Cómo frenar una propagación: Análisis de complejidad y resolución práctica

Por
Javier Galiana Ruiz de la Hermosa

UNIVERSIDAD COMPLUTENSE MADRID

Doble Grado en Ingeniería Informática y Matemáticas
Facultad de Informática

Dirigido por
Ismael Rodríguez Laguna, Fernando Rubio Diez

How to stop a propagation: Complexity analysis and implementation.

Madrid, 2020–2021

Cómo frenar una propagación: Análisis de complejidad y resolución práctica

Memoria que se presenta para el Trabajo de Fin de Grado

Javier Galiana Ruiz de la Hermosa

Dirigido por

Ismael Rodríguez Laguna, Fernando Rubio Diez

Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

Madrid, 2021

A todos aquellos que nunca dejaron de creer en mí.
En especial, a mis padres.

Resumen

Las propagaciones en todas sus variantes constituyen una parte fundamental de la existencia humana. Las enfermedades, un virus informático o incluso la propagación de los impulsos nerviosos suponen ejemplos muy variados que reflejan la importancia del estudio de las mismas, para entender su naturaleza y, en muchos casos, cómo erradicarlas. Este problema ha concentrado el esfuerzo de muchos profesionales de todos los ámbitos, como la medicina o la psicología. Otros estudios previos han investigado cómo las propagaciones avanzan por un entorno [1], llevando a cabo un acercamiento al problema desde un punto de vista mucho más estático, pues no adaptan su estrategia para contener la propagación de manera dinámica como nosotros pretendemos. A pesar de esto, los modelos que se basan en predecir la forma en la que una propagación avanza suelen fallar debido a la gran cantidad de variables en consideración y al desconocimiento inicial cuando una propagación emerge.

Este estudio nos permitirá adentrarnos de una forma genérica, teórica y práctica en el funcionamiento de las propagaciones; qué podemos hacer contra ellas y cuál es la dificultad de frenarlas. A lo largo de este escrito profundizaremos en las propagaciones desde una perspectiva distinta, la teoría de complejidad. Probaremos que el problema de encontrar una estrategia óptima para lograr frenar una propagación es `PSPACE-completo` bajo determinadas condiciones.

La intratabilidad de este problema hace necesario el uso de métodos heurísticos para solucionarlo de manera subóptima, en nuestro caso, un algoritmo minimax con poda alfa-beta. De esta manera, crearemos un modelo desde cero que nos servirá para representar el arranque de una propagación y la inmediata intervención del apaciguador, figura que simulará la acción humana en el intento de frenar la misma. Nos apoyaremos en ejemplos representativos que nos permitirán analizar las fortalezas y debilidades del modelo y nos ayudarán a entender de manera sencilla el funcionamiento del algoritmo minimax aplicado a las propagaciones.

Asimismo, este trabajo tiene la finalidad de suponer una mejora social en determinados sectores pues tiene aplicabilidad real, brindando una herramienta que permita adelantarse a aquellas complicaciones que podría generar cualquier tipo de propagación. Para ello, presentaremos un caso de estudio donde basamos nuestro problema en datos reales, extraídos de la propagación de la COVID-19 en España, como son el Presupuesto de Sanidad Pública para cada una de las Comunidades Autónomas o el porcentaje de Movilidad interterritorial en Marzo de 2020.

Palabras clave: *propagación*, *apaciguador*, *complejidad*, *NP-duro*, *PSPACE-completitud*, *algoritmo minimax*, *poda alfa-beta*, *intermediación.*

Abstract

Propagations in all their variants are a fundamental part of human existence. Diseases, computer viruses, or even nerve impulses are very varied examples that reflect the importance of studying them to understand their nature and, in some cases, how to eradicate them. This problem has concentrated the efforts of many professionals from all fields, such as medicine or psychology. Other previous works have researched on how a propagation spreads through an environment [1], addressing the problem from a much more static perspective, as they do not adapt their strategy to stop the propagation dynamically as we intend. However, those models that are based on predicting how a propagation progresses often fail, the reason being the large number of variables under consideration and the initial lack of knowledge when a propagation emerges.

This study will allow us to enter into a general, theoretical, and practical approach about how propagations work; what can we do against them, and how complex is to stop it. Throughout this paper, we will deepen in propagations from a new approach, the complexity theory. We will prove that the problem of finding an optimal strategy for stopping a propagation is `PSPACE-complete` under some standard assumptions.

The intractability of this problem makes it necessary to use heuristic methods to solve it in a suboptimal way, in our case, a minimax algorithm with alpha-beta pruning. In this way, we will create a model from scratch that will serve to represent the beginning of propagation and the immediate intervention of the appeaser, a figure that will simulate the human action attempting to stop it. We will use representative examples that will allow us to analyse the strengths and weaknesses of the model and will help us to understand in a simple way how the minimax algorithm applied to propagations works.

Likewise, this work is intended to bring about a social improvement in certain sectors as it has real applicability, providing a tool that allows us to anticipate those complications that could be generated by any type of propagation. To this end, we will present a case study where we base our problem on real data, extracted from the COVID-19 propagation in Spain, such as the Public Health budget for each region or the percentage of inter-territorial mobility from March 2020.

Keywords: *propagation*, *appeaser*, *complexity*, *NP-hard*, *PSPACE-completeness*, *minimax algorithm*, *alpha-beta pruning*, *betweenness centrality*.

Índice general

Capítulo 1

Introducción

En esta introducción introduciremos en primer lugar la temática que vertebrará nuestro texto, junto con los motivos por los cuales nos hemos visto interesados en desarrollarla. Más adelante, explicaremos los fundamentos teóricos que nos permitirán entender sin mayor dificultad los principales resultados y procedimientos que emplearemos para demostrar y poner en práctica nuestro estudio acerca de las propagaciones.

1.1 Motivación

La naturaleza humana lleva intrínseca la necesidad de encontrar respuestas rápidas y eficientes a los problemas que nos rodean. Y todavía en mayor medida cuando estos problemas hacen que las bases de nuestra sociedad (la economía, la salud o la seguridad de la población) están en peligro. Por este motivo, estamos interesados en encontrar un tema que nos permita estudiar un problema aún por analizar y con el que pudiéramos impactar en la vida de las personas.

A medida que pasan los años, el cambio climático aumenta el riesgo de aparición de plagas en los cultivos. De igual manera, los incendios aumentan la deforestación en distintas partes del planeta a un ritmo vertiginoso y el incremento de la desinformación y las *fake news* condicionan el pensamiento crítico de los ciudadanos a la hora de tomar decisiones. Todos estos ejemplos representan situaciones cada vez más comunes que comparten una base no tangible: todos contienen un mecanismo de propagación implícito.

Las propagaciones tienen en muchos casos connotaciones negativas, representando algo contra lo que se debe luchar o protegerse, de ahí que su estudio sea importante para la propia existencia del ser humano. En consecuencia, trataremos que este estudio suponga un paso más en el análisis de las mismas (en particular, de las negativas que sería necesario frenar), intentando esclarecer su complejidad y brindando una implementación que permita su representación y estudio.

1.2 Problemas intratables

Comenzaremos presentando los conceptos básicos de la teoría de la complejidad: las clases `P` y `NP` de problemas resolubles en tiempo polinómico mediante maquinas de Turing

deterministas y no deterministas, respectivamente, y la técnica de reducción en tiempo polinómico, así como los conceptos de NP-completo y NP-duro, propiedades que tienen determinados problemas. Además, conoceremos la clase PSPACE y las nociones relacionadas con ella. Todos estos conceptos serán recurrentes a lo largo de nuestro estudio sobre el problema de la propagación[1].

La noción de algoritmo eficiente considera como tales a aquellos algoritmos que pueden ser ejecutados en un tiempo limitado superiormente por una cota polinómica que depende directamente del tamaño del problema, es decir, el número de datos que definen el mismo. A diferencia de estos, los algoritmos considerados como no eficientes son aquellos que requieren al menos un tiempo no polinómico para algunas de las instancias del problema. Consecuentemente, podemos hablar de algoritmos polinómicos o no polinómicos. El concepto de complejidad algorítmica está directamente relacionado con la eficiencia de la que hablamos.

Decimos que un problema pertenece a la clase P si puede ser resuelto mediante algoritmos polinómicos deterministas, mientras que un problema pertenece a NP si puede ser resuelto mediante algoritmos no deterministas polinómicos. La relación entre todos los problemas que pertenecen a alguna de estas clases viene dada por reducciones polinómicas entre ellos, es decir, por medio de una transformación de complejidad polinómica entre instancias de cada uno de los problemas que siempre garantiza el mismo resultado en ambos. En la práctica, consideraremos como problemas intratables computacionalmente a los problemas NP-duros. Si un problema satisface que todo problema perteneciente a NP es reducible polinómicamente a él diremos que el problema es NP-duro, independientemente de que pertenezca a NP o no. En consecuencia, llegamos a una definición crucial dentro de estos conceptos.

Definición 1.1. *Un problema p es* NP-completo *si está en* NP *y además es* NP-duro.

Un ejemplo de problema NP-completo es el problema del viajante. La importancia de esta clasificación y de la existencia de esta clase es que no ha sido posible hasta la fecha encontrar algoritmos polinomiales para ninguno de los problemas que pertenecen a la misma pero, a la vez, no ha sido posible demostrar la inexistencia de los mismos. De esta forma, el desarrollo de un algoritmo polinómico para cualquiera de ellos implicaría automáticamente que todos estos problemas estarían en P. La cuestión $\mathtt{P} \neq \mathtt{NP}$ está abierta, siendo necesario encontrar nuevas técnicas que permitan probarla.

Representamos todas estas clases de complejidad y resumimos nuestro conocimiento de las relaciones entre ellas en la Figura 1.1. Aún no sabemos si estas contenciones son, en realidad, igualdades, o si por contra la figura podría colapsar incluso en una única clase.

No se conoce ningún algoritmo polinómico que pueda encontrar una solución óptima para algún problema que sea NP-duro. Por este motivo, existen ciertas aproximaciones, más o menos satisfactorias, que pertenecen a alguna de las siguientes familias:

- Algoritmos exactos, que siempre encuentran una solución óptima pero tardan tiempo exponencial, como pueden ser ramificación y poda, programación dinámica, etc.

[1]Pueden estudiarse con mayor detenimiento en los libros *An overview on polynomial approximation of NP-hard problems* [2], *Introduction to the Theory of Computation* [3] e *Introducción a la teoría de autómatas, lenguajes y computación* [4].

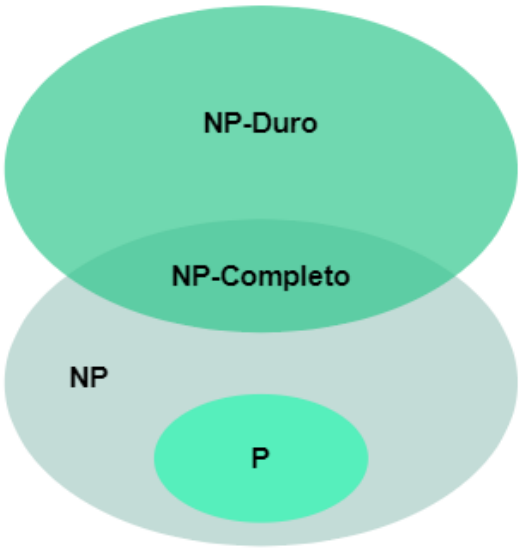


Figura 1.1: Relación de clases de complejidad [5].

- Algoritmos heurísticos, que encuentran una solución subóptima (en ocasiones, infactible) pero tardan menos tiempo, como pueden ser los algoritmos genéticos, los algoritmos de aproximación, etc.

De forma análoga, podemos trasladar todos estos conceptos de complejidad en función del espacio utilizado por el algoritmo.

Definición 1.2. *Si denotamos* $\mathsf{SPACE}(t(n))$ *al conjunto de todos los problemas que pueden ser resueltos usando* $O(t(n))$ *espacio para una función* t *de tamaño de entrada* n*, podemos definir* PSPACE *como el conjunto de lenguajes que son decidibles en espacio polinómico en una maquina de Turing determinista. Formalmente,*

$$\mathsf{PSPACE} = \cup_k \mathsf{SPACE}(n^k)$$

Siguiendo el mismo proceso que antes, los problemas PSPACE-completo son aquellos problemas de decisión en PSPACE tales que todo problema en PSPACE puede ser reducido a él en tiempo polinómico. Los problemas PSPACE-completos pueden verse como los problemas más difíciles de la clase PSPACE y se conjetura que estos problemas están fuera de las clases de complejidad P y NP, pero no hay prueba de ello. Si alguno de ellos estuviera en NP, entonces PSPACE = NP, o si alguno estuviera en P, entonces PSPACE = P. Ejemplos muy conocidos de problemas PSPACE son el problema TQBF y muchos de los problemas que consisten en hallar estrategias ganadoras para juegos. Dichos problemas serán formalmente introducidos y resultarán fundamentales posteriormente.

1.3 Teoría de grafos

A lo largo de todo el escrito, recurriremos en múltiples ocasiones a cierta terminología relacionada con los grafos, que constituirán la base del problema a desarrollar y, por lo tanto, es necesario repasar a continuación.

Dado un grafo $G = (V, A)$, donde V representa el conjunto de vértices y $A \in \mathcal{P}'(V \times V)$ el conjunto de aristas que los relacionan, donde $\mathcal{P}'(V \times V)$ denota el conjunto de todos los

multiconjuntos de elementos de $V \times V$, definimos ahora cierta notación que mantendremos a lo largo de la explicación de nuestro problema.

Definimos el *cardinal* de un grafo, $|G|$, como el número de vértices que posee V. De la misma forma, si hablamos del *cardinal* de un subconjunto de vértices $P \subseteq V$, haremos referencia al número de nodos que posee este subconjunto y lo denotaremos como $|P|$.

El *conjunto de adyacencia*, $Ady(v)$, del grafo G para un vértice $v \in V$ es el conjunto de vértices pertenecientes a V que son adyacentes a v, es decir, tales que existe una arista $a \in A$ tal que une ambos vértices.

Por último, llamaremos *complementario* de un subconjunto $P \subseteq V$ al conjunto que contiene a todos los vértices de V que no están en P y lo denotaremos como $\overline{P}$.

Terminamos así de establecer los conocimientos básicos que nos permitirán comprender el desarrollo del problema que, a lo largo de la próxima sección, quedará íntegramente definido.

Chapter 2

Introduction

In this introduction, we will first introduce the subject that will drive our text, along with the reasons why we have been interested in its study. Later on, we will explain the theoretical notions that will allow us to easily understand the main results and procedures that we will use for our complexity proof and for putting into practice our study about propagations.

2.1 Motivation

Human nature is inherent to the need for rapid and efficient responses to the problems surrounding us. This is especially true when these problems shake the foundations of our society (the economy, health, or security of citizens). For this reason, we are interested in finding a subject that would allow us to study a problem yet to be analysed and with which we could have an impact on people's lives.

As the years go by, climate change increases the risk of crop pests. Similarly, fires increase deforestation in different parts of the planet at a dizzying rate, and the increase in disinformation and fake news influence citizens' critical thinking when it comes to making decisions. All those examples represent increasingly common situations that share a non-tangible basis: they all contain an implicit propagation mechanism.

In many cases, propagations have negative connotations, representing something to be fought against or protected, so their study is aligned with the very existence of human beings. Consequently, we hope this study contributes to their analysis (in particular, the nevative ones that need to be stopped), clarifying their complexity and providing an implementation that allows their representation and study.

2.2 Intractable Problems

We will begin by introducing the basic concepts of complexity theory: the classes `P` and `NP` of problems solvable in polynomial time by deterministic and non-deterministic Turing machines, respectively, and polynomial reductions, as well as the concepts of `NP-complete` and `NP-hard`, properties that certain problems have. In addition, we will learn about the

class PSPACE and the notions related to it. All these concepts will recur throughout our study of the propagation problem[1].

The notion of efficient algorithms considers as such those algorithms that can be executed in a time bounded by a polynomial number that depends directly on the size of the problem, in other words, the number of data that define the problem. In contrast to these, algorithms considered as non-efficient are those that require at least a non-polynomial time for some of the problem instances. Consequently, we can speak about polynomial or non-polynomial algorithms. The concept of algorithmic complexity is directly related to the efficiency we are talking about.

We say that a problem belongs to the class P if it can be solved through deterministic polynomial algorithms, while a problem belongs to NP if it can be solved using non-deterministic polynomial algorithms. The relation between all the problems within each of these classes is given by polynomial reductions between them, that is, through a polynomial transformation between instances of each of the problems guaranteeing the same result in both of them. In practice, we will consider computationally intractable problems as computationally to those belonging to NP-hard. If a problem satisfies that every problem belonging to NP is polynomially reducible to it we will say that the problem is NP-hard, regardless of whether it belongs to NP or not. Consequently, we arrive at a crucial definition within these concepts.

Definición 2.1. *A problem p is* NP-complete *if it belongs to* NP *and it is also* NP-hard*.*

An example of a NP-complete problem is the Travelling Salesman Problem. The importance of this classification is that it has not been possible to date to find polynomial algorithms for any of the problems that belong to this class but, at the same time, it has not been possible to prove their non-existence. Thus, the development of a polynomial algorithm for any of them would automatically imply that all these problems would belong to P. The question $\mathtt{P} \neq \mathtt{NP}$ is an open one, and new techniques must be found to prove it.

We represent all these complexity classes and summarise our knowledge of the relationships between them in Figure 2.1. We do not yet know whether these inclusions are in fact equalities, or whether the figure could collapse even into a single class.

