativa entre el uso de $\texttt{cruce}_1$ y $\texttt{cruce}_2$.

A continuación ejecutamos estos mismos algoritmos sobre un conjunto de instancias creadas a propósito para que los algoritmos voraces funcionen mal, inspiradas en el último contraejemplo del capítulo anterior, variando para diferentes valores del parámetro n. Los algoritmos voraces siempre encuentran una solución de calidad fija, independientemente del tamaño de la instancia original:

	$\texttt{A}_1$	$\texttt{A}_2$	$\texttt{A}_3$	$\texttt{A}_4$	$\texttt{A}'_1$	$\texttt{A}'_2$	$\texttt{A}'_3$	$\texttt{A}'_4$
Solución	3	2	2	1	3	3	3	1

Y los algoritmos genéticos obtienen lo siguiente:

Algoritmos Genéticos		Mínimo	Máximo	Media	Varianza
$\texttt{ini}_1$	$\texttt{cruce}_1$	1.166	3.75	1.84	0.650
	$\texttt{cruce}_2$	1.122	4.0	2.033	0.978
$\texttt{ini}_2$	$\texttt{cruce}_1$	1.285	3.2	2.085	0.430
	$\texttt{cruce}_2$	1.5	3.75	2.193	0.648

n:		5	6	7	8	9	10	11	12	13	14
$\texttt{ini}_1$	$\texttt{cruce}_1$	5	4	7	8	9	5	5	12	4	12
	$\texttt{cruce}_2$	5	6	6	7	9	4	9	4	10	4
$\texttt{ini}_2$	$\texttt{cruce}_1$	5	6	7	4	7	5	6	7	5	5
	$\texttt{cruce}_2$	5	5	6	8	6	6	5	5	4	5
óptima:		7	8	9	10	11	12	13	14	15	16

En cada casilla de esta última tabla se muestra el número de testimonios ciertos en la ruta devuelta por el algoritmo. Observamos como en este caso los algoritmos genéticos funcionan mejor que los voraces, pero también empeoran su rendimiento en comparación con las instancias aleatorias. Además, como cabe esperar, los genéticos que utilizan la función $\texttt{ini}_2$, es decir, que toman las soluciones voraces como parte de la población inicial, se ven bastante influenciadas por estas y resultan menos eficaces que las que comienzan en rutas aleatorias.

6.3.2 Test de Kruskal-Wallis

A continuación queremos comparar los algoritmos sobre instancias de mayor tamaño, pero en este caso no podemos calcular las soluciones óptimas eficientemente por ser un problema

intratable. Para ello, vamos a recurrir al método estadístico de Kruskal-Wallis que resumimos brevemente a continuación, con la información extraida de [14].

Este método pretende comprobar la veracidad de la hipótesis nula H_0, que postula que las medianas de las distintas muestras (resultados de los algoritmos) son iguales, y la hipótesis alternativa consiste en que al menos dos algoritmos tienen medianas distintas. El método, al aplicarse sobre nuestros doce algoritmos, comienza obteniendo un ranking para cada instancia $\mathtt{x}_j$, asignando a cada algoritmo un número del 1 al 12 de tal forma que queden ordenados de menos a más eficaz, denotando por r_{ij} el rango del algoritmo i en la instancia x_j. Posteriormente el método calcula el ranking promedio para cada algoritmo (siendo N el número de instancias):

$$\overline{r}_i = \frac{1}{N}\sum_{j=1}^{N} r_{ij}$$

y el ranking promedio entre todos los algoritmos es $\overline{r} = \frac{1}{2}(N+1)$. Finalmente, y sin entrar en más detalle, evalúa la expresión

$$H = (N-1)\frac{\sum_{i=1}^{12} N(\overline{r}_i - \overline{r})^2}{\sum_{i=1}^{12}\sum_{j=1}^{N}(r_{ij} - \overline{r})^2}$$

La distribución de H puede ser aproximada por una distribución chi-cuadrado con $12-1$ grados de libertad, y entonces el algoritmo utiliza el valor H obtenido experimentalmente para determinar si $P(x \leq H)$ es menor que un parámetro α de tolerancia dado como entrada.