There is no known polynomial algorithm that can find an optimal solution for any problems that belongs to NP-hard. Hence, there are certain approximations, more or less satisfactory, which belong to one of the following families:

- Exact algorithms, which always find an optimal solution but take exponential time, such as branch and bound, dynamic programming, etc.
- Heuristic algorithms, which find a suboptimal (sometimes infeasible) solution but take less time, e.g. genetic algorithms, approximation algorithms, etc.

Similarly, we can translate all these concepts of complexity in terms of the space used by the algorithm.

[1]These concepts can be studied in more detail in the books *An overview on polynomial approximation of NP-hard problems* [2], *Introduction to the Theory of Computation* [3] e *Introducción a la teoría de autómatas, lenguajes y computación* [4].

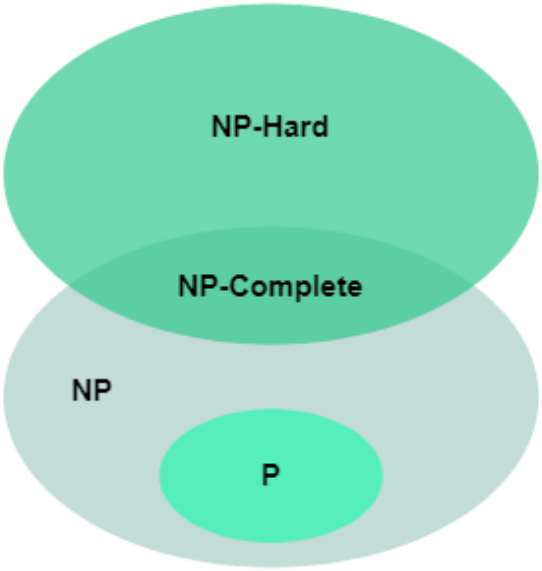


Figure 2.1: Relationship between complexity classes [5].

Definición 2.2. *If we denote* SPACE$(t(n))$ *as the set of all problems that can be solved using* $O(t(n))$ *space for a function* t *in terms of the input size* n*, we can define* PSPACE *as the set of languages that are decidable in polynomial space on a deterministic Turing machine. Formally,*

$$\mathsf{PSPACE} = \cup_k \mathsf{SPACE}(n^k)$$

Following the same process as before, PSPACE-complete problems are those decision problems in PSPACE such that every problem in PSPACE can be reduced to it in polynomial time. The PSPACE-complete problems can be seen as the hardest problems in the PSPACE class and it is conjectured that these problems are outside the P and NP complexity classes, but there is no proof of this. If any of them were in NP, then PSPACE = NP, or if any were in P, then PSPACE = P. Well-known examples of PSPACE problems are the TQBF problem and many of the problems that consist of finding winning strategies for games. Such problems will be formally introduced and will be fundamental later on.

2.3 Graph theory

Throughout this work, we will often refer to certain terminology related to graphs, which will form the problem basis representation and, therefore, it is necessary to be reviewed below.

Given a graph $G = (V, A)$, where V represents the set of vertices and $A \in \mathcal{P}'(V \times V)$ the set of edges that relate them, where $\mathcal{P}'(V \times V)$ denotes the set of all the multisets of elements of $V \times V$, we now define certain notation that we will maintain throughout the exposition of our problem.

We define the *cardinal* of a graph, $|G|$, as the number of vertices that V possesses. In the same way, if we talk about the *cardinal* of a subset of vertices $P \subseteq V$, we will refer to the number of nodes that this subset has and we will denote it as $|P|$.

The *adjacency set*, $Ady(v)$, of the graph G for a vertex $v \in V$ is the set of vertices

belonging to V that are adjacent to v, that is, such that there exists an edge $a \in A$ such that it joins both vertices.

Finally, let us call the *complement* of a subset of vertices $P \subseteq V$ to the set that contains all the vertices of V that are not in P and we will denote it as $\overline{P}$.

Thus, we have finished establishing the basic knowledge that will allow us to understand the development of the problem which, in the next section, will be fully specified.

Capítulo 3

Definición del problema

A lo largo de este capítulo definiremos el problema a tratar, del cual estudiaremos y probaremos su dificultad, que se verá directamente afectada por la forma que hemos elegido para representarlo. Consideraremos una representación muy sencilla: la propagación nacerá en uno de los nodos del grafo y se irá propagando en función de unos parámetros de propagación (enteros positivos), mientras que el apaciguador intentará frenarlo a toda costa dificultando su avance cortando un determinado número de aristas del grafo. Por esta razón, la sencillez con la que definimos nuestro problema nos impedirá en ciertas ocasiones representar situaciones que se dan en la vida real, por ejemplo, el hecho de que una propagación no siempre tenga la misma intensidad. La importancia de simplificar la representación del problema radica en que demostrar la dificultad de problemas más simples es mucho más interesante, ya que cualquier generalización de un problema siempre tiene, al menos, la misma dificultad que el problema particular. Así pues, a lo largo de este capítulo veremos la complejidad establecida para nuestra definición del problema, así como la forma en la que algunas generalizaciones la conservan. Sin más dilación, comenzamos a definir aquellos conceptos que sentarán las bases de nuestro problema.

Definición 3.1. *Un grafo de propagación es un grafo no dirigido $G = \{V, A\}$, donde cada nodo $v_i \in V$ es una terna $v_i = (i, a_i, b_i)$ donde i representa el identificador del nodo, a_i representa la presión necesaria para que el nodo sea alcanzado y b_i la capacidad de propagación que posee v_i para poder alcanzar a otros nodos. Cabe destacar que, de esta manera, $a_i, b_i \in \mathbb{Z}^+ \cup \{0\}$ y dos nodos podrán coincidir en los valores de a y b, quedando unívocamente diferenciados por su identificador. Para facilitar la notación a partir de ahora, omitiremos de la tupla el identificador pues, en nuestro caso, estamos especialmente interesados en las cargas que darán lugar a la propagación. Llamaremos conjunto de propagación al conjunto $P \subseteq V$ de nodos del grafo que han sido alcanzados por la propagación (infectados) y es claro que $\overline{P}$ son los nodos que aún no han sido alcanzados por la misma (no infectados).*

Nótese que de acuerdo a la definición de grafo dada en las nociones básicas, en nuestro grafo de propagación permitimos la existencia de *multiaristas*, cosa que nos será de gran utilidad más adelante y nos servirán para representar casos concretos como, por ejemplo, varias carreteras que conectan dos ciudades o lazos muy fuertes entre dos personas. A continuación, veamos la definición formal de propagación dentro de un grafo.

Definición 3.2. *Decimos que el conjunto de nodos $Q \subseteq V$ puede infectarse a partir del*

conjunto de nodos $P \subseteq V$ en un paso de propagación cuando cada nodo $p_i \in P$ reparte su capacidad de propagación b_i en r_i partes b_i^j que representan la parte de carga del nodo p_i que va a parar a $q_j \in Q$ si p_i y q_j son adyacentes, cumpliéndose:

$$\forall q_j \in Q, \quad \sum_{p_i \in P \cap Ady(q_j)} b_i^j \geq a_j,$$

y además

$$\forall p_i \in P, \quad \sum_{j=1}^{r} b_i^j \leq b_i.$$

Asumiremos que se cumple que b_i^j será 0 siempre y cuando p_i y q_j no sean adyacentes o bien no se necesite la carga de p_i para alcanzar a q_j (pues ya existen otros nodos pertenecientes a P que son capaces de inyectarlo solos).

Aquellos nodos libres (no infectados) que se encuentran directamente conectados a los nodos pertenecientes a la propagación decimos que pertenecen a la *zona de guerra*, un espacio que jugará un papel vital dentro de nuestro problema, pues en muchas ocasiones será donde se centre la acción. Presentamos a continuación un ejemplo de todo esto en la Figura 3.1, donde la propagación intenta avanzar a otros nodos que aún no han sido alcanzados. En rojo se presentan los nodos que ya pertenecen a P; los nodos que tienen el contorno discontinuo son los que van a ser alcanzados por la propagación en ese turno y el resto son los que aún están a salvo.

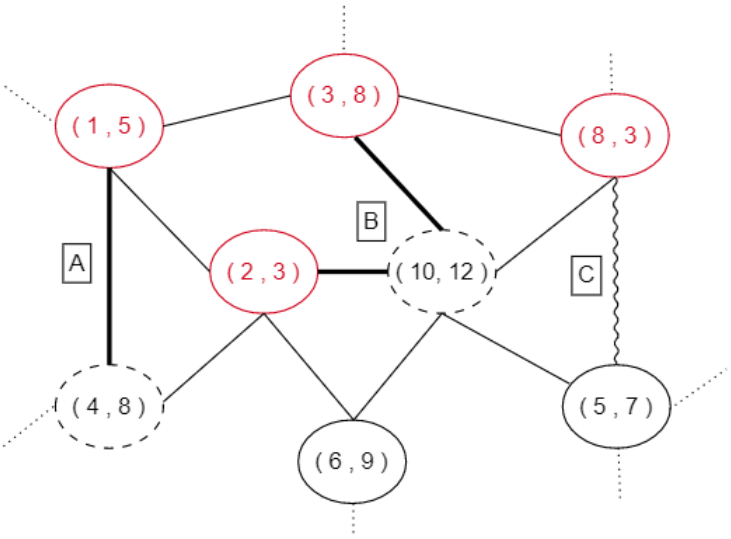


Figura 3.1: Ejemplo de distintas formas de propagación.

La situación A del grafo de la Figura 3.1 es una propagación estándar, donde la propagación avanza desde el nodo $(1,5)$ al $(4,8)$ porque el primero tiene una capacidad de propagación superior a la resistencia del segundo. Sin embargo, en la situación B los nodos $(3,8)$ y $(2,3)$ no son capaces de infectar el nodo $(10,12)$ por separado, y necesitan unir fuerzas para conseguir su objetivo. Así pues, la suma de sus cargas infectivas $(3 + 8)$ es superior a la presión necesaria (10) para alcanzar al nodo que hasta ahora era libre. Por último, en la situación C observamos que la propagación no es capaz de alcanzar el nodo $(5,7)$ porque solo dispone del nodo $(8,3)$ para avanzar y la carga del mismo

no es suficiente para neutralizar la resistencia del primero. Consecuentemente, en este turno no es posible que el nodo $(5, 7)$ se incluya a la propagación, quedando libre por el momento.

Una vez conocido a fondo el rol de la propagación dentro del grafo y los problemas que puede generar, presentamos a continuación el rol del *apaciguador*. Su objetivo será frenar el avance de la propagación, eliminando del grafo de propagación aristas de A con la esperanza de poder aislarla y salvar el mayor número de nodos posibles de su alcance. Veamos a continuación un ejemplo del funcionamiento del apaciguador con ayuda de la Figura 3.2.

Utilizando el mismo grafo de propagación que el de la Figura 3.1, se le da ahora la oportunidad al apaciguador de cortar, por ejemplo, dos aristas en su turno, antes de que la propagación avance e infecte nodos libres de la misma. De esta forma, el apaciguador debe ser inteligente y adelantarse a cuáles podrían ser los nodos que la propagación intentará atacar o cuáles pueden ser cruciales para lograr su objetivo.

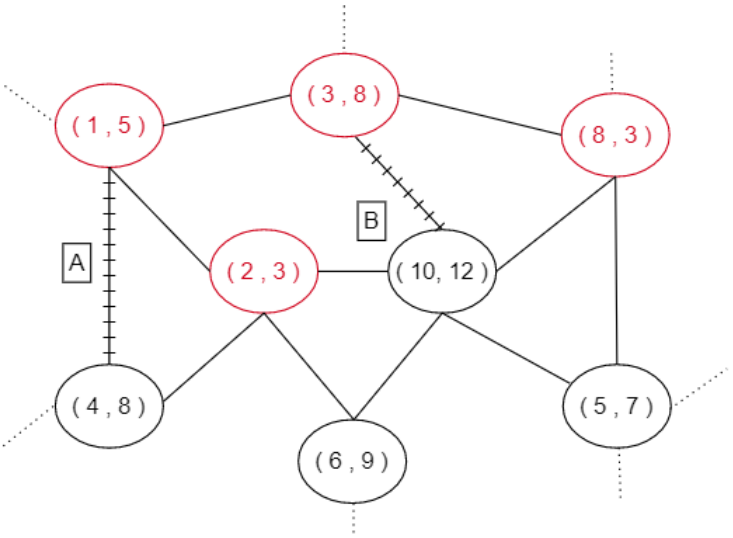


Figura 3.2: Ejemplo de apaciguador.

Así pues, presentamos en la Figura 3.2 el caso mejor para la situación de la que hablábamos, donde el apaciguador ha conseguido, cortando las aristas A y B, frenar completamente la propagación en esta zona del grafo, puesto que el vértice $(2, 3)$ ya no es capaz de propagarse a ninguno de sus nodos vecinos y el nodo $(8, 3)$ tampoco, como hemos dicho anteriormente. De esta manera podemos intuir que, en este caso, realizando esta acción óptima, conseguiría su objetivo de frenar el avance de la propagación. Esto no será así siempre, pues en otras ocasiones, aunque el apaciguador juegue sus mejores cartas, estará condenado a perder.

Ahora estamos preparados para definir nuestro juego de la propagación, donde el apaciguador intentará frenar a la propagación.

Definición 3.3. *El juego de la propagación, que denotamos como PG, se define como sigue. Sea $G = (V, A)$ un grafo de propagación y supongamos que $s, k \in \mathbb{N}$ son parámetros de entrada del juego. Consideremos que la propagación y el apaciguador juegan al siguiente juego por turnos.*

La configuración inicial del juego c_0 cumple la condición $|P| = p_0$, donde p_0 representa el número de nodos que comienzan a desarrollar la propagación inicialmente. Dichos nodos son todos aquellos en los que su información sea de la forma $(0, b_j)$, es decir, que no necesitan ninguna presión externa para ser infectados.

Los jugadores irán alternándose en cada turno para ejecutar un cambio en el grafo de propagación. El apaciguador comenzará y en cada uno de sus turnos eliminará un máximo de k aristas del conjunto A para evitar que la propagación continúe avanzando por el grafo. Así pues, se modifica el conjunto de aristas A a lo largo de los turnos de tal forma que el número de aristas decrece los turnos pasan, dificultando el avance de la propagación.

Por otra parte, en cada uno de los turnos de la propagación, ésta avanzará desde los nodos infectados a otros nodos que, al comienzo del turno, no habían sido aún alcanzados, es decir, aquellos que no pertenecen a P. La propagación no tiene ningún limite de número de nodos a los que puede avanzar, siempre y cuando estas propagaciones sean válidas de acuerdo a lo expuesto en la Definición 3.2. Nótese que el valor b_i de cada uno de los nodos solo puede utilizarse para propagar una única vez por turno, aunque pudiéndose dividir dicha carga en varios nodos todavía no infectados y conectados a él. Después del turno de la propagación, los nuevos nodos a los que ha llegado la misma quedarían añadidos al conjunto P de nodos alcanzados.

Asimismo, cabe destacar que la propagación y el apaciguador podrían tener la opción de *pasar* en su turno, no realizando ningún avance o corte respectivamente. Es importante destacar que estamos interesados en explorar siempre las mejores jugadas para cada uno de los jugadores y pasar nunca mejora las opciones de ninguno de los dos jugadores (en particular, asumiremos que la carga no "se gasta" y por lo tanto no tiene sentido que la propagación la "guarde para después"). Por tanto, que alguno de ellos tome racionalmente la decisión de pasar simplemente indicaría que el juego ha quedado decantado hacia una de las dos partes, pues la propagación no avanza porque no tiene a dónde o el apaciguador no corta aristas porque el resultado ya es obvio. Por este motivo, no tendremos en cuenta esta decisión de ahora en adelante, pues no aporta ningún caso especial que merezca la pena explorar.

Definición 3.4. *Sea G un grafo de propagación y P el conjunto de nodos alcanzados por la propagación. Llamaremos instancia o configuración del juego en un turno $i \in \mathbb{N}$ a la terna (G, P, i).*

A tal efecto, sean c_i y c_{i+1} dos configuraciones consecutivas de juego. Decimos que un movimiento z hace evolucionar al juego desde c_i a c_{i+1} si, en caso de que el turno lo posea la propagación (es decir, i es impar) la propagación avance conforme a la Definición 3.2, y en caso de que el turno lo posea el apaciguador (i par) se corten determinadas aristas, modificando en cada caso la configuración de juego como hemos explicado anteriormente, transformando c_i en c_{i+1}.

Cuando tratamos con el juego de la propagación turno a turno, no podemos prever de antemano una secuencia c de configuraciones por las que pasará el juego pues, en cada turno, el apaciguador es libre de escoger unos movimientos que condicionarán cómo avanzará la propagación en el siguiente, y esto a su vez condicionará de nuevo los movimientos del apaciguador, etc. y lo mismo ocurre con la propagación. Así pues, definimos el árbol de posibles evoluciones del juego (ver Figura 3.3) como un árbol donde cada nodo representa una configuración del juego, cada arista conecta cada configuración con

cada posible configuración siguiente, y los veredictos de quién es el vencedor se toman en las hojas del mismo. Podemos definir una posible *estrategia* para un jugador tomando solo una porción de ese árbol: para cada nodo que representa una configuración en la que le toca a ese jugador, solo hay un hijo que representa la decisión que toma y, para esa decisión, se abre el abanico de decisiones que puede tomar el jugador rival. De esta manera, podemos ver una estrategia como la decisión que toma un jugador (en la figura, una estrategia se muestra en aquellas ramas con trazo más oscuro), tal que para cualquier cosa que haga el rival en el turno siguiente, el jugador tome una única decisión en el turno siguiente, tal que para cualquier cosa que haga el rival... Por lo tanto, la secuencia de los nodos de cada una de las ramas desde la raíz a las hojas representa una posible secuencia de decisiones dentro del juego donde cada nivel nuevo de posibles movimientos depende directamente del nivel previo producido por la decisión del jugador contrario.

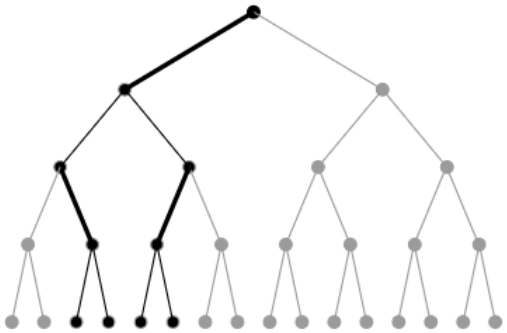


Figura 3.3: Ejemplo de estrategia.

El objetivo entonces es decidir si es posible o no encontrar una estrategia exitosa que en todos los casos garantice la victoria del apaciguador, es decir, que frene el avance de la propagación antes de queden menos nodos a salvo de los requeridos (s). Después de que en cada turno el jugador correspondiente haya realizado su movimiento en función de su estrategia, se comprueban las condiciones de victoria para cada una de las partes. En el caso de la propagación, su condición de triunfo en el juego está basada completamente en el número de nodos a los que ha conseguido propagarse en total, consiguiendo la victoria si $|\overline{P}| < s$. Por otra parte, si hablamos del triunfo del apaciguador, su condición de victoria es la opuesta, $|\overline{P}| \geq s$. La forma de lograr este objetivo es conseguir que aquellas aristas que queden en pie tras sus cortes no sean suficientes para que la propagación pueda avanzar, es decir, que la propagación quede completamente bloqueada pues la suma de las cargas efectivas de infección no sea superior a la presión necesaria para propagarse a cualquier nodo destino $v_i \in \overline{P}$.

$$\sum_{v_j \in P \cap Ady(v_i)} b_j < a_i$$

Un caso límite del anterior sería que el apaciguador lograse romper todas las aristas adyacentes a los nodos que pertenecen a la propagación, aislando la misma de tal manera que no pudiera propagarse a ningún nodo que aún no ha sido expuesto. De esta forma, si entonces el número de elementos libres de propagación es superior a s, es decir, $|\overline{P}| \geq s$, el apaciguador habrá logrado su objetivo de salvar a al menos s nodos de la propagación.

En consecuencia, el juego finalizará cuando alguno de los jugadores, ya sea la propagación o el apaciguador, logre su objetivo. Asumimos que todas las decisiones tomadas por cada una de las partes son públicas y visibles por el otro jugador. Diremos pues que una estrategia se considera exitosa para el apaciguador para una configuración concreta si existe un movimiento que puede realizar en dicha configuración tal que, para todas las posibles jugadas de la propagación, existe un movimiento del apaciguador tal que ..., y así sucesivamente hasta que en algún turno posterior que cumpla su condición de victoria. Es decir, tal que todas las hojas de las ramas que abre dicha estrategia denotan una victoria del apaciguador.

Definición 3.5. *El éxito en el juego de la propagación, denotado como* `EPP`*, es el siguiente problema: dado el grafo de propagación $G = (V, A)$, un subconjunto $P \subseteq V$ de vértices que han sido alcanzados por la propagación, el número k de cortes posibles del apaciguador y un número $s \in \mathbb{N}$ (denotando el número de nodos que deben quedar libres de la misma), ¿existe una estrategia exitosa para el apaciguador desde la configuración c, tal que consiga salvar s nodos del alcance de la propagación, es decir, $|\overline{P}|$ sea mayor o igual que s?*

Una vez conocido el problema de decisión es trivial definir el problema de optimización de la propagación, que denotamos por `PPO`*. Dados los mismos datos de entrada que antes excepto s, el* `PPO` *consiste en encontrar una estrategia que maximice el número de nodos que quedarán libres en el caso peor.*

Ahora introduciremos la complejidad de `EPP`.

Teorema 3.1. `EPP` $\in$ `PSPACE-completo`.

La demostración de este teorema, desarrollada en el siguiente capítulo (Capítulo 4), incluye en las primeras líneas un método bastante directo para computar `EPP` en espacio polinómico y tiempo exponencial, categorizándolo en `PSPACE`. El resto de la demostración se dedica a demostrar su `PSPACE-dureza`. Esta dificultad motiva que sea solucionado por métodos heurísticos, como el algoritmo minimax, como veremos en el Capítulo 5.

Para mostrar la facilidad con la que podemos generalizar nuestro problema para representar escenarios más complejos, introduciremos brevemente algunos ejemplos de estas extensiones. Por un lado, podríamos establecer que el grafo fuese un *grafo ponderado*, en el que cada arista tuviese asociado un peso y, por lo tanto, el apaciguador tuviese un peso máximo que erradicar en cada turno. En la realidad, es común cuando hablamos de propagaciones de fuego en un incendio forestal que haya mayor dificultad a la hora de construir cortafuegos en unas zonas que en otras. Por otra parte, cada uno de los nodos podría tener una *fecha de expiración*, es decir, un número de turnos durante los cuales puede seguir contribuyendo a propagar una vez ha sido alcanzado por la propagación. La traducción directa a la realidad, siguiendo con el ejemplo del incendio, queda directamente reflejada en que un árbol no puede arder para siempre, tarde o temprano acaba consumiéndose.

Podemos asegurar que la dureza de los problemas resultantes de añadir dichas modificaciones a nuestro problema seguiría siendo al menos la misma (`PSPACE-duro`). Esta dureza es trivialmente heredada del problema original `EPP`, ya que lo generalizan.

Capítulo 4

Análisis de complejidad

En este capítulo, analizaremos la complejidad del Problema de la Propagación desarrollado en el capitulo anterior, demostrando que EPP es un problema PSPACE-completo, como ya se estableció en el Teorema 3.1, a través de una reducción polinómica al problema TQBF.

Teorema. EPP $\in$ PSPACE-completo.

Demostración. En primer lugar, probemos la pertenencia a PSPACE. Veamos que es fácil comprobar que un algoritmo minimax que comprueba todas las posibles secuencias del juego hasta $|V| - s$ movimientos de la propagación soluciona EPP en espacio polinómico con respecto al orden del tamaño de la entrada.

Nuestro algoritmo utilizará un vector de ternas (id, a, b) que recogerá la información de todos los nodos, donde *id* sea un identificador que denote unívocamente al nodo y *a,b* serán los parámetros de propagación del nodo. Dicho vector irá acompañado de una matriz de adyacencia que indicará las aristas que conectan los nodos, y juntos describirán la información necesaria para construir el grafo G utilizando un tamaño de memoria del orden de $|V|^2$.

Para definir una instancia del juego, basta con tener en cuenta el grafo anterior, los parámetros del juego k, s que son dos números naturales, y el conjunto de identificadores de nodos P que han sido alcanzados por la propagación. El tamaño de $|P|$ será como máximo del orden de $|V|$, pues no puede haber más nodos pertenecientes a la propagación de los ya existentes.

A pesar del tamaño exponencial del árbol, causado por la gran cantidad de posibilidades que se abren a la hora de elegir entre la variedad de opciones que tiene cada jugador para su jugada, mientras el algoritmo se está ejecutando solo necesitamos recordar dónde estamos a lo largo de la exploración y si donde nos encontramos puede aún pertenecer a una estrategia exitosa. Para lo primero, como decíamos antes, se requiere una cantidad polinómica de memoria (pues necesitaremos recordar los movimientos que nos han llevado a estar en esa rama) mientras que, para lo segundo, necesitaremos una cantidad constante. Asumiendo que s representa el número de nodos que queremos salvar de la propagación, y $|V|$ el número total de nodos en el grafo, por construcción del problema, el máximo valor para la profundidad en la que nos encontraremos en el árbol de decisión nunca será

mayor a $2 * (|V| - s)$. Esto es así, puesto que la propagación avanza, si es posible, en uno de cada dos turnos, por lo que necesitaremos como mucho el doble de turnos que $|V| - s$ para comprobar que ha sido frenada y que por lo tanto hay, al menos, s nodos libres de la propagación.

Asimismo, es claro que cada acción específica de nuestro algoritmo requiere un número constante de variables auxiliares. Concluimos así que el algoritmo es ejecutable en espacio polinómico y, por lo tanto, EPP $\in$ PSPACE.

Para probar que nuestro problema es PSPACE-duro, consideramos el problema TQBF (*true quantified boolean function problem*), que extiende SAT permitiendo la cuantificación de las variables, ya sea existencialmente ($\exists$) o universalmente ($\forall$). Este problema de decisión PSPACE-completo está definido como explicamos a continuación. Una QBF es decir, una instancia de TQBF tiene la forma

$$Q_1 x_1 Q_2 x_2 ... Q_n x_n \varphi$$

donde φ es una fórmula proposicional conteniendo exactamente las variables $x_i \in \{\top, \bot\}$ y $Q_i \in \{\exists, \forall\}$ con $(i = 1...n)$, donde los cuantificadores $\exists$ y $\forall$ se alternan estrictamente en la secuencia $Q_1, ..., Q_n$ y no hay ninguna variable libre, es decir, sin ningún cuantificador asociado. TQBF es el conjunto de expresiones QBF que son evaluadas a $\top$. Por lo tanto, encontrar una solución para el problema TQBF consiste en averiguar si la expresión dada equivale a $\top$ o a $\bot$ en función de la asignación dada a cada variable. Además, a partir de ahora supondremos que la fórmula proposicional φ está expresada en forma normal conjuntiva (CNF), es decir, una conjunción de cláusulas, donde cada cláusula está compuesta por una disyunción de literales. Esta versión restringida del problema TQBF, también es PSPACE-completa, como se explica en el Capítulo 4 del libro *Computational Complexity: A Modern Approach* de S. Arora y B. Barak [6].

Para probar que EPP $\in$ PSPACE-duro, reduciremos polinómicamente TQBF a EPP. En primer lugar, presentaremos informalmente la demostración. La propagación y el apaciguador jugarán al juego de la propagación. Cada valor posible, $\top$ o $\bot$, de cada una de las variables representará un nodo en el grafo de la propagación. Así pues, si un nodo es alcanzado por la propagación, la variable asociada a él quedará con el valor correspondiente a dicho nodo, siendo responsabilidad del apaciguador evitar que a una misma variable se le asignen dos valores distintos, para evitar contradicciones que implicarán la derrota del apaciguador. La propagación se encargará de establecer los valores de las variables asociadas a los cuantificadores universales, mientras que el apaciguador controlará las variables existenciales. Establecemos en nuestra instancia de EPP $k = 1$ y, por lo tanto, el apaciguador solo podrá cortar una única arista por turno. De esta manera, para cada variable bajo el control de la propagación, ésta tendrá libertad para elegir qué valor quiere dar a dichas variables ($\top$ o $\bot$) en su turno mientras que, para las variables que controla el apaciguador, éste podrá ir dirigiendo la propagación, cortando la arista que lleve al valor que no le interese, para que la propagación solo tenga la opción de avanzar hacia el valor que el apaciguador haya elegido. Sea p el número de cuantificadores en la expresión QBF y q el número de cláusulas de la expresión CNF, φ. Para que el apaciguador consiga su objetivo y tenga éxito en el juego de la propagación, tendrá que lograr salvar un número de nodos del alcance de la propagación durante los turnos del juego, en particular, $\frac{5p}{2} + q$. Para la instancia del problema EPP que construiremos, comprobaremos si existe una estrategia

exitosa para el apaciguador.

Introduciremos ahora la noción de *nodos bomba* que resultarán vitales a lo largo de esta demostración. Sea $v \in V$ un nodo del grafo de propagación, se considera que v es un *nodo bomba* cuando su valor b, que representa la carga infectiva hacia otros nodos, es mayor que la suma de todas las resistencias del resto de nodos, es decir, cuando es un nodo capaz de avanzar de una forma catastrófica en favor a la propagación. Denotaremos este valor especial como ∞, el cual será un simple azúcar sintáctico, pues siempre podria provocarse el mismo efecto usando cierto $b \in \mathbb{N}$ mayor que la suma de las cargas requeridas para infectar al resto de nodos del grafo. Además, todos los nodos bomba estarán unidos entre sí, por lo que con que cualquiera de ellos sea alcanzado por la propagación, en un solo turno todos estos nodos serían alcanzados por la propagación. Puestos en esta situación, parece evidente que es de vital importancia para el apaciguador evitar que éstos sean alcanzados, puesto que significaría una victoria inminente para la propagación. Por otro lado, para esta demostración en concreto denominaremos *nodos puente* a aquellos nodos que no se utilicen para representar la asignación de un valor a una variable pero tampoco sean nodos bomba.

El proceso de asignación de las variables del problema es representado implícitamente por el propio grafo que construimos para la instancia de `EPP`, donde un nodo puente que aparece como "raíz" inicia la propagación y los posibles valores $(\top, \bot)$ de la variable existencial más externa de la fórmula (x_1) forman el primer nivel. Los posibles valores de la primera variable universal (x_2) forman el segundo nivel y ambos están conectados con los dos nodos que representaban la primera variable. Finalmente, otro nodo puente comunica los valores de x_2 con el siguiente par de variables. De esta forma, se irían concatenando todas las variables de dos en dos y resultaría un grafo como el que puede apreciarse en la Figura 4.1, que será la base de nuestro problema.

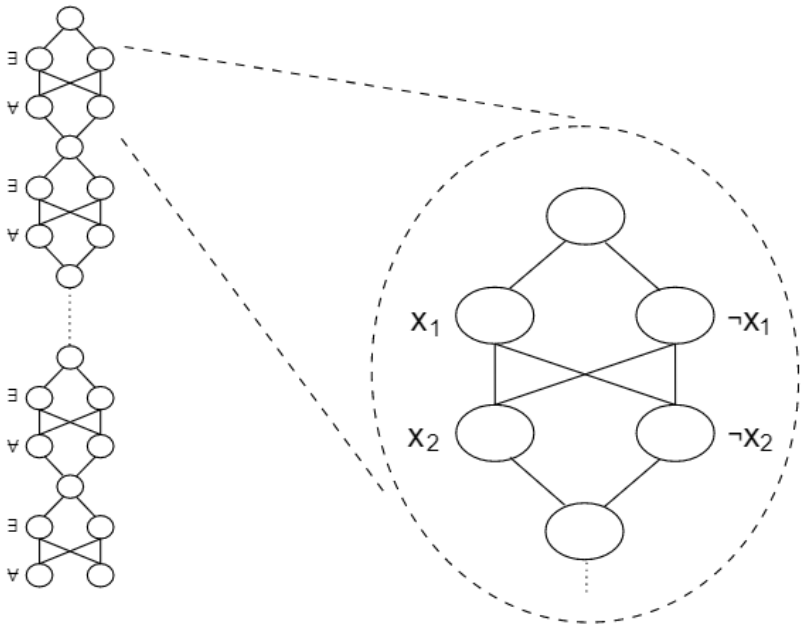
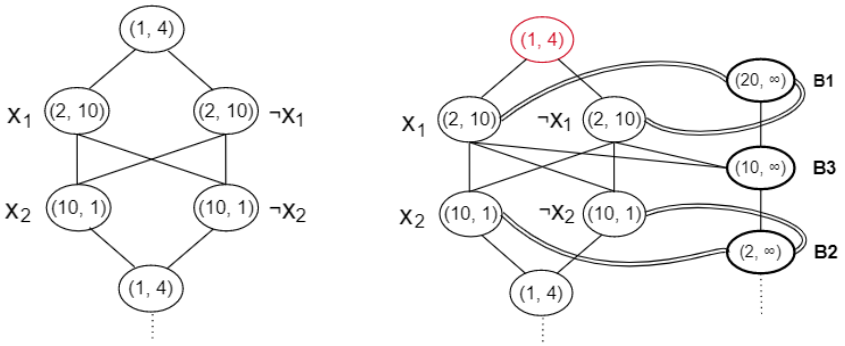


Figura 4.1: Estructura general.

Cabe destacar que, descendiendo a través de esta estructura de grafo, se podrá ir eligiendo solo uno de los valores posibles para cada variable existencial (ya veremos próximamente

como lo forzaremos, introduciendo nodos bomba) pero, desde cada valor existencial, será posible alcanzar cualquiera de los dos posibles valores de la siguiente variable universal. Además, los nodos puente funcionarán como nexo entre las iteraciones del proceso de asignación de valor de las variables, existiendo siempre un nodo puente entre los valores de variables universales (pares) y los valores de variables existenciales (impares). Por ejemplo, en la Figura 4.1 vemos dos nodos puente: el que se sitúa en la parte superior y dará comienzo a la propagación, y el de la parte inferior, que comunicaría con los nodos de la siguiente variable existencial (x_3), formando una cadena que nos facilitará el proceso iterativo. Por este motivo, desarrollaremos nuestra explicación para el primer par de variables (x_1 y x_2), siendo completamente análogo para el resto.



(a) Estado inicial del grafo

(b) Estado inicial incluyendo las bombas asociadas

Figura 4.2: Estado inicial.