Hemos aplicado este método utilizando una calculadora online [15], sobre un conjunto de 100 instancias medianas del problema *rest-WPO(1)*, con hasta 250 testigos, duración del día de 100 unidades de tiempo, y con grafos de hasta 20 vértices, y posteriormente, con otro conjunto de 50 instancias con hasta 500 testigos, días de 200 unidades temporales y grafos de 100 vértices. Los resultados para ambos grupos de instancias quedan recogidos en las siguientes tablas:

	$\mathtt{A}_1$	$\mathtt{A}_2$	$\mathtt{A}_3$	$\mathtt{A}_4$	$\mathtt{A}'_1$	$\mathtt{A}'_2$	$\mathtt{A}'_3$	$\mathtt{A}'_4$
Mediana	50	50	50.5	23.5	56	56.5	59.5	30
Suma de rangos	58277	60335	61282	39101	62494	64126.5	64666	43832.5

	Genéticos $\mathtt{ini}_1$		Genéticos $\mathtt{ini}_2$	
	$\mathtt{cruce}_1$	$\mathtt{cruce}_2$	$\mathtt{cruce}_1$	$\mathtt{cruce}_2$
Mediana	62.5	60	64.5	64.5
Suma de rangos	65738.5	65009.5	67969	67769

El valor de H obtenido para el primer grupo de instancias es 77.4252, y el valor de $P(x \leq H)$ es prácticamente 1, de lo que podemos concluir que hay alguno de los algoritmos cuya mediana es significativamente distinta, estadísticamente hablando.

	G_1^1	G_1^1	G_1^1	A_1	A_2	A_3	A_4	A'_1	A'_2	A'_3	A'_4
G_1^1	7.29	-22.31	-20.31	74.62	54.03	44.56	266.38	32.44	16.12	10.73	219.06
G_2^1	0	-29.6	-27.6	67.33	46.75	37.27	259.09	25.15	8.83	3.44	211.77
G_1^2	-29.6	0	2	96.92	76.34	66.87	288.68	54.75	38.43	33.03	241.37
G_2^2	-27.6	2	0	94.92	74.34	64.87	286.68	52.75	36.43	31.03	239.37
A_1	67.33	96.92	94.92	0	-20.58	-30.05	191.76	-42.17	-58.5	-63.89	144.44
A_2	46.75	76.34	74.34	-20.58	0	-9.47	212.34	-21.59	-37.91	-43.31	165.03
A_3	37.27	66.87	64.87	-30.05	-9.47	0	221.81	-12.12	-28.44	-33.84	174.5
A_4	259.09	288.68	286.68	191.76	212.34	221.81	0	-233.93	-250.25	-255.65	-47.31
A'_1	25.15	54.75	52.75	-42.17	-21.59	-12.12	-233.93	0	-16.32	-21.72	186.62
A'_2	8.83	38.43	36.43	-58.5	-37.91	-28.44	-250.25	-16.32	0	-5.39	202.94
A'_3	3.44	33.03	31.03	-63.89	-43.31	-33.84	-255.65	-21.72	-5.39	0	208.33

Esta tabla muestra la resta entre el ranking promedio de cada par de algoritmos, representando en rojo las parejas de algoritmos con más diferencia entre ellas. Denotamos, por ahorrar espacio, G_j^i al algoritmo genético que utiliza $\texttt{ini}_i$ y $\texttt{cruce}_j$.

	A_1	A_2	A_3	A_4	A'_1	A'_2	A'_3	A'_4
Mediana	5	6	6	3.5	10	10	10	6
Suma de rangos	49623.5	53064.5	53435.5	36104	70427.5	70721	71120.5	54105.5

	Genéticos $\texttt{ini}_1$		Genéticos $\texttt{ini}_2$	
	$\texttt{cruce}_1$	$\texttt{cruce}_2$	$\texttt{cruce}_1$	$\texttt{cruce}_2$
Rank sum (R):	6	7	10	10
Median:	59422	59544	71516	71516

El valor de H obtenido es 118.4878, y la probabilidad $P(x \leq H)$ es prácticamente 1, por lo tanto existen diferencias significativas y se puede descartar la hipótesis H_0.

	G_2^1	G_1^2	G_2^2	A_1	A_2	A_3	A_4	A'_1	A'_2	A'_3	A'_4
G_1^1	-1.22	-120.94	-120.94	97.99	63.58	59.87	233.18	-110.05	-112.99	-116.99	53.17
G_2^1	0	-119.72	-119.72	99.21	64.8	61.09	234.4	-108.83	-111.77	-115.76	54.39
G_1^2	-119.72	0	0	218.92	184.51	180.8	354.12	10.88	7.95	3.95	174.11
G_2^2	-119.72	0	0	218.92	184.51	180.8	354.12	10.88	7.95	3.95	174.11
A_1	99.21	218.92	218.92	0	-34.41	-38.12	135.19	-208.04	-210.98	-214.97	-44.82
A_2	64.8	184.51	184.51	-34.41	0	-3.71	169.6	-173.63	-176.57	-180.56	-10.41
A_3	61.09	180.8	180.8	-38.12	-3.71	0	173.31	-169.92	-172.86	-176.85	-6.7
A_4	234.4	354.12	354.12	135.19	169.6	173.31	0	-343.23	-346.17	-350.17	-180.01
A'_1	-108.83	10.88	10.88	-208.04	-173.63	-169.92	-343.23	0	-2.94	-6.93	163.22
A'_2	-111.77	7.95	7.95	-210.98	-176.57	-172.86	-346.17	-2.94	0	-4	166.16
A'_3	-115.76	3.95	3.95	-214.97	-180.56	-176.85	-350.17	-6.93	-4	0	170.15