Las Figuras 4.2 y 4.3 ilustran el avance de la propagación en el grafo. Como especificamos en la definición del problema, el primer turno es del apaciguador, y el único nodo que de momento pertenece a la propagación es el nodo puente de la parte superior. Veamos que, partiendo de la Figura 4.2, es necesario que el apaciguador corte una de las dos aristas que salen del primer nodo puente que ya pertenece a P para intentar frenar en su mayor medida a la propagación. Podría ocurrir que el apaciguador decidiese no cortar ninguna arista adyacente al nodo perteneciente a la propagación, cortando una arista alejada de la *zona de guerra*, y permitiendo al nodo inicial con valores $(0, 4)$ propagarse tanto a x_1 (con valor $(2, 10)$) como a $\neg x_1$ (con valor $(2, 10)$) en el mismo turno. Este escenario presenta uno de los casos prohibidos del problema `TQBF`, que no permite poner una misma variable a sus dos valores posibles. Por este motivo, es de vital importancia evitar que esto ocurra y, para ello, emplearemos los nodos bomba. Uniendo x_i y $\neg x_i, 1 \leq i \leq p$, a un nodo bomba que necesite la suma de sus capacidades infectivas para poder ser alcanzado por la propagación, conseguiremos que el apaciguador no pierda el interés en cortar las aristas que pertenecen a la zona de guerra, pues de ellas depende que pierda inminentemente la partida. Así pues, los nodos que representan x_1 y $\neg x_1$ quedarán unidos a la B1 $(20, \infty)$ y en el caso de x_2 y $\neg x_2$, a la B2 $(2, \infty)$. De esta manera, cortando ya sea la arista que lleva a x_1 ($x_1 = \top$) o la que lleva a $\neg x_1$ ($x_1 = \bot$), vemos que el apaciguador se encarga de dar valor a la primera variable, que como hemos explicado anteriormente tiene asociado un cuantificador existencial.

Pero, ¿qué pasaría si el apaciguador intentase desactivar un nodo bomba cortando una de las aristas que lleva a él? Así, conseguiría aislar el nodo bomba y la propagación avanzaría hacia los dos valores de una misma variable en el mismo turno, lo que llevaría a una

contradicción dentro de la instancia del problema TQBF. Para evitar esto establecemos la necesidad de que los nodos bomba que controlan la unicidad en el valor de las variables tienen que estar unidos a los nodos que representan los valores de dichas variables por *multiaristas*, es decir, muchas aristas que conectan los mismos nodos (en este caso, cada nodo bomba con los nodos de valores de variables correspondientes). Aunque el apaciguador se centrase en cortar repetidamente estas aristas, el número de aristas que unen el nodo bomba al otro nodo siempre será superior a la cantidad de cortes que puede realizar el apaciguador a lo largo de todos los turnos. Nótese que el surgimiento de las multiaristas para nada rompe lo ya acordado en la Definición 3.1, donde ya se contemplaba la necesidad de las mismas y la posibilidad de su aparición con cualquier finalidad.

Prosiguiendo con esta instancia, supongamos pues que el apaciguador ha cortado la arista que lleva a x_1 por lo que en el siguiente turno, el de la propagación, ésta solo tiene una opción: propagarse hacia la variable que no ha cortado el apaciguador. Le tocaría otra vez el turno al apaciguador, pero como necesitamos que sea la propagación la que elija entre los valores x_2, $(10, 1)$, y $\neg x_2$, $(10, 1)$, pues es una variable universal, debemos entretener al apaciguador de alguna forma. De nuevo, recurriremos a los nodos bomba para ello. Añadiendo el nodo bomba (en este caso, B3) con una resistencia igual a la capacidad de propagación del nodo existencial precedente, conseguiremos pues focalizar la atención del apaciguador en él durante este turno si, de nuevo, no quiere perder de forma inminente.

A continuación, sería momento de que la propagación avanzase, como adelantamos anteriormente, eligiendo entre los valores de la variable universal y, una vez tomada la decisión, la propagación avanza. En el caso que vemos en la Figura 4.3, por ejemplo, la propagación decide escoger x_2.

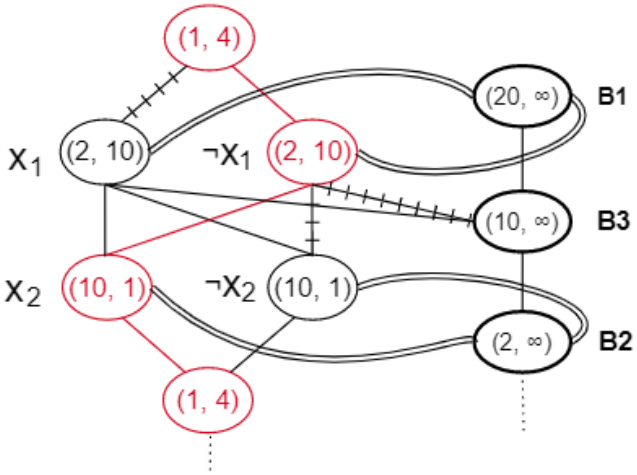


Figura 4.3: Posible desarrollo de la primera iteración.

Por último, los dos siguientes turnos son triviales. El apaciguador no tendrá otra opción que cortar la arista que lleva al otro valor de la variable universal que la propagación ha conseguido alcanzar pues, en caso de que no lo hiciera, la propagación podría conseguir poner la misma variable a dos valores distintos, activando el nodo bomba relacionado con la variable y perdiendo de manera inmediata. Por otra parte, la propagación solo podrá ejecutar un movimiento: avanzar hacia el nodo puente del siguiente nivel y, de esta forma,

comenzar la siguiente iteración del proceso de la misma manera en que se ha procedido ahora.

Nótese que, por construcción, la propagación no es capaz de subir a niveles superiores ni capaz de avanzar desde los mismos. Lo primero sería posible desde uno de los nodos que representan variables universales pero, por construcción, tienen menos carga de propagación que el existencial; otra posible forma de subir sería desde un nodo puente a un nodo universal, pero ocurre igual. Por otra parte, no es posible que la propagación avance en una dirección distinta a la siguiente variable pues el camino a seguir está dirigido por el apaciguador, solo pudiéndose salir de él si éste comete algún error, lo que llevaría inminentemente a la victoria de la propagación.

Una vez establecido cómo se desarrolla el proceso iterativo de elegir el valor de cada una de las variables, queda responder la pregunta de cómo se comprueba la satisfacción de la fórmula φ. Recordemos que dicha fórmula está compuesta por la conjunción de cláusulas que, a su vez, son disyunciones de literales. Así pues, la fórmula será evaluada a verdadera si y solo si, por la definición de conjunción lógica, cada una de las cláusulas es verdadera; o, lo que es lo mismo, la fórmula será evaluada a falsa si y solo si cualquiera de las cláusulas es falsa. De esta manera, es de vital importancia comprobar que, según se van tomando decisiones a lo largo del grafo para poder establecer el valor de las variables, se tenga también en cuenta que las cláusulas se mantienen ciertas. Para comprobar que cada una de las cláusulas sean verdaderas, debemos plantearnos cuándo son falsas y, como cada una de las cláusulas son disyunciones, es claro que será falsa cuando todos los literales lo sean. En conclusión, la fórmula será evaluada a verdadera si y solo si no todos los literales de una misma cláusula son falsos.

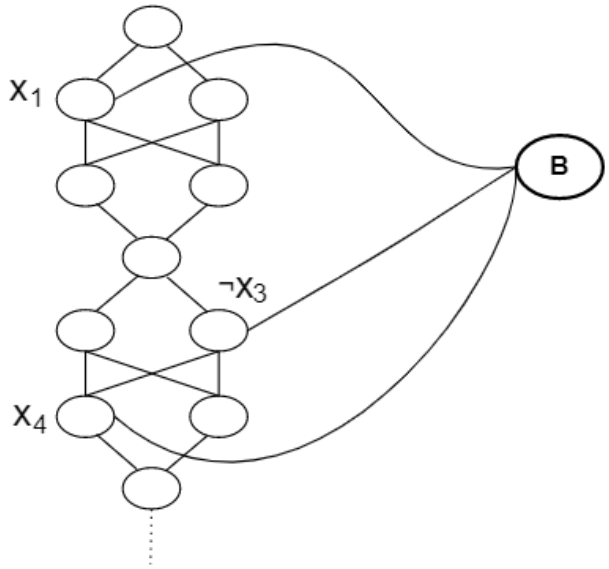


Figura 4.4: Satisfactibilidad de un ejemplo de cláusula de φ

La pregunta de cómo comprobar la satisfacción de la fórmula φ claramente pasa por chequear los valores de los variables y, como es necesario notificar si los literales son falsos, la traslación directa a nuestro problema consiste en unir la negación de cada uno de los literales de cada cláusula a un nodo bomba, como podemos ver en la Figura 4.4. La presión necesaria para ser alcanzada por la propagación tiene que ser por necesidad la

suma de la capacidad de propagación que poseen cada uno de los nodos que representan la negación de los literales de modo que, si (y solo si) todos los nodos han sido alcanzados, el nodo bomba estará en peligro.

Una vez vistos todos los detalles sobre cómo funcionan las bases de nuestra demostración, vamos a explicar qué se entiende por una estrategia vencedora dentro de esta instancia. Si nos restringimos a la definición del problema, es claro que el objetivo que tiene que tener el apaciguador en mente es conseguir salvar los nodos establecidos por la instancia del problema. Para que esto sea posible. lo más importante que debe tener el apaciguador en cuenta si quiere conseguir una estrategia victoriosa es, como hemos estado viendo durante todo el proceso, evitar a toda costa que la propagación alcance a un nodo bomba compaginando, de la mejor manera posible, la buena elección de las variables existenciales para que las bombas asociadas a las cláusulas de la fórmula φ no sean alcanzadas y, a su vez, que la propagación avance por el grafo de forma que complete la elección de valor para todas las variables.

A continuación, presentamos formalmente la reducción polinómica desde TQBF a EPP. Sin pérdida de generalidad, sea $\exists x_1 \forall x_2 ... \exists x_{p-1} \forall x_p \varphi$ una instancia de TQBF, donde $\varphi = \alpha_1 \wedge ... \wedge \alpha_q$ y, para todo $1 \leq i \leq q$, tenemos que $\alpha_i = a_1^i \vee ... \vee a_{l_i}^i$ donde, para todo $1 \leq j \leq l_i$, los a_j^i son literales sobre variables proposicionales.

Desde dicha instancia de TQBF, construiremos una instancia de EPP de la siguiente manera:

- El conjunto de vértices del grafo de propagación, V, está compuesto por aquellos nodos que están asociados a los valores $\top, \bot$ de las variables $x_1 ... x_p$, los nodos bomba utilizados en cada uno de los casos que veremos a continuación y los nodos puente que unen cada una de las iteraciones en la elección de valores para las variables.
- El conjunto de aristas del grafo de propagación, A, está compuesto por las aristas que unen los vértices de V de la forma que hemos visto en el desarrollo anterior. Cabe destacar la existencia de multiaristas en el caso de algunos nodos bomba.
- Los parámetros del juego quedan, como hemos estado viendo, establecidos de la siguiente forma:
  - $s = \frac{5p}{2} + q$, donde p es el número de variables y q el número de cláusulas que aparecen en la fórmula CNF,
  - $k = 1$.
- La configuración inicial del juego es c_0, compuesta por el grafo inicial G de la forma que hemos establecido en las explicaciones anteriores, el conjunto P de nodos que pertenecen a la propagación, inicializado a un único nodo (el primer nodo puente), y el turno $i = 0$.
- El número total de vértices, $|V|$, es la suma de
  - El número de nodos que están relacionados con el valor de una de las variables es, claramente, $2p$.
  - Para saber el número exacto de nodos bomba tenemos que describir los tres casos en los que aparecen en nuestra instancia:

- Un nodo bomba por cada una de las variables, para evitar que una misma variable sea puesta a $\top$ y $\bot$ a la vez $\rightarrow p$.
- Un nodo bomba extra por cada una de las variables universales, para garantizar el adecuado funcionamiento de los turnos $\rightarrow \frac{p}{2}$.
- Un nodo bomba por cada una de las cláusulas que aparecen en la fórmula `TQBF`, para comprobar si la fórmula φ se satisface $\rightarrow q$.

De esta manera, el número de nodos bomba es $p + \frac{p}{2} + q = \frac{3p}{2} + q$.

– El número de nodos puente, siguiendo la construcción realizada anteriormente, es $\frac{p}{2}$

Por lo tanto, $|V| = 2p + \frac{3p}{2} + q + \frac{p}{2} = 4p + q$.

Es fácilmente comprobable que el tamaño de la instancia del `EPP` es polinómico con respecto al tamaño de la instancia original de `TQBF`. A continuación, probaremos que la solución del problema original `TQBF` es *sí* si y solo si la respuesta a la instancia de `EPP` que hemos construido es *sí*.

$\Longrightarrow$ Supongamos que $\exists x_1 \forall x_2 ... \exists x_{p-1} \forall x_p \varphi$ es verdadero. Eso es, existe un x_1 tal que para todo x_2, existe un x_3 tal que ... tal que φ se satisface. Eso significa que, si vemos el problema `TQBF` como un juego en el que el jugador 1 elige el valor de las variables impares y el jugador 2 escoge el valor de las pares, donde el objetivo es que el jugador 1 complete la cláusula φ, entonces el jugador 1 tiene una estrategia ganadora. Veamos que la translación directa de esta estrategia hacia el juego de la propagación es exitosa para el apaciguador. En el juego `TQBF` el valor de las p variables es alternativamente fijado por cada jugador, siguiendo el orden en el que aparecen en la expresión `QBF`, hasta que todas las variables tienen un valor asignado, así que evidentemente esto también ocurre de esta manera cuando el jugador 1 sigue dicha estrategia ganadora.

Nótese que no en todos los turnos de la propagación y del apaciguador se pone valor a una de las variables. Es más, si el apaciguador llegase a hacer un movimiento inesperado resultaría en la victoria inmediata de la propagación, incluso sin necesidad de llegar a establecer los valores de todas las variables, pues en todos esos casos la propagación lograría alcanzar algún nodo bomba. Así pues, si el apaciguador decidiera cortar una arista alejada de la zona de propagación que le diera ventaja a la misma, no bloquease alguno de sus avances o formara parte de un conjunto de multiaristas, se declararía inmediatamente después la victoria de la propagación.

Supongamos entonces que el apaciguador no hace ningún movimiento inesperado que le haga perder inmediatamente y la propagación sigue las normas del juego según se han establecido. Podemos asegurar entonces que ninguno de los nodos bomba encargados de garantizar la correcta rotación de los turnos antes de elegir los valores para las variables universales han sido alcanzados y, así, tenemos los primeros $\frac{p}{2}$ nodos a salvo.

Una vez comprobado lo anterior, es claro que tenemos una instancia `TQBF` en la que a cada variable se le ha asignado un único valor y la fórmula φ se satisface. De esta manera, podemos garantizar que ninguno de los nodos bomba que se encargaban de asegurar que la propagación no alcanzaba a dos valores de variables contrarios

$(\top, \bot)$ ha sido alcanzado por la propagación y, por lo tanto, tenemos otros p nodos que se han salvado. Por otro lado, como a cada una de las variables se le ha asignado un único valor, el valor opuesto de cada una tampoco ha sido alcanzado por la propagación, sumando p nodos más a la lista de nodos salvados. La propagación llegaría así a la parte final del grafo habiendo conquistado $\frac{3p}{2}$ nodos, los $\frac{p}{2}$ nodos puente y p nodos que representan los valores a los que se fijan las variables TQBF, y no pudiendo avanzar más pues no puede ir hacia arriba a por otros nodos que representan otros valores de las variables ya fijadas ni puede atacar a los nodos bomba, por construcción.

Por último, como explicábamos anteriormente, al tener una instancia de TQBF en la que la fórmula φ se satisface, podemos garantizar que ninguna de las cláusulas que forman la conjunción es falsa, y por lo tanto ninguno de los nodo bomba relacionados con ellas ha sido alcanzado por la propagación, lo que nos da los últimos q nodos libres. Por consiguiente, quedarían $\frac{5p}{2} + q$ nodos para asegurar que existe una estrategia ganadora para el apaciguador en el juego de la propagación.

$\Longleftarrow$ Supongamos ahora que el apaciguador tiene una estrategia ganadora en el juego de la propagación. Veamos que una estrategia TQBF derivada de dicha estrategia es necesariamente ganadora para el jugador 1. Una estrategia ganadora para el apaciguador implica que, tras jugar por turnos con la propagación, ha conseguido detener el avance de la propagación y salvar al menos $\frac{5p}{2} + q$ nodos del alcance de la misma. Veamos pues que no hay forma de que dichos nodos salvados sean diferentes a los que nos convienen para realizar la traslación entre las instancias de ambos problemas.

En primer lugar, si el apaciguador ha conseguido salvar cualquier cantidad de nodos es porque ninguno de los nodos bomba ha sido alcanzado por la propagación, puesto que, de ser así, por construcción de la instancia y el funcionamiento de los nodos bomba, todos los nodos hubieran sido alcanzados por la propagación. Por lo tanto, el hecho de que ninguno de los nodos bomba $(\frac{3p}{2} + q)$ haya sido alcanzado nos permite asegurar dos cosas: que ninguna variable ha obtenido el valor de $\top$ y $\bot$ a la vez por lo que hemos salvado otros p nodos (haciendo el total de $\frac{5p}{2} + q$); y que ninguna de las cláusulas de la fórmula CNF ha fallado. En consecuencia, está claro que existe un x_1 tal que para todo x_2, existe un x_3 tal que ... tal que φ se satisface y concluimos pues que la estrategia para TQBF derivada de la parte de la estrategia exitosa de EPP (donde solo se llevan a cabo movimientos válidos dentro del problema TQBF) es una estrategia ganadora para el jugador 1 en TQBF.

□

Capítulo 5

Resolución práctica

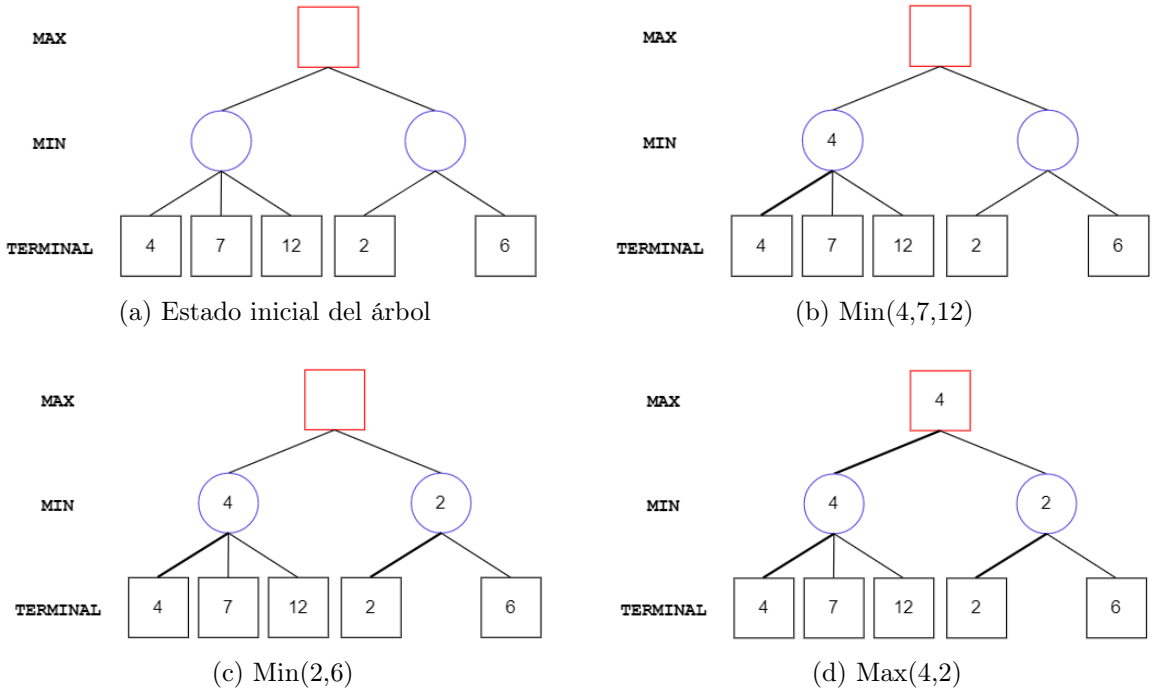
A lo largo de este capítulo presentaremos nuestro algoritmo heurístico utilizado para hacer frente al problema EPP: el algoritmo minimax con poda alfa-beta, con el que pretendemos averiguar, dada una configuración del juego de la propagación, cuál es el primer paso que debe dar el apaciguador para intentar frenar de la mejor manera posible el avance de la propagación. Todo el código al que se hará referencia en este capítulo ha sido desarrollado en Python, utilizando el servicio Cloud basado en Notebooks de Jupyter, Colaboratory de Google, y se puede encontrar aqui [7].

Antes de comenzar a desmenuzar todos los aspectos relacionados con nuestra implementación de dicho algoritmo, es momento de especificar las distintas utilidades del mismo, nacidas directamente de la motivación de nuestro problema. Por una parte, la principal línea intuitiva que podemos seguir es utilizar nuestro algoritmo como una herramienta para saber en tiempo real cuál es la mejor jugada que podemos realizar (como cualquiera de los jugadores) dada una configuración determinada del juego de la propagación. De esta forma, obtenemos una herramienta con una aplicación social que podría ser utilizada para ver cuál es la mejor estrategia en una situación real de cualquier tipo de propagación. Por otra parte, si lo miramos desde el punto de vista de un juego, podemos utilizar el algoritmo o bien para simular qué haría el oponente en su turno y jugar contra el algoritmo, o bien para que juegue el algoritmo contra si mismo, alternándose y realizando siempre la mejor jugada para cada uno de los jugadores. Esto permitiría estudiar con detenimiento los tiempos y el porqué de cada una de las decisiones que ambos jugadores van tomando. Teniendo todo esto en cuenta, pasamos a explicar paso a paso la base sobre la que se sustenta nuestra implementación: el algoritmo minimax.

El *algoritmo minimax* es un algoritmo recursivo usado principalmente para intentar averiguar el movimiento óptimo para nuestro jugador asumiendo que el otro jugador estará jugando también de manera óptima. Sigue un funcionamiento bastante similar a la forma en que nosotros como jugadores pensaríamos: “si hago este movimiento, entonces mi oponente podrá hacer estos otros movimientos y yo podré...". El algoritmo minimax es llamado así debido a que ayuda a escoger el movimiento óptimo para minimizar la pérdida, mientras que el otro jugador busca su estrategia para maximizar tu pérdida. Pero, ¿cómo medimos esta pérdida?

Cada una de las configuraciones de juego y, por lo tanto, cada una de las iteraciones que realiza el algoritmo minimax, tiene asociado un valor heurístico, que determina cómo de

bueno es el estado del juego donde nos encontramos. Por consiguiente, el algoritmo explorará los nodos del árbol asignándoles un valor numérico mediante una función heurística, empezando por los nodos terminales y subiendo hacia la raíz. Veamos el funcionamiento de este algoritmo con un ejemplo trivial:



(a) Estado inicial del árbol

(b) Min(4,7,12)

(c) Min(2,6)

(d) Max(4,2)

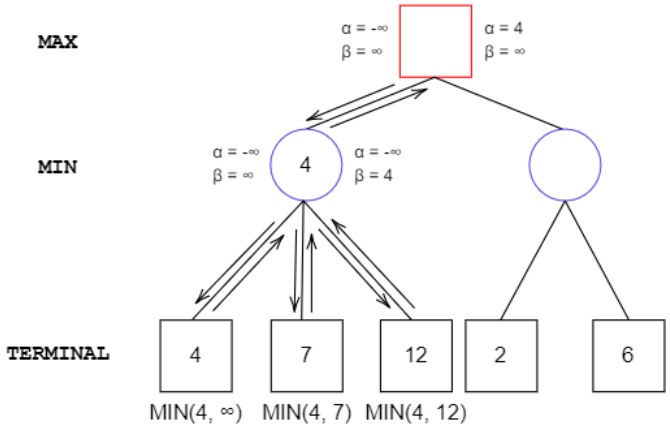
Figura 5.1: Ejemplo del algoritmo minimax.

Como podemos ver en la Figura 5.1, hay dos jugadores involucrados en el juego, llamados MIN y MAX. Mientras que el jugador MAX intenta obtener el máximo resultado posible, MIN intenta lo contrario. En primer lugar, generamos el árbol completo de juego con la posición actual de juego en la raíz. Para ello, generamos para cada nodo todos los posibles movimientos que suceden a esa configuración del juego. En el caso de las hojas, que representan los estados finales del juego (nodos terminales), si uno de los jugadores pierde o ambos quedan empate, se le asigna el valor obtenido para esa sucesión de jugadas. Así, subiendo desde los nodos hojas hasta la raíz, el mínimo entre los nodos con valor 4, 7 y 12 es 4, y en el caso de la otra rama el mínimo entre 2 y 6 es 2. Para finalizar, el máximo entre 4 y 2 es claramente 4, que queda reflejado en el nodo raíz y sirve para escoger cuál de las dos opciones de jugada será más beneficiosa para el jugador MAX a pesar de que el jugador contrario haga su mejor movimiento.

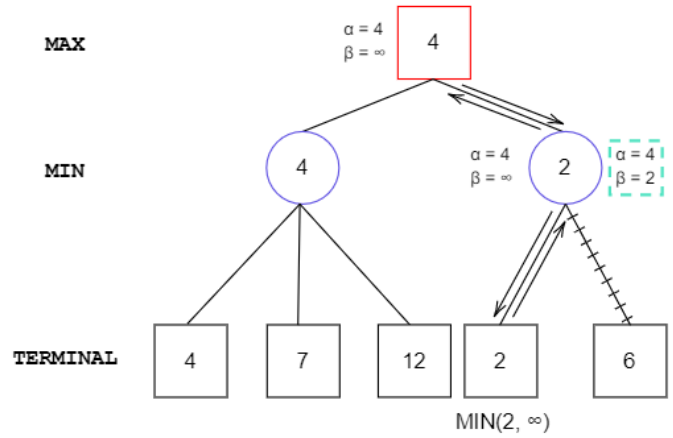
Ahora nos preguntamos, ¿es necesario recorrer todo el árbol generado por el minimax para poder tomar una decisión? Es claro que recorriendo todos los niveles del árbol encontraremos la solución óptima pero, ¿a qué precio? Un recorrido general del árbol en el que muchos nodos se abren muchas posibilidades de jugadas en sus hijos podría dar lugar a un árbol muy denso, imposible de recorrer en un tiempo adecuado, pues el tiempo necesario para bajar un nivel más va creciendo exponencialmente a lo largo del árbol. Así pues, es necesario encontrar un equilibrio entre una profundidad lo suficientemente grande como para poder hacer una estimación heurística lo suficientemente buena, y lo suficientemente pequeña como para que no se retrase demasiado la ejecución. La cuestión más importante radica entonces en encontrar dicha función heurística que nos permita

reflejar de la manera más fiel posible la idoneidad de un estado para el jugador que quiere maximizar. De esta manera, mientras que un jugador intenta maximizar este valor, el otro pretende minimizarlo. Para esto, y enlazando con lo que explicábamos anteriormente, cuanto más fiel a la realidad y más ajustada sea la función heurística en cada una de las situaciones del juego, menor profundidad será necesaria explorar.

Además, para aliviar esta situación, existen algunas optimizaciones que pueden ser añadidas al algoritmo. Afortunadamente, es posible encontrar la misma solución óptima sin necesidad de buscar en cada uno de los nodos del árbol de juego. Por eso, eliminaremos nodos añadiendo la *poda alfa-beta* al algoritmo minimax, que seguirá devolviendo el mismo movimiento que el estándar, pero elimina (poda) nodos que sabemos que no afectan a la decisión final. Utilizaremos las variables alfa y beta para indicar la mejor y peor opción de cada jugador en lo que llevamos explorado del árbol, que quiere maximizar o minimizar respectivamente. Por lo tanto, estarán inicializadas siempre al máximo y mínimo valor posible de la función heurística. Nótese que cada nodo tendrá que llevar la cuenta de cuánto valen estos valores, y que alfa solo podrá ser modificado en el turno de MAX y beta en el turno de MIN. Veamos, con un ejemplo similar al anterior, cómo funciona esta poda[1]:



(a) Exploración primera rama



(b) Poda segunda rama

Figura 5.2: Poda alfa-beta.

[1]Podemos ver más ejemplos y explicaciones acerca del algoritmo minimax con poda alfa-beta en [8].

En primer lugar, inicializamos el valor de $\alpha = -\infty$ y $\beta = \infty$ como los peores casos posibles. La condición que se debe cumplir para poder podar un nodo es que alfa se convierta en un número mayor o igual que beta. Comenzando en la raíz de nuestro árbol y con los valores de $alfa$ y *beta* inicializados, llevamos esos valores hasta el nodo hoja más a la izquierda. Ahora, desde el nodo del segundo nivel, recorremos sus estados terminales hijos y, en función de sus valores heurísticos asociados, actualizaremos los valores de alfa a 4, pues es el valor mínimo que alfa puede tener. En este punto (Figura 5.2 *(b)*), tenemos que $\alpha = 4$ y $\beta = \infty$. Llevándonos estos valores a la rama de la derecha, y calculando el $MIN(2, \infty)$ obtenemos $\alpha = 4$ y $\beta = 2$. Podemos entonces podar el resto de hijos de esta rama puesto que alfa es en este momento superior a beta, lo que implica que, sea lo que sea que encontremos en el resto de ramas, el mínimo de todos será como mucho 2, que ya es menor que α. Actualizamos $\beta = 2$ en la raíz y, como el jugador de la raíz busca maximizar, obtenemos que el valor heurístico óptimo es 4. Por lo tanto, somos capaces de encontrar el mismo valor óptimo que el minimax simple sin siquiera haber mirado en todas las hojas. Llevándonos este sencillo ejemplo a casos más complejos, donde el árbol esté compuesto por una gran cantidad de nodos con demasiados hijos, esta poda puede resultar muy útil para evitar recorrer el árbol en su completitud, preservando la optimalidad de la solución devuelta.

Sin mayor dilación, veamos la aplicación directa de este algoritmo en nuestro problema, donde cada nodo del árbol minimax reflejará una configuración determinada de nuestro juego de la propagación. En primer lugar, veremos cómo generaremos cada uno de los niveles del árbol y, una vez entendido ese proceso y su dificultad, entraremos en detalles sobre qué heurística ha sido elegida para nuestra implementación y por qué.

Cada nivel del árbol representa entonces un estado de juego de acuerdo a la Definición 3.4, unívocamente definido por el grafo (nodos y matriz de adyacencia), el conjunto de nodos alcanzados por la propagación, P, y el turno, representado por la profundidad en la que nos encontramos en el árbol y cuál de los dos jugadores lo posee. A la hora de generar los posibles movimientos que tiene cada uno de los jugadores en cada turno del minimax en función de la configuración actual del juego, disponemos de dos métodos que calculan las diferentes posibilidades que tiene cada jugador y generan las instancias del juego correspondientes a ellas.

En primer lugar, el método *generateAppeasing* calcula los diferentes cortes que puede hacer el apaciguador en su turno, primeramente calculando qué aristas tienen al menos uno de los nodos en sus extremos libres de propagación (pues cortar las demás aristas es inútil para el apaciguador) y generando por combinatoria todas las posibles formas de cortar dichas aristas en función del parámetro de entrada del juego k. Así pues, tomamos los distintos subconjuntos de aristas de tamaño k y generamos las distintas configuraciones del juego de la propagación en el que esas aristas quedasen eliminadas del grafo. Cabe destacar que, a la hora de elegir qué aristas pueden ser cortadas, no tenemos en cuenta aquellas aristas que son consideradas como irrelevantes. Ciertas aristas se consideran *irrelevantes* si son las únicas que conectan los nodos $X_1...X_r$ con Y y tenemos que $\sum_{i=1}^{r} b_{X_i} \leq a_Y$ por lo que la propagación no podría avanzar nunca a ese nodo, aunque llegara a conquistar todos los $X_1...X_r$, y el nodo Y estaría a salvo en cualquier caso. Esta decisión intenta reflejar un caso de juego real, en el que el apaciguador tiene claro que ese nodo está a salvo y no quiere perder uno de sus cortes en una arista que, aunque siga en pie, no será *relevante* a lo largo del juego.

Otra cuestión que es importante mencionar durante el proceso de selección de qué aristas podrán ser cortadas es el caso de las multiaristas. En aquellos grafos en los que existan multiaristas, es claro que la decisión que hay que tomar no es qué arista hay que cortar entre todas las que unen, por ejemplo, los nodos X e Y, puesto que son todas iguales, sino cuántas. De esta manera, al recorrer la matriz de adyacencia en busca de aristas susceptibles de ser cortadas, añadimos a nuestro conjunto de aristas libres el mínimo entre el número de cortes disponibles k y el número de aristas que haya entre el correspondiente par de nodos. Así, si solamente hay una arista conectando los nodos X e Y el mínimo será 1 pero si existen m aristas conectándoles solo hará falta estudiar los casos en los que solo eliges de 1 a $min(m, k)$ aristas dentro ese conjunto.

Por otra parte, *generatePropagation* es el método encargado de, dado el conjunto de nodos que han sido alcanzados por la propagación, encontrar aquellos nodos que pertenecen a la zona de guerra, es decir, aquellos que están en la frontera con la propagación pero aún no han sido alcanzados por ella, y calcular, a través de problemas de programación lineal, cuáles son las propagaciones factibles a dichos nodos desde los nodos que ya pertenecen a la propagación y son adyacentes a ellos. Veamos con más detenimiento este proceso, que supone la parte vertebral de la actuación de la propagación.

Una vez calculado el conjunto de nodos pertenecientes a la zona de guerra generamos el conjunto potencia de estos nodos, que representa las diferentes posibilidades que tiene la propagación para avanzar. Con cada una de estas posibilidades generamos un problema de propagación lineal. El objetivo de dicho problema es averiguar si la propagación tiene alguna manera de repartir las cargas infectivas de sus nodos de forma que, en su turno, logre colonizar exactamente la combinación de nodos objetivo considerada. Fijémonos en que solo nos interesa saber si hay alguna forma de alcanzar dicha combinación. Como las cargas sin usar no otorgan ningún valor añadido en los turnos siguientes, saber si hay varias maneras de alcanzar los mismos nodos objetivo en el turno actual es irrelevante. Como veremos luego, nos quedaremos, por ejemplo, con las más *baratas*. El problema de propagación lineal está compuesto por la función objetivo y dos tipos de restricciones. Sea entonces cierto subconjunto de nodos (fr) que pertenecen a la zona de guerra y son alcanzables desde alguno de los nodos pertenecientes a la propagación, es decir, existe alguna propagación válida en cuanto a la Definición 3.2 para ese nodo; para cada uno de estos, sea ady su lista de nodos adyacentes pertenecientes a la propagación, y sea $pr \subseteq P$ el conjunto total de nodos pertenecientes a la propagación involucrados en esta propagación en concreto (que serán todos los nodos en P que sean adyacentes a algún nodo en fr).