Podemos concluir que el algoritmo voraz $\mathtt{A}_4$ es con diferencia el menos efectivo, siendo el que peores resultados da en todas las muestras realizadas. Es seguido del algoritmo persistente correspondiente $\mathtt{A}'_4$, que desde las instancias pequeñas y medianas también daba malos resultados en comparación con el resto, y los algoritmos voraces no persistentes, cuyo resultado empeora y se desliga más del resto sobre todo en estas últimas instancias más grandes.

Respecto a los algoritmos genéticos, percibimos como su desempeño, al menos con estos parámetros, empeora notablemente al incrementar el tamaño de las instancias. En concreto, sospechamos que se debe al mayor tamaño del grafo, que aumenta mucho las rutas posibles y dificulta al algoritmo genético generar aleatoriamente rutas prometedoras. De esta forma, así como en las instancias medianas con grafos de hasta 20 vértices los algoritmos genéticos destacaban sobre los voraces (incluso los iniciados con $\mathtt{ini}_1$), en las instancias más grandes, los que utilizan $\mathtt{ini}_1$ se quedan algo atrás en comparación con los mejores voraces, y los que utilizan las soluciones voraces como parte de su población inicial consiguen una leve mejoría respecto a estos, casi inapreciable.

6.3.3 Comparativa de Parámetros Genéticos

Finalmente, en esta sección comparamos distintas combinaciones de parámetros en los algoritmos genéticos, modificando la probabilidad de mutación, el tamaño de la población y el número de iteraciones. Utilizaremos 10 instancias aleatorias del problema *WPO(1)*, con grafos de hasta 50 vértices, duración del día de 100 unidades de tiempo y hasta 5 testimonios por testigo, siendo alrededor de un 20% de ellos testimonios negados. Como en este caso no tratamos con instancias del problema *rest-WPO(1)*, no podemos aplicar la función $\mathtt{ini}_2$ de población inicial.

Mostramos en la siguiente tabla la media y varianza obtenidas tras repetir cada versión del algoritmo genético 10 veces sobre cada instancia, donde las diferencias entre las distintas versiones probadas son el número de iteraciones n, el tamaño de la población k, la probabilidad de mutación p y el método de cruce utilizado:

			media	varianza	media	varianza	media	varianza	media	varianza	media	varianza
		Instancia:	1		2		3		4		5	
$\texttt{cruce}_1$	$p = 0.1$	n=1000, k=250	8.8	0.56	6	0.2	4	0	13.3	1.01	5.9	0.49
		n=500, k=500	8.9	0.48	5.3	0.21	4.4	0.24	14	2.4	5.3	0.21
		n=250, k=1000	7.9	0.89	4.8	0.36	4	0	12.4	1.03	4.5	0.25
	$p = 0.2$	n=1000, k=250	9.21	0.76	5.5	0.25	4.1	0.08	14.5	2.45	6.2	0.36
		n=500, k=500	9.2	0.29	5.9	0.26	4.2	1.01	14.7	0.76	6.2	11.4
		n=250, k=1000	8.2	0.36	5.1	0.09	4.1	0.09	12.2	0.96	5.3	0.41
$\texttt{cruce}_2$	$p = 0.1$	n=1000, k=250	8.7	0.61	5.1	0.089	4.3	0.21	12.4	2.44	6	0.2
		n=500, k=500	9.1	0.89	5.2	0.16	4.3	0.21	12.9	2.89	5.9	0.29
		n=250, k=1000	8	1	4.8	0.36	4	0	12.4	2.44	5.4	0.44
	$p = 0.2$	n=1000, k=250	9.2	0.56	4.9	0.09	4.1	0.09	12.5	0.65	5.9	0.29
		n=500, k=500	9.5	0.45	5.3	0.41	4.2	0.16	13.9	2.89	6.3	0.81
		n=250, k=1000	8.1	1.29	4.8	5.56	4	0	10.5	1.45	5.4	0.24