Construimos el problema de programación lineal de esta forma

$$\begin{aligned} \min \quad & \sum_{i,j} x_{i,j} \\ \text{sa} \quad & \forall j \in fr \quad \sum_{i \in ady(j)} x_{i,j} \geq a_j, \\ & \forall i \in pr \quad \sum_{j \in pr(i)} x_{i,j} \leq b_i, \\ & \forall i \in P, j \in fr \quad x_{i,j} \geq 0, \end{aligned}$$

donde las variables $x_{i,j}$ representarán la dosis de carga que envía el nodo perteneciente a

la propagación $i \in ady(j)$ al nodo perteneciente a la zona de guerra $j \in fr$. Cabe destacar que el primer tipo de restricciones representan la suma de cargas necesaria para poder alcanzar un nodo $j \in fr$ y, por lo tanto, es necesario sumar todas las posibles cargas de aquellos nodos adyacentes a $j \in fr$ que pertenecen a la propagación, y esta suma debe ser mayor o igual a la resistencia a_j del nodo j. De este tipo existen l restricciones, siendo l el número total de nodos que se encuentran en peligro, es decir, el cardinal de fr. Por otra parte, el segundo tipo de restricciones se centra en los limites que tiene cada uno de los nodos pertenecientes a la propagación ($i \in pr(j)$) que están relacionados con los nodos de fr para emitir carga y avanzar a estos. Por este motivo, estas restricciones consisten en la suma de las dosis de carga que emite cada nodo, que siempre tendrá que ser menor o igual al máximo del que disponen b_i.

Realmente, no estamos muy interesados en la función objetivo y su resultado tras resolver el problema sino, simplemente, en si ese problema tiene una solución factible o no, es decir, si la propagación entre esos nodos es posible o no. Aun así, nos decantamos por un problema de minimización por el simple hecho intuitivo de utilizar el menor número de carga posible que es lo que, de forma natural, podría ocurrir en una aplicación real directa y, por ese mismo motivo, la función objetivo es la suma de todas las dosis. Veamos un ejemplo para entender mejor todo esto.

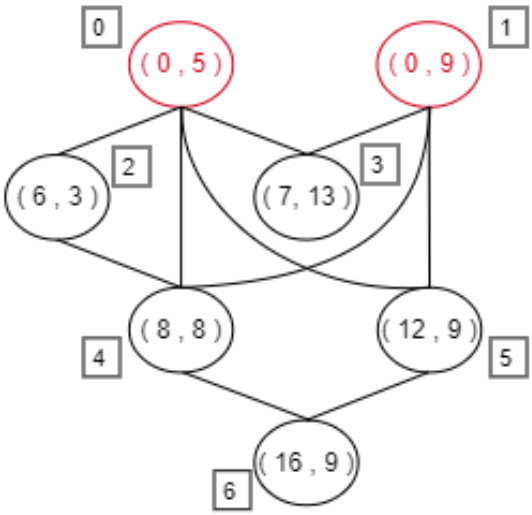


Figura 5.3: Ejemplo de programación lineal para la propagación.

En la situación que se nos presenta en la Figura 5.3, los dos nodos que pertenecen a la propagación son los nodos 0 y 1, $P = \{0, 1\}$, mientras que los nodos que pertenecen a fr son el resto, pues todos están relacionados al menos con uno de los nodos de la propagación. Centrándonos en una de las posibles propagaciones, en concreto la de mayor cardinal, vamos a ver el desarrollo del problema de programación lineal para comprobar la factibilidad de la propagación. La lista ady está compuesta por otras listas que nos indican con qué nodos de la propagación están relacionados cada uno de los nodos de fr. Así pues, para el nodo 2, $ady(2) = \{0\}$ mientras que para el nodo 5 $ady(5) = \{0, 1\}$. De esta manera, comprobaremos si a cada nodo de la frontera es posible que le llegue la suficiente carga para ser alcanzado por la propagación (por ejemplo, en el caso del nodo 6 esto sería $x_{0,5} + x_{1,5} \geq 12$). También, tendremos que comprobar que la suma de carga que expulsa cada nodo de la propagación no es superior a su capacidad de propagación,

esto es, por ejemplo para el nodo 2, $x_{1,3} + x_{1,4} + x_{1,5} \leq 9$. Uniendo todas las restricciones del ejemplo quedaría el siguiente problema de programación lineal:

$$\begin{array}{ll} \min & x_{0,2} + x_{0,3} + x_{0,4} + x_{0,5} + x_{1,3} + x_{1,4} + x_{1,5} \\ \text{sa} & x_{0,3} \geq 6 \\ & x_{0,3} + x_{1,3} \geq 3 \\ & x_{0,4} + x_{1,4} \geq 8 \\ & x_{0,5} + x_{1,5} \geq 11 \\ & x_{0,2} + x_{0,3} + x_{0,4} + x_{0,5} \leq 5 \\ & x_{1,3} + x_{1,4} + x_{1,5} \leq 9 \\ & x_{i,j} \geq 0, \forall i \in P, j \in fr \end{array}$$

Es claro que este ejemplo resulta infactible pues, con solo fijarnos en el nodo 2, es imposible que se satisfaga $x_{0,2} \geq 5$ cuando $x_{0,2} + x_{0,3} + x_{0,4} + x_{0,5} \leq 5$. Sin embargo, simplificando el intento de propagación a un conjunto de nodos más pequeño, $fr = \{3, 4\}$, el problema de programación lineal quedaría

$$\begin{array}{ll} \min & x_{0,3} + x_{0,4} + x_{1,3} + x_{1,4} \\ \text{sa} & x_{0,3} + x_{1,3} \geq 3 \\ & x_{0,4} + x_{1,4} \geq 8 \\ & x_{0,3} + x_{0,4} \leq 5 \\ & x_{1,3} + x_{1,4} \leq 9 \\ & x_{i,j} \geq 0, \forall i \in p, j \in fr \end{array}$$

que es factible con esta solución: $x_{0,3} = 3, x_{0,4} = 2, x_{1,3} = 0, x_{1,4} = 6$.

En nuestro código, hemos utilizado *pyomo*[9], un paquete de Python que permite formular, solucionar y analizar un variado conjunto de modelos de optimización. Uno de sus principales usos es definir problemas generales de optimización lineal, crear instancias del problema y solucionar dichas instancias usando solucionadores de código libre. Así pues, utilizando *pyomo* conseguimos averiguar la factibilidad de cada una de las propagaciones posibles. En el caso en que la propagación sea factible, se dejan de explorar aquellas propagaciones que constituyan un subconjunto de la misma. Es decir, si por ejemplo, como veíamos en nuestro ejemplo anterior, la propagación a los nodos $\{3, 4\}$ es factible, entonces es inútil comprobar que es factible propagarse solo al nodo 3 o al nodo 4 (pues para la propagación nunca será mejor estratégicamente propagarse solo al nodo 3, pudiéndose propagar a ambos). Por este motivo, solo contemplamos el conjunto de propagaciones factibles con mayor cardinal (y no los subconjuntos de los mismos), con los que generamos las instancias de posibles jugadas de la propagación y pasamos al siguiente nivel en nuestro árbol del minimax.

Una vez entendido el proceso de generación de los hijos de cada uno de los nodos de nuestro árbol minimax, queda claro que, para instancias en las que haya una gran cantidad de nodos y aristas, la complejidad de recorrer el árbol en su totalidad aumenta exponencialmente. En concreto, si tenemos r aristas en una configuración concreta y podemos hacer k cortes, las instancias a explorar que genera el método *generateAppeasing()* son $\binom{r}{k} = \frac{r!}{k!(r-k)!}$. Asimismo, si podemos propagarnos a s nodos distintos, deberemos comprobar 2^s combinaciones posibles, y en función de cuántas sean factibles, ese será el número de nodos a explorar (al menos s). Así pues, es necesario encontrar una función heurística

lo suficientemente buena como para poder limitar la profundidad del árbol según nuestra conveniencia.

Como hemos ido explicando en párrafos anteriores, la función heurística debe ser capaz de representar de la manera más fiel posible la idoneidad de un estado para el jugador. Para ello y de acuerdo con nuestra representación del juego, la función debe otorgar un valor a cada uno de los estados de juego, que dependa de la configuración del mismo y represente un valor a minimizar por el apaciguador y un valor a maximizar por la propagación. Con esto en mente, una primera aproximación a una función heurística sencilla podría ser la suma de las capacidades de propagación del conjunto P. Así, mientras que la propagación estaría interesada en que este valor creciese, lo que implicaría antes o después su victoria, el apaciguador tendrá que hacer todo lo posible para que ocurra lo contrario. El problema de esta sencilla heurística es precisamente eso, que quizás es demasiado sencilla y no es capaz de diferenciar ciertos casos que pueden resultar cruciales en nuestra exploración como, por ejemplo los *cuellos de botella*.

Un ejemplo de cuello de botella serían, en la Figura 4.1, aquellos nodos puente que comunicaban los nodos que representaban las variables universales con las existenciales de la siguiente iteración. De esta manera, si conseguimos salvar estos nodos que constituyen un cuello de botella, evitaríamos que la propagación alcanzase otro conjunto de nodos que podría ser fácilmente salvado. Para conseguir representar este concepto de forma numérica y así poder utilizarlo en nuestra heurística, utilizaremos la noción de *intermediación* ("Betweenness centrality" [10]), una medida que cuantifica la frecuencia o el número de veces que un nodo actúa como un puente a lo largo del camino más corto entre otros dos nodos. La idea intuitiva para entender el algoritmo empleado [11] es que si se eligen dos nodos al azar, y luego también al azar uno de los eventuales posibles caminos más cortos entre ellos, entonces los nodos con mayor intermediación serán aquellos que aparezcan con mayor probabilidad dentro de este camino. En complejidad computacional, determinar el camino más corto para cada par de nodos de un grafo $G = (V, A)$ sin pesos, como en nuestro caso, se puede lograr en tiempo $O(V * A)$ y espacio $O(|V| + |E|)$ utilizando el *algoritmo de Brandes* [12].

Así, utilizando el algoritmos de Brandes asignamos a cada nodo v del grafo G un valor b_v que representa su intermediación, y posteriormente lo normalizamos para que quede comprendido entre 0 y 1, convirtiéndose en un factor muy útil para su uso dentro de la heurística, pues representa la conectividad de aquellos nodos pertenecientes a la propagación dentro de la zona de guerra. Pero no solo es importante la intermediación que tengan estos nodos, sino también cuántos son y cuánta carga son capaces de transmitir. La heurística para una configuración del juego de la propagación, (G, P, i), queda entonces determinada por:

$$h(v) = (|P|^2) * m_P * b_v$$

donde utilizamos el cuadrado para recalcar la importancia que tiene el número de nodos que posea la propagación en nuestro estudio, pues es lo que decidirá quién se llevará la victoria.

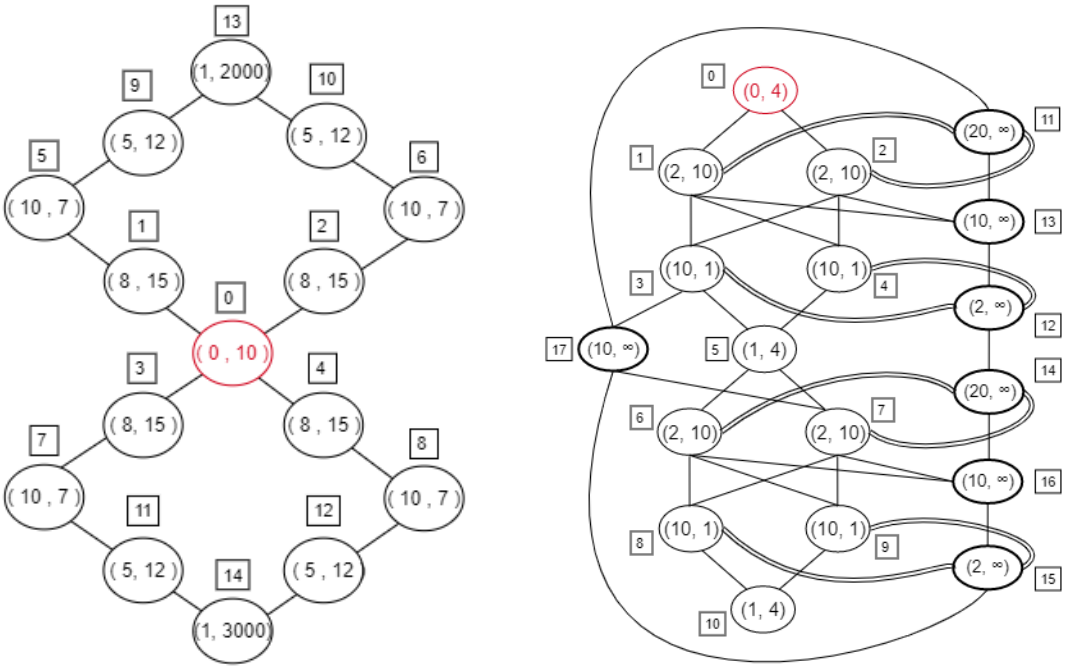
En consecuencia, el jugador cuyo objetivo es maximizar el valor heurístico (la propagación) estará interesado en aquellos nodos que tengan una intermediación muy alta y que, a su vez tengan una alta carga (para poder retransmitir al resto una vez haya sido

infectado) mientras que, en el lado contrario, el apaciguador intentará aislar estos nodos para poder frenar el avance de la propagación y minimizar el número de nodos que son alcanzados por la misma.

En nuestro caso, al algoritmo minimax con poda alfa-beta le proporcionamos un factor, llamémosle σ, del cual dependerá directamente la profundidad a la que el algoritmo llegue en el árbol y que será establecido a conveniencia por el programador. Esta decisión se basa únicamente en la intención de no tener que alterar el código del algoritmo minimax en función de cada ejemplo. Así, la profundidad a la que llega nuestro algoritmo es el primer número entero mayor que $\sigma * (|V| - s)$.

Otra de las medidas tomadas para intentar optimizar el funcionamiento de nuestro algoritmo minimax y evitar que tenga que recorrer profundidades innecesarias es ordenar los nodos hijos generados por las funciones *generateAppeasing()* y *generatePropagation()* por orden creciente y decreciente de su valor heurístico asociado. Así, favorecemos que la actualización del alfa y beta se hagan cuanto antes, evitando explorar ramas innecesarias que no aportarían ningún valor a nuestro problema.

Una vez entendido su funcionamiento, veamos el rendimiento de nuestro algoritmo con algunos ejemplos. Para ello, nos fijaremos en los aspectos clave que nos indicarán cómo de bueno ha sido su rendimiento cuando ponemos a jugar a un jugador contra otro, realizando las mejores jugadas posibles en ambos casos. En primer lugar, tenemos el ejemplo ya visto en la Figura 5.3, donde vemos un grafo en el que la propagación surge de los nodos 0 y 1. Con este ejemplo y con diferentes valores para los parámetros *syk*, obligamos al apaciguador a realizar su mejor jugada pues, de lo contrario, perdería al no salvar los suficientes nodos.



(a) Ejemplo 2 (b) Ejemplo TQBF

Figura 5.4: Ejemplos utilizados para estudiar el rendimiento.

En segundo lugar, con el primer ejemplo que aparece en la Figura 5.4 intentamos com-

probar que efectivamente el algoritmo es capaz de adelantarse con suficiente antelación a los movimientos del oponente, teniendo que apreciar que existen dos nodos vitales, con una alta carga de propagación, que es necesario salvar. Otro ejemplo que hemos utilizado para medir la eficacia de nuestro algoritmo es el que podemos observar en la Figura 5.4, donde podemos ver un claro ejemplo de la estructura que hemos construido para la demostración de complejidad del Capítulo 4. En la imagen omitimos algunas de las aristas que relacionan todos los nodos bomba entre sí, para facilitar la visibilidad de los nodos en un grafo tan árido. Además, en la implementación de éste estableceremos 500 como valor concreto de la capacidad de propagación de los nodos bomba.

Veamos las comparaciones entre tiempo y profundidad de cada uno de los ejemplos en la siguiente tabla, así como un resumen de los parámetros y decisiones tomadas sobre cada uno de ellos. Como hemos explicado anteriormente, todas las ejecuciones se realizan desde el servicio Cloud, Colaboratory de Google, pudiéndose encontrar la implementación y algunos de los siguientes ejemplos ahí [7]. Todas las ejecuciones se realizan siempre con los mismos recursos, que se asignan independientemente de la sesión con la que se acceda, lo que nos permite poder hacer comparaciones entre distintas instancias en función del tiempo.

	s	k	Profundidad total $\sigma * (\|V\| - s)$	Tiempo	Solución óptima
Ejemplo 1	3	1	3	1,04s	✓
		2	3	2,91s	✓
	4	1	3	0,91s	✓
		2	3	2,44s	✓
	5	3	2	0,05s	✗
			3	5,58s	✓
Ejemplo 2	2	1	3	1m 10s	✗
			4	52s,65	✓
	10	1	3	5s	✗
			4	9s	✓
	12	2	2	20s	✗
			3	25s	✓
Ejemplo TQBF	11	1	4	8s	✗
			5	12min 53s	✓
			6	34minn 12s	✓

Cuadro 5.1: Resumen del rendimiento.

En el Ejemplo 1 (Figura 5.3), nos dimos cuenta tras probar con las distintas instancias resumidas en la Tabla 5.1 que el factor σ tendía a aquel número que permitiera al algoritmo bajar al menos dos niveles, es decir, la profundidad necesaria para encontrar la solución óptima para este ejemplo. En este caso $\sigma * (|V| - s) = \sigma * (7 - s) > 2$, por lo que tendríamos que $\sigma > \frac{2}{(7-s)}$, pues $7 - s$ es mayor que 0 al no poder ser s superior al número de nodos totales. Nótese que es un mayor estricto y no valdría un valor de σ que hiciese que el algoritmo solo bajase dos niveles. Cabe destacar que observando la tabla podemos ver la importancia de lo expuesto pues, en la instancia en la que $s = 5$ y $k = 3$ cuando bajamos solo dos niveles de profundidad, el algoritmo no consigue encontrar la solución óptima pero, cuando bajamos tres, el apaciguador logra encontrar el movimiento óptimo

para vencer a la propagación.

En el caso del Ejemplo 2 (Figura 5.4), ocurre lo mismo que en el caso anterior aunque, por tratarse de un ejemplo más complejo, necesitaríamos que el algoritmo descendiese al menos hasta el nivel 4 para poder encontrar la solución óptima. De esta forma, si $\sigma > \frac{4}{(17-s)}$ conseguiremos que desde la primera llamada al algoritmo los jugadores sean conscientes de la existencia de los nodos 13 y 14, que son de vital importancia debido a su carga, y podrán elegir su estrategia en función de ese conocimiento. De nuevo, se vuelve a poner de manifiesto la necesidad de alcanzar la profundidad necesaria.

Por ultimo, para el ejemplo que refleja la reducción polinómica de nuestro ejemplo de la instancia relacionada con el problema `TQBF` (Figura 5.4), tomaremos como ya vimos $s = \frac{5p}{2} + q = 11$ y $k = 1$. La salida de nuestro algoritmo es la siguiente, que concuerda con la construcción realizada en el Capítulo 4:

```
Turno del apaciguador, que corta las aristas: [(0, 1)]
Turno de la propagación, que se propaga a los nodos: {2}
Turno del apaciguador, que corta las aristas: [(2, 13)]
Turno de la propagación, que se propaga a los nodos: {3}
Turno del apaciguador, que corta las aristas: [(2, 4)]
Turno de la propagación, que se propaga a los nodos: {5}
Turno del apaciguador, que corta las aristas: [(5, 7)]
Turno de la propagación, que se propaga a los nodos: {6}
Turno del apaciguador, que corta las aristas: [(6, 16)]
Turno de la propagación, que se propaga a los nodos: {8}
Turno del apaciguador, que corta las aristas: [(6, 9)]
Turno de la propagación, que se propaga a los nodos: {10}
Turno del apaciguador, que corta las aristas: [(4, 5)]
Fin de juego
El ganador es el apaciguador.
```

Figura 5.5: Algoritmo Minimax aplicado a la instancia TQBF.

Una vez más, nos damos cuenta que una profundidad no suficiente hace que la respuesta del algoritmo no sea la indicada, mientras que una profundidad superior a la necesaria hace que nuestro algoritmo se demore demasiado. Por este motivo, recalcamos la importancia de la elección apropiada de la profundidad para cada uno de los casos, siendo estrictamente necesario el análisis previo del problema para realizar la elección correcta.

Observando la tabla, podemos apreciar que a medida que aumenta la complejidad de la instancia, ya sea por la aridad del grafo, por la profundidad del árbol que hay que alcanzar o por la elección de los parámetros, el tiempo aumenta en una cantidad considerable. Para intentar aliviar este problema, hemos añadido a la implementación un fichero *log* en el que se irá actualizando cuál es la mejor jugada hasta el momento para el jugador correspondiente mientras el algoritmo se ejecuta. Así, en una situación real en la que no se dispusiera del tiempo necesario para esperar a la solución óptima, podemos utilizar la mejor encontrada hasta el momento.

Viendo nuestra implementación, de nuevo podemos asegurar la facilidad con la que podríamos generalizar nuestro problema y en particular dicha implementación, a otros casos que podría resultar interesante de analizar. Una de las generalizaciones de las que hablábamos anteriormente en el Capítulo 3 era la posibilidad de *ponderar las aristas* para que el apaciguador tuviera una cierta cantidad de puntos por turno para gastar en cortar aristas, en lugar del parámetro k de aristas que puede cortar. Llevar esta generalización

a nuestro código supondría un cambio en nuestra representación de las aristas, acompañando la matriz de adyacencia con un diccionario que relacionase cada arista con su valor, e. Además, el parámetro k pasaría a determinar la cantidad de peso que es capaz el apaciguador de cortar. De esta manera, una vez generados por combinatoria los distintos subconjuntos de aristas que se pueden cortar y comprobando de manera sencilla si esos cortes son factibles (sumando el peso asociado a cada arista del subconjunto), podríamos seguir la misma estrategia que al generar las jugadas de la propagación: seleccionar únicamente los subconjuntos maximales factibles.

Por otra parte, otra generalización interesante a estudiar giraba en torno a la idea de asociar a cada nodo una *fecha de expiración*, es decir, una cantidad de turnos durante los cuales dicho nodo es capaz de propagarse hacia otros una vez ha sido alcanzado por la propagación. La traducción de este requisito a nuestro código es tan simple como añadir un atributo, ttl, a la tupla que guarda la información de cada nodo y sustituir el conjunto P por un diccionario que relacione el id (único) de cada nodo con el ttl que le queda. Una vez la propagación alcance a cada nodo, se añadiría una nueva entrada al diccionario y después de cada turno todos los ttl disminuirían en una unidad, eliminando aquellos que quedasen consumidos ($ttl = 0$).

Capítulo 6

Caso de estudio: Comunidades Autónomas de España

En este capítulo presentaremos un caso de estudio donde el problema propuesto y los métodos que hemos utilizado para resolverlos pueden ser aplicados a un entorno realista. Después de introducir la implementación heurística en el capítulo anterior, reproducir un caso real nos ayudará a determinar cuáles son las dificultades y retos a los que realmente nos estamos enfrentando, relacionados con la gran cantidad de posibles jugadas que aparecen en cada turno. El entorno real que desarrollamos en esta sección nos obligará a resolver problemas que no aparecían con los casos artificiales que veíamos anteriormente, como la inicialización del problema y cómo y en función de qué variables establecer de forma óptima los valores (a, b) de cada uno de los nodos del grafo.

Nótese que a pesar de que el concepto de propagación pueda sonar directamente relacionado a fuerzas de la naturaleza como incendios o enfermedades, puede tener aplicaciones muy variadas en otros entornos, tales como evitar la propagación de *fake news* o la filtración de información sensible en Internet y redes sociales, que un virus informático destruya completamente información importante en un sistema o que un ejército enemigo alcance el castillo de la ciudad y logre destronar al rey. Aun así, aplicaremos nuestro algoritmo a un caso de estudio a la orden del día, la pandemia *COVID-19* en nuestro país, España.

Con ello, encontramos una aplicación directa de todos los conceptos desarrollados hasta la fecha. Para construir una instancia de nuestro juego de la propagación comenzamos por el grafo de propagación compuesto por nodos que, en este caso, representarán las capitales de las comunidades autónomas de España, y la matriz de adyacencia que relacionará dos capitales si el territorio de sus respectivas comunidades es colindante o, en caso de las islas y ciudades autónomas, pertenecen a aquellas comunidades más cercanas a las mismas. Además, uniremos Cataluña y la Comunidad de Madrid por ser el viaje entre sus capitales, Barcelona y Madrid, el que con más frecuencia se realiza de forma nacional. Así pues, el mapa de las capitales de comunidades autónomas españolas adaptado al juego de la propagación quedará como puede observarse en la Figura 6.1.

Está claro que la propagación se refiere a la propagación de la pandemia por los territorios españoles y el apaciguador podría ser representado por todo el equipo sanitario que se encarga de intentar frenar el avance de la misma, a través de ciertos métodos que

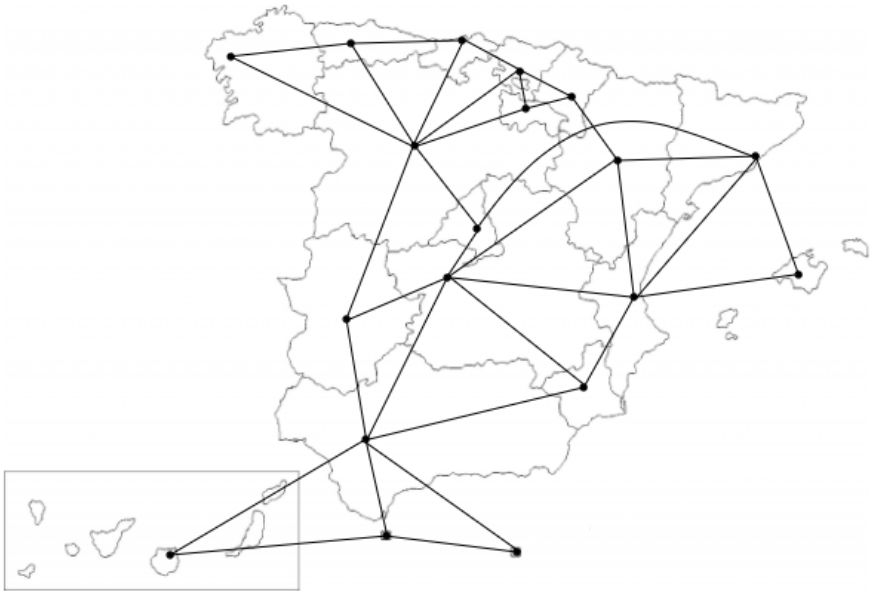


Figura 6.1: Mapa de comunidades autónomas.

pueden asemejarse al corte de aristas, como puede ser el confinamiento de ciertas zonas perimetrales o las cuarentenas obligatorias de pacientes positivos.

Para cada una de las comunidades autónomas, elegiremos los valores de (a, b) de forma acorde a la situación. Cabe destacar la importancia de que dichos valores deben ser cuidadosamente elegidos, en función de determinadas variables, para que estén compensados, es decir, que los valores de a y b para cada una de las comunidades deben estar dentro del mismo rango de valores y no existir grandes desigualdades pues, de lo contrario, no sería posible poder reflejar el funcionamiento de la propagación en el grafo. Para el caso de a, necesitamos encontrar un valor que represente la capacidad que tiene una comunidad autónoma para defenderse del intento de contagio del resto. De esta manera, lo más sencillo que se nos puede ocurrir que mida este criterio es el Presupuesto de Sanidad que cada una de ellas tuvo asignado en el 2020[1], presupuesto que mide la cantidad y calidad de recursos que cada una de ellas tuvo para poder hacer frente a la pandemia. Por otro lado, para escoger el valor de b en cada caso nos enfocamos en buscar un dato que refleje una de las principales causas de contagio que se dieron en Marzo de 2020: los desplazamientos. Para ello, tomamos los Porcentajes de Movilidad hacia otras comunidades autónomas[2] de cada una de ellas en dicho mes y lo aplicamos a la cantidad total de habitantes para estimar un número concreto de personas que se desplazaron fuera de su comunidad. Debido a que los datos relacionados con b están dirigidos a un periodo de tiempo menor, dividiremos los datos de los Presupuestos Anuales en 12 meses para que representen solo el presupuesto mensual. Además, la elección final de a y b quedará condicionada porque, con la intención de favorecer los cálculos de nuestro algoritmo, decidamos no utilizar números tan grandes, dividiendo los datos finales por 1000 y aproximando según conveniencia los mismos. A continuación, podemos ver en la Tabla 6.1 todos estos datos reflejados así como la elección definitiva de a y b para cada autonomía y el mapa de España en la Figura 6.2 en forma de grafo con los datos asociados.

[1]Datos extraídos del Ministerio de Sanidad [13].

[2]Datos extraídos del Estudio de movilidad a partir de la telefonía móvil realizado en Marzo de 2020 por el Instituto Nacional de Estadística [14].

Comunidad Autónoma	Número de habitantes	Presupuesto de Sanidad Anual	Presupuesto de Sanidad Mensual	Porcentaje de movilidad mensual	Movilidad mensual aplicada a habitantes	a	b
Andalucia	8.464.411	10.824.520	902.043	11,32	958171,3252	90	96
Aragón	1.329.391	2.061.799	171.817	13,21	175612,5511	17	18
Asturias	1.018.784	1.815.603	151.300	13,07	133155,0688	15	13
Baleares	1.171.543	1.724.941	143.745	12,29	143982,6347	14	14
Canarias	2.175.952	3.132.986	261.082	14,18	308549,9936	26	31
Cantabria	582.905	922.059	76.838	14,53	84696,0965	8	8
Castilla y León	2.394.918	3.534.049	294.504	12,5	299364,75	30	30
Castilla - La Mancha	2.045.221	2.990.643	249.220	10,56	215975,3376	25	22
Cataluña	7.780.479	9.733.585	811.132	13,77	1071371,958	81	107
Comunidad Valenciana	5.057.353	6.732.209	561.017	14,58	737362,0674	56	74
Extremadura	1.063.987	1.742.520	145.210	10,08	107249,8896	15	11
Galicia	2.701.819	4.107.324	342.277	15,78	426347,0382	35	42
Madrid	6.779.888	8.165.992	680.499	13,87	940370,4656	68	94
Murcia	1.511.251	1.922.557	160.213	12,36	186790,6236	16	19
Navarra	661.197	1.158.559	96.547	13,73	90782,3481	10	9
País Vasco	2.220.504	3.978.448	331.537	16,37	363496,5048	33	36
La Rioja	319.914	459.647	38.304	12,11	38741,5854	4	4
Ceuta	84.202	7.566	631	12,32	10373,6864	1	5
Melilla	87.076	1.171	98	12,49	10875,7924	1	5

Cuadro 6.1: Datos relacionados con la instancia.

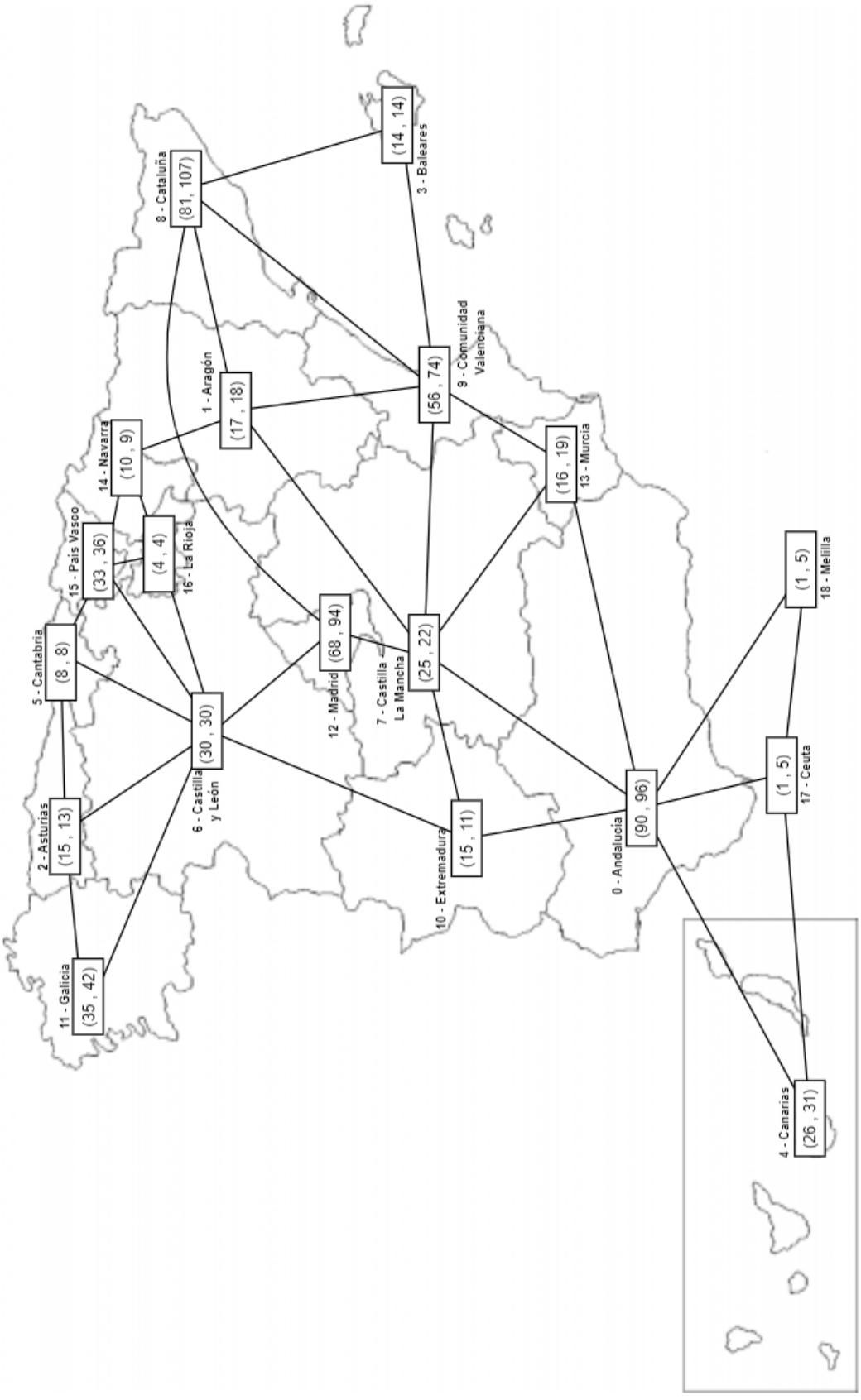


Figura 6.2: Mapa de comunidades autónomas con valores asociados.