		Instancia:	6		7		8		9		10	
$\texttt{cruce}_1$	$p = 0.1$	n=1000, k=250	67.4	7.83	7.2	0.16	7.1	1.01	11.3	0.69	23.6	0.44
		n=500, k=500	64.2	5.76	8.1	0.48	7	0	11.3	0.61	22.6	0.84
		n=250, k=1000	62.3	2.81	7.4	1.44	7	0	9.3	1.2	22.1	1.08
	$p = 0.2$	n=1000, k=250	65.8	7.35	7.8	0.76	7	0	11.4	0.83	23.8	0.56
		n=500, k=500	67.6	0.49	8.1	0	7	0.48	11.1	0.09	23.2	0.15
		n=250, k=1000	63.9	12.2	7	0.4	7	0	10.3	0.41	23.3	1
$\texttt{cruce}_2$	$p = 0.1$	n=1000, k=250	65.5	27.2	7.7	0.41	7	0	10.9	0.48	23.4	0.63
		n=500, k=500	64.7	1.61	7.7	0.21	7	0	10.7	0.61	23.2	0.76
		n=250, k=1000	61.8	0.55	7.3	1.01	7	0	9.7	1.21	21.9	0.68
	$p = 0.2$	n=1000, k=250	64.9	3.89	8	0.8	7	0	11	1.4	23.3	0.21
		n=500, k=500	67.1	10.89	8.3	0.81	7	0.09	11.3	0.24	23.3	0.21
		n=250, k=1000	64	11.8	7.3	0.41	7	0	9.9	0.69	23	1

Hay varias comparaciones interesantes que pueden surgir de esta tabla. En primer lugar, si comparamos las diferencias entre $\texttt{cruce}_1$ y $\texttt{cruce}_2$, el primero de los cruces parece obtener resultados ligeramente mejores, pues suele alcanzar medias algo superiores a las que logra el otro algoritmo, pero esta diferencia es insuficiente como para concluir que un método sea estrictamente superior al otro.

Respecto a la probabilidad de mutación, observamos que en la mayoría de instancias se obtienen mejores soluciones medias al aumentar la probabilidad a $p = 0.2$, y además, sorprendentemente, se reduce el valor de la varianza también con este parámetro, lo que indica que las soluciones obtenidas están más cercanas a la media, habiendo menos variabilidad

entre una ejecución y otra, lo que hace al algoritmo más consistente. Esto parece sugerir que el algoritmo termina convergiendo a máximos locales, poblaciones donde todos los individuos de la población obtienen resultados parecidos, y sus correspondientes cruces y mutaciones no mejoran este valor, de los cuales es más fácil escapar al tener una probabilidad de mutación más elevada.

Finalmente, respecto a la configuración escogida para el algoritmo (número de iteraciones n y tamaño de población k), observamos cómo un número pequeño de iteraciones $n = 250$, pese a tener un gran tamaño de población, $k = 1000$, obtiene resultados inferiores a las otras dos configuraciones probadas. Sin embargo, no parece existir una diferencia significativa en las soluciones obtenidas por el algoritmo con $n = 1000$, $k = 250$, ni por el algoritmo con $n = k = 500$. En ciertas instancias uno es superior al otro, pero en otras instancias sucede a la inversa, sin ser lo sufucientemente notables estas diferencias como para poder decantarnos por uno o por el otro.

Conclusión

En este trabajo hemos definido un modelo formal que permite expresar testimonios que dan la ubicación de un grupo de personas durante un cierto intervalo de tiempo, para posteriormente poder realizar razonamientos sobre la veracidad o incompatibilidad de varios de estos testimonios. Hemos realizado un estudio pormenorizado de diferentes variantes del problema de decidir si todos los testimonios son compatibles o no, para ver cuáles son los elementos concretos de la definición que implican la completitud en la clase NP, y bajo que restricciones se puede resolver en tiempo polinómico, observando como sutiles diferencias en la formulación del problema suponen grandes saltos en su complejidad.

Posteriormente, hemos analizado el problema de maximizar el número de testigos que dicen simultáneamente la verdad, centrándonos en particular en una versión muy restringida del problema, consiguiendo su APX-dureza, y en un intento de demostrar la pertenencia a APX ha quedado reflejada una dificultad intrínseca en el problema, ya que diversos algoritmos voraces de aproximación propuestos no consiguen alcanzar la cota buscada en el ratio de aproximación. Finalmente hemos resuelto en la práctica la versión del problema con un sólo actor mediante un algoritmo genético, comparando sus resultados con los algoritmos voraces, arrojando conclusiones sobre la efectividad de estos voraces en la práctica pese a su mal comportamiento teórico, ya que superan a los genéticos.