Cabe resaltar que escogemos para Ceuta y Melilla el valor de 5, cuando debería ser 1, para aliviar la diferencia entre Comunidades y Ciudades Autónomas y darles la oportunidad para poder contribuir al grafo.

En cuanto al resto de factores involucrados en el juego, s, k, quedarán determinados por conveniencia en base a datos extraídos también del Instituto Nacional de Estadística. En el caso de s, tomaremos distintos valores de para estudiar el rendimiento de nuestro algoritmo, teniendo en cuenta que, por ejemplo, $s = 9$ representaría salvar al menos al 50 % de las comunidades autónomas. Por otra parte, como hemos explicado anteriormente, k representa los cortes que puedan hacer los sanitarios. Un único corte podría parecer inútil en un grafo con tanta cantidad de aristas, pero por el contrario poner un k demasiado elevado, llevándolo a nuestro caso real aislando muchas comunicaciones entre regiones, podría llevar al enfado de la población debido a las altas restricciones. Por consiguiente y teniendo en cuenta lo difícil que es coordinar asuntos sanitarios entre comunidades, un k justo y beneficioso para ambos jugadores podría ser un 10 % del número de aristas totales, es decir, $k = 3$.

Comunidades en las que comienza la propagación	s	k	Profundidad	Tiempo	Solución óptima para el apaciguador
4- Canarias, 9 - Comunidad Valenciana	9	2	3	26s	✓
			4	1m 32s	✓
		3	3	2min 21s	✓
			4	59min 18s	✓
4- Canarias, 9 - Comunidad Valenciana, 12 - Madrid	9	2	3	2min 8s	✓
			4	2min 30s	✓
		3	3	24min 25s	✓
			4	48min 13s	✓
9 - Comunidad Valenciana, 12 - Madrid, 15 - País Vasco	9	2	4	3min 5s	✗
		3	4	45min34s	✓
0 - Andalucía, 1 - Aragón, 11 - Galicia	9	2	4	2h 59min	✗
		3	4	2h 53min	✗
6 - Castilla y León, 7 - Castilla-La Mancha	9	2	3	5min 11s	✓
			4	7min 3s	✓
12 - Madrid	9	1	4	9s	✗
			5	10min14s	✓

Cuadro 6.2: Resumen del rendimiento caso natural.

A la hora de elegir en qué Comunidad Autónoma comenzaba la propagación nos hemos decantado, en primer lugar, por los lugares en los que realmente aparecieron los primeros casos en España (Canarias, Comunidad Valenciana y Madrid) y, en otros casos, en los lugares en los que mayor incidencia ha tenido el COVID o aquellos lugares estratégicos que nos interesaban para estudiar el rendimiento del algoritmo (por ejemplo, Andalucía o País Vasco). En consecuencia, presentamos en la Tabla 6.2 los datos relativos a las distintas instancias de nuestro caso de estudio, tomando como parámetros de propagación $s = 9$, $k \in \{2, 3\}$ y ajustando σ para que el algoritmo baje 3, 4 o 5 niveles según conveniencia del estudio. Enfrentamos a la propagación de la COVID-19 contra el apaciguador, donde ambos realizarán su movimiento óptimo aplicando el algoritmo minimax con poda alfa-beta turno a turno.

Como podemos observar en la tabla, la topología del grafo está decantada a favorecer al apaciguador pues, sin mayor complicación, consigue la victoria en la mayoría de las configuraciones. Aun así, vemos detalles interesantes en algunas instancias como en las que la propagación comienza en los nodos 9, 12 y 15 (Comunidad Valenciana, Madrid y País Vasco, respectivamente). En esos casos, cuando la $k = 2$ el apaciguador no consigue frenar el avance de la propagación mientras que cuando $k = 3$, es decir, cuando el apaciguador tiene más recursos, sí que logra detenerla, salvando la mitad de las comunidades. Esto puede reflejarse en la realidad fácilmente, pues si desde el comienzo de la pandemia las autoridades hubiesen podido "cortar" de alguna forma más conexiones quizás el daño hubiese sido menor.

A pesar de que en muchas ocasiones, y en distintas profundidades, el algoritmo consiga encontrar una estrategia para que el apaciguador logre la victoria, la calidad de estas estrategias depende directamente de la profundidad a la que baje el algoritmo. Por ejemplo, en el caso en el que la propagación comienza en Canarias, Comunidad Valenciana y Comunidad de Madrid ocurre esto. Mientras que el algoritmo minimax con profundidad 3 logra una estrategia victoriosa que se desarrolla en cinco turnos, el algoritmo con profundidad 4 logra que el apaciguador gane en solo tres turnos, como podemos ver en la Figura 6.3.

```
Turno del apaciguador, que corta las aristas: [(1, 9), (6, 12), (8, 12)]
Turno de la propagación, que se propaga a los nodos: {17, 3, 13, 7}
Turno del apaciguador, que corta las aristas: [(0, 7), (7, 10), (1, 14)]
Turno de la propagación, que se propaga a los nodos: {8, 1, 18}
Turno del apaciguador, que corta las aristas: [(5, 6), (6, 15), (15, 16)]
Fin de juego
El ganador es el apaciguador.
```

(a) Estrategia con profundidad 3

```
Turno del apaciguador, que corta las aristas: [(7, 10), (6, 12), (4, 17)]
Turno de la propagación, que se propaga a los nodos: {1, 3, 7, 8, 13}
Turno del apaciguador, que corta las aristas: [(0, 7), (0, 13), (1, 14)]
Fin de juego
El ganador es el apaciguador.
```

(b) Estrategia con profundidad 4

Figura 6.3: Ejemplo Canarias, Comunidad Valenciana y Comunidad de Madrid.

Otro detalle a destacar, que no aparece en nuestra tabla resumen, radica en la elección de $s = 9$, que como hemos mencionado antes, representa la mitad de las comunidades. Probando con otras instancias en las que s es un valor menor, dichas instancias dejan de ser interesantes pues, de nuevo, la victoria se convierte en asequible para el apaciguador debido a la topología del grafo.

Asimismo, viendo los ejemplos podemos comprobar que aquellas comunidades con un alto presupuesto en Sanidad se han mantenido inexpugnables en todo momento, no siendo alcanzadas a no ser que la propagación comenzase en ellas. Este hecho también favorece lo que explicábamos anteriormente, que haya ciertos nodos con valores de a muy altos respecto a las b de otros hace que a la propagación le sea imposible avanzar. Además, puesto que hemos estimado ambos valores a partir de unidades diferentes, es difícil cuantificar cuántos puntos de b son necesarios para contrarrestar cada punto de a. Por este motivo, con intención de explorar nuevos casos más interesantes en los que los valores del grafo estuvieran un poco más *equilibrados* para ambos jugadores, decidimos reducir el valor de a de todas las comunidades por un factor común elegido a conveniencia, en

este caso multiplicando por $0,6$, favoreciendo el avance de la propagación.

En la siguiente tabla aparecen dos instancias representativas que han cambiado su comportamiento a raíz de la modificación de los valores de a, mostrándonos la importancia de la elección correcta de las profundidades del algoritmo minimax en cada caso.

Comunidades en las que comienza la propagación	s	k	Profundidad	Tiempo	Solución óptima para el apaciguador
4- Canarias, 9 - Comunidad Valenciana, 12 - Madrid	9	2	3	18s	✗
			4	53s	✓
6 - Castilla y León, 7 - Castilla-La Mancha	9	2	3	54,31	✗
			4	57min 45s	✓

Cuadro 6.3: Resumen del rendimiento con a modificado.

En ambos casos, al igual que ocurría con la instancia en la que la propagación comenzaba en la Comunidad de Madrid en la Tabla 6.2, la elección correcta de la profundidad es crucial. Mientras que con profundidades bajas el algoritmo no encuentra una solución óptima para el apaciguador en dichas instancias, con aumentar la profundidad un nivel más consigue hacerlo sin repercutir notablemente en el tiempo. En la siguiente figura, podemos observar los diferentes movimientos que realiza el apaciguador en función a la profundidad que se explora en el árbol minimax, en la instancia en la que la propagación comienza en Castilla y León y Castilla-La Mancha.

```
Turno del apaciguador, que corta las aristas: [(6, 12), (6, 16)]
Turno de la propagación, que se propaga a los nodos: {1, 10, 11, 13}
Turno del apaciguador, que corta las aristas: [(0, 10), (6, 15)]
Turno de la propagación, que se propaga a los nodos: {9, 2, 5, 14}
Turno del apaciguador, que corta las aristas: [(3, 9), (8, 9)]
Turno de la propagación, que se propaga a los nodos: {16}
Fin de juego
El ganador es la propagación.
```

(a) Estrategia con profundidad 3

```
Turno del apaciguador, que corta las aristas: [(1, 7), (6, 12)]
Turno de la propagación, que se propaga a los nodos: {5, 10, 11, 13, 16}
Turno del apaciguador, que corta las aristas: [(7, 9), (0, 10)]
Turno de la propagación, que se propaga a los nodos: {2, 15}
Turno del apaciguador, que corta las aristas: [(0, 7), (7, 12)]
Turno de la propagación, que se propaga a los nodos: {14}
Turno del apaciguador, que corta las aristas: [(1, 14), (0, 17)]
Fin de juego
El ganador es el apaciguador.
```

(b) Estrategia con profundidad 4

Figura 6.4: Ejemplo Castilla y León y Castilla-La Mancha.

Como reflejan los datos obtenidos, aunque sea de una forma teórica y no representativa al completo de la realidad, buscando las estrategias óptimas y llevándolas a cabo en el momento apropiado es posible frenar las propagaciones en contextos tan actuales como la COVID-19. Si trasladásemos este caso de estudio desde la gran escala de las comunidades autónomas a escalas más pequeñas como miembros de una familia, amigos o la población de un municipio, conseguiríamos una herramienta capaz de determinar la mejor estrategia en función de ciertas variables para frenar el avance de la COVID-19 en nuestras vidas.

Como decíamos al principio de este escrito, en la realidad están involucradas innumerables variables no solo biológicas sino también sociales a la hora de tomar decisiones precisas y óptimas, por lo que cualquier modelo usado en una situación real debería tener en cuenta muchos más factores.

Capítulo 7

Conclusión

En este estudio hemos definido formalmente el Juego de la Propagación, haciendo especial hincapié en buscar una estrategia exitosa para el apaciguador, figura encargada de frenar el avance de la misma, salvando un mínimo de nodos de su alcance. Hemos probado que este problema es `PSPACE-completo`, probando su intratabilidad a través de una reducción polinómica desde el problema `TQBF`. La intratabilidad del problema nos ha motivado a solucionarlo aplicando métodos heurísticos, en particular, el algoritmo minimax.

Hemos desarrollado experimentos prácticos, utilizando tanto ejemplos artificiales como un caso de estudio basado en datos reales extraídos de la crisis de la COVID-19 en España. Estos ejemplos nos han permitido estudiar el rendimiento del algoritmo minimax con poda alfa-beta aplicado a nuestro problema, reflejando la complejidad del mismo y el rigor con el que hay que escoger sus parámetros. Esto ocurre especialmente con la profundidad que explora el minimax, pues de ella depende completamente si el algoritmo es capaz de encontrar la solución óptima, debiendo estudiar con cuidado las peculiaridades de cada ejemplo.

Nuestro caso de estudio expuso una de las principales dificultades a la hora de utilizar datos reales: la elección de los valores (a, b) que representan cada uno de los nodos. En el caso en el que no exista una equilibrada elección de estos valores, la partida quedará completamente decantada hacia uno de los dos jugadores. Esta es una de las principales limitaciones de nuestro modelo, que es incapaz de representar la gran cantidad de variables que influyen en el avance de una propagación, desde biológicas a sociales. A pesar de esto, esperamos que nuestros resultados sean interesantes para otros investigadores y constituyan un paso más en el estudio de las propagaciones y cómo frenarlas.

Con respecto a posible trabajo futuro, aparte de continuar mejorando nuestra implementación, desarrollando algoritmos más sofisticados y eficientes que nos permitan representar de la mejor manera posible la realidad, urge estudiar la complejidad de otros problemas que surgen a raíz del Juego de la Propagación así como generalizaciones del problema visto en este escrito, como algunos de los mencionados a lo largo del Capítulo 3 que permitirán representar de manera más fidedigna el funcionamiento de las propagaciones. Previsiblemente, cualquier nueva versión de nuestro problema que incorporase nuevos factores sería una generalización de nuestro problema original, por lo que necesariamente tendríamos como mínimo su dureza en `PSPACE`. No obstante, esos nuevos factores podrían impedir la pertenencia a `PSPACE` y provocar una dureza superior (e.g. `EXPTIME`).

Chapter 8

Conclusion

In this study we have formally defined the Propagation Game, with special emphasis on finding a successful strategy for the appeaser, a figure in charge of slowing down its progress by saving a minimum number of nodes from its reach. We have proved that this problem is PSPACE-completo, demonstrating its intractability through a polynomial reduction from the TQBF problem. Its intractability has encouraged us to solve it by applying heuristic methods, in particular, the minimax algorithm.

We have developed practical experiments, using both artificial examples and a case study based on real data extracted from the COVID-19 crisis in Spain. These examples allowed us to study the performance of the minimax algorithm with alpha-beta pruning applied to our problem, reflecting its complexity and the rigor which its parameters must be chosen with. This is especially important for the depth explored by the minimax since it depends entirely on whether the algorithm can find the optimal solution, and why each sample must be carefully assessed.

Our case study exposed one of the main obstacles when using real data: the choice of (a, b) values to represent each of the nodes. When there is not a balanced choice between these values, the game will be completely biased towards one of the two players. This is one of the main limitations of our model, which is unable to represent the large number of variables that influence the spread of a propagation, ranging from biological to social ones. Despite this, we hope that our results will be valuable to other researchers and constitute a further step in the study of propagations and how to stop them.

Regarding possible future work, besides further improvements to our implementation, we will work on developing more sophisticated and efficient algorithms that would allow us make a better representation of reality. It is also necessary to study the complexity of other problems that appear from the Propagation Game as well as generalizations of the problem seen in this work, such as some of the problems mentioned in Chapter 3 that will allow a more reliable representation of the performance of the propagations. Foreseeably, any new version of our problem that incorporates new factors would be a generalization of our original problem, so we would necessarily have at least its hardness in PSPACE. However, those new factors could prevent belongingness to PSPACE and cause a higher hardness (i.e., EXPTIME).

Índice de figuras

Índice de cuadros

Bibliografía

[1] Óscar Toledano, B. Mula, S. N. Santalla, J. Rodríguez-Laguna, and Óscar Gálvez, "Effects of confinement and vaccination on an epidemic outburst: a statistical mechanics approach," 2021.

[2] V. T. PASCHOS, "An overview on polynomial approximation of NP-hard problems," *Yugoslav Journal of Operations Research*, vol. 19, no. 1, pp. 3–40, 2009.

[3] M. Sipser, *Introduction to the Theory of Computation*. Thomson Course Technology, 2006.

[4] J. E. Hopcroft, R. Motwani, and J. Ullman, "Introducción a la teoría de autómatas, lenguajes y computación," 2002.

[5] Baeldung on Computer Science., "P, NP, NP-complete and NP-hard problems in computer science – Baeldung on Computer Science." https://www.baeldung.com/cs/p-np-np-complete-np-hard, 2021.

[6] S. Arora and B. Barak, *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[7] Javier Galiana Ruiz de la Hermosa., "Cómo frenar una propagación: resolución práctica.." https://colab.research.google.com/drive/1PLmMxwmPPnRb6Nu6qfmPygV1sZnhGCuV?usp=sharing, 2021.

[8] Rashmi Jain., "Minimax algorithm with alpha-beta pruning." https://www.hackerearth.com/blog/developers/minimax-algorithm-alpha-beta-pruning/, 31 de marzo, 2017.

[9] M. L. Bynum, G. A. Hackebeil, W. E. Hart, C. D. Laird, B. L. Nicholson, J. D. Siirola, J.-P. Watson, and D. L. Woodruff, *Pyomo–optimization modeling in python*, vol. 67. Springer Science & Business Media, third ed., 2021.

[10] Wikipedia contributors, "Betweenness centrality — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Betweenness_centrality&oldid=1018484781, 2021.

[11] Corey Abshire., "An implementation of brandes algorithm." https://github.com/coreyabshire/tron/blob/master/brandes.py, Enero, 2019.

[12] University of Cambridge., "Brandes algorithm." https://www.cl.cam.ac.uk/teaching/1718/MLRD/handbook/brandes.html, Marzo, 2020.

[13] Expansión / Datosmacro.com., "Presupuestos de las comunidades autónomas: Sanidad." https://datosmacro.expansion.com/estado/presupuestos/espana-comunidades-autonomas?sc=PR-G-F-31, 2020.

[14] Instituto Nacional de Estadística., "Estudios de movilidad a partir de la telefonía móvil." https://www.ine.es/jaxiT3/Datos.htm?t=35167, Marzo, 2020.

Javier Galiana Ruiz de la Hermosa
Junio, 2021
Ult. actualización 15 de junio de 2021
LaTeX lic. LPPL & powered by TEFLON CC-ZERO