Cabe mencionar que este trabajo de fin de grado se ve complementado con el correspondiente trabajo realizado en el grado de matemáticas, donde analizamos además la aproximabilidad de otras variantes del problema, obteniendo que la versión general no se puede aproximar con ratio mejor que 0, y que la versión con un solo testimonio por testigo es completa en la clase $Poly\text{-}APX$.

Los siguientes pasos para ampliar el trabajo, que pueden dar pie a líneas de investigación futuras, serían estudiar la cuestión que queda abierta en este texto, sobre si la versión restringida pertenece a APX, o si es dura en alguna clase de aproximabilidad superior. Asimismo, a raíz del estudio detallado del problema de decisión, se pueden formular muchas variantes distintas del problema, relajando alguna de las restricciones y buscando algoritmos polinómicos que sigan resolviéndolo.

Conclusion

In this work we have defined a formal model that allows us to formulate testimonies that describe the location of a group of people during a certain time interval, in order to be able to reason about the veracity or inconsistency of a number of them. We have undertaken a in-depth study of different variants of the problem of deciding whether all the testimonies are compatible or not, in order to see which are the specific elements of the definition that imply *NP*-completeness, and under which restrictions the problem can be solved in polynomial time, observing the way in which subtle differences in the statement of the problem suppose great leaps in its complexity.

We have subsequently discussed the problem of maximizing the number of witnesses who can be believed simultaneously, focusing in particular on a very restricted version of the problem, prooving its APX-hardness. In an attempt to prove its membership in APX, an apparent underlying difficulty has been revealed, since several proposed greedy approximation algorithms fail to achieve the desired ratio. Finally, we have developed a genetic algorithm for the single-actor version of the problem, comparing its results with those of the greedy algorithms and drawing conclusions on the great performance of these greedy algorithms over genetic ones, despite their theoretically poor behaviour.

It is interesting to remark that this bachelor's thesis is complemented with the respective one carried on for the degree in mathematics, where we study the approximability of different variations of the problem, obtaining that the more general version cannot be approximated with a ratio better than 0, and that the version with a single testimony per witness is complete in *Poly-APX*.

Further steps to extending the present work, which may give rise to future lines of research, would be to study the question left open in this text, as to whether the restricted version belongs to APX, or is hard in some higher approximability class. Likewise, as a result of the detailed study on the *NP*-completeness of the decision problem, many different variations can be formulated, by relaxing some of the constraints and looking for polynomial algorithms that still solve the problem.

Bibliografía

[1] Patricia L. Brantingham. Computational criminology. In *2011 European Intelligence and Security Informatics Conference*, pages 3–3, 2011.

[2] Richard Berk. Algorithmic criminology. 2013.

[3] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[4] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Science & Business Media, 2012.

[5] Pierluigi Crescenzi. A short guide to approximation preserving reductions. In *Proceedings of Computational Complexity. Twelfth Annual IEEE Conference*, pages 262–273. IEEE, 1997.

[6] Bruno Escoffier and Vangelis Th Paschos. A survey on the structure of approximation classes. *Computer Science Review*, 4(1):19–40, 2010.

[7] Wikipedia contributors. Hamiltonian path problem — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Hamiltonian_path_problem&oldid=1053285848, 2021.

[8] Romeo Rizzi, Alexandru I Tomescu, and Veli Mäkinen. On the complexity of minimum path cover with subpath constraints for multi-assembly. In *BMC bioinformatics*, volume 15, pages 1–11. BioMed Central, 2014.

[9] Rıza Özçelik. Solving minimum path cover on a dag. https://towardsdatascience.com/solving-minimum-path-cover-on-a-dag-21b16ca11ac0, 2019.

[10] Delbert R Fulkerson. Note on dilworth's decomposition theorem for partially ordered sets. In *Proc. Amer. Math. Soc*, volume 7, pages 701–702, 1956.

[11] Bengt Aspvall, Michael F Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information processing letters*, 8(3):121–123, 1979.

[12] Wikipedia contributors. Genetic algorithm — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Genetic_algorithm&oldid=1081816519, 2022. [Online; accessed 9-May-2022].

[13] Genetic algorithms - crossover and mutation operators for paths. https://localcoder.org/genetic-algorithms-crossover-and-mutation-operators-for-paths, 2022.

[14] Wikipedia contributors. Kruskal–wallis one-way analysis of variance — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Kruskal%E2%80%93Wallis_one-way_analysis_of_variance&oldid=1080714637, 2022. [Online; accessed 25-May-2022].

[15] Kruskal wallis test calculator, statistics kingdom. https://www.statskingdom.com/kruskal-wallis-calculator.html.