Análisis de complejidad y resolución práctica de un problema de expansión empresarial

Complexity analysis and practical resolution of a business expansion problem

Trabajo de Fin de Grado
Curso 2022–2023

Autor
David Pantoja Sánchez

Directores
Ismael Rodríguez Laguna
Fernando Rubio Diez

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Resumen

Este Trabajo de Fin de Grado es un estudio de las expansiones empresariales de forma principalmente teórica cuyo objetivo principal es exponer, practicar y ampliar algunos conocimientos aprendidos en el grado de Ingeniería Informática, e implementar una herramienta capaz de, pese a manejar situaciones simplificadas, ayudar a tomar decisiones de expansión empresarial.

Comenzaremos explicando conceptos de complejidad computacional del tiempo y del espacio que serán importantes en el estudio y el algoritmo minimax. Continuaremos estableciendo la definición del problema de expansión empresarial de forma teórica.

Una gran parte del trabajo se dedicará al análisis de la complejidad del problema. Primero demostraremos que pertenece a PSPACE usando un algoritmo minimax y después reduciremos polinómicamente el conocido problema PSPACE-completo TQBF a nuestro problema de expansión empresarial. Para ello, encontraremos una transformación de las posibles entradas de TQBF a entradas de nuestro problema de forma que sólo sea posible expandirse de una forma determinada, que garantizará que la solución del problema de expansión sea siempre la misma que la del problema TQBF original.

A continuación, explicaremos la herramienta que hemos desarrollado. Esta herramienta es capaz de, entre otras cosas, resolver problemas de expansión empresarial y TQBF y simular manualmente situaciones de expansión concretas. Estudiaremos el rendimiento de esta herramienta en función de varios parámetros. La herramienta utilizará un algoritmo minimax con poda alfa-beta. Habiendo demostrado la intratabilidad del problema, será necesario utilizar métodos heurísticos para resolverlo de forma subóptima.

Concluiremos el estudio usando la herramienta desarrollada para estudiar un caso de expansión empresarial.

Palabras clave

PSPACE-completitud, complejidad, algoritmo minimax, poda alfa-beta, gestión empresarial, múltiplo EBITDA.

Abstract

This Final Degree Project is a study of business expansions in a mainly theoretical way whose main objectives are to showcase, apply and expand some concepts learned in the Computer Science Engineering degree, and to implement a tool capable of, despite handling simplified situations, helping to make business expansion decisions.

We will begin explaining some computational complexity problems that will be important in the study and the minimax algorithm. Then, we will establish the theoretical definition of the problem of business expansions.

A large part of the project will be devoted to the analysis of the complexity of the problem. First, we will prove it belongs to PSPACE using a minimax algorithm and then we will polynomically reduce the well-known PSPACE problem TQBF to our business expansion problem. To accomplish this, we will find a transformation of the possible inputs of TQBF to inputs of our problem so that it is only possible to expand in a certain way, which will guarantee that the solution of TQBF and the expansion problem match for every possible pair of inputs.

Next, we will explain the tool we have developed. This tool is capable of, among other things, solve business expansion and TQBF problems and simulate concrete expansion situations manually. We will study the performance of this tool depending on various parameters. The tool will use a minimax algorithm with alpha-beta pruning. Having proven the intractability of the problem, it will be necessary to use heuristic methods to solve it suboptimally.

We will conclude the study using the tool we have developed to study a case of business expansion.

Keywords

PSPACE-completeness, complexity, minimax algorithm, alpha-beta pruning, business management, EBITDA multiple.

Índice

Índice de figuras

Capítulo 1

Introducción

En esta introducción del estudio vamos a explicar primero por qué hemos elegido esta temática. Continuaremos explicando la intratabilidad y el algoritmo minimax, dos conceptos que vamos a utilizar en las demostraciones teóricas y en la resolución práctica de nuestro estudio.

1.1. Motivación

Una de las principales motivaciones de este estudio de expansiones empresariales es el desarrollo académico del autor en el área de la complejidad computacional.

En el estudio, vamos a centrarnos en la aplicación del problema que definiremos más adelante en situaciones de expansión empresarial, pero está definido de forma muy genérica y podría aplicarse también a otras situaciones de expansión, como a juegos de simulación de imperios y a la publicidad, donde la expansión podría entenderse como la compra de espacios publicitarios en medios de comunicación o patrocinios. Por lo tanto, demostrar la intratabilidad de este problema y desarrollar una herramienta heurística que pueda ayudar en estas situaciones es otra razón por la que hemos elegido esta temática.

Finalmente, al igual que nosotros hemos consultado otros Trabajos de Fin de Grado [6, 7] para realizar nuestro estudio, esperamos que este trabajo pueda ser utilizado como referencia en futuros estudios sobre la complejidad computacional y, más concretamente, demostraciones de PSPACE-completitud.

1.2. Problemas intratables

En esta sección vamos a introducir conceptos importantes de la complejidad computacional del tiempo y del espacio como la NP-completitud y la PSPACE-completitud [8, 10, 14]. Estos conceptos serán de vital importancia en el resto del estudio.

1.2.1. Complejidad temporal

La complejidad temporal de un algoritmo que resuelve un problema es la medida del tiempo que tarda en resolver el problema respecto al tamaño de la entrada. Esta medida es utilizada para comparar la eficiencia de algoritmos que resuelven un mismo problema y para estudiar la viabilidad de un algoritmo para una situación determinada.

Un algoritmo de tiempo polinómico es uno cuya complejidad temporal está acotada superiormente en el caso peor por un polinomio del tamaño de la entrada.

Un problema de decisión es un tipo de problema que tiene como únicas salidas posibles "sí" y "no".

La clase P es el conjunto de problemas de decisión que pueden ser resueltos por un algoritmo determinista de tiempo polinómico.

Un algoritmo no determinista de tiempo polinómico es un algoritmo que puede ser ejecutado en tiempo polinómico por una máquina de Turing no determinista. NP es el conjunto de problemas de decisión que pueden ser resueltos por un algoritmo no determinista de tiempo polinómico, lo cual no quiere decir que para dichos problemas tengan que existir algoritmos deterministas de tiempo polinómico que los resuelvan (en adelante, nos referiremos a algoritmos deterministas cuando no especifiquemos su tipo). La cuestión $P \stackrel{?}{=} NP$ es uno de los problemas no resueltos más importantes de las ciencias de la informática teórica. Todos los problemas en P están en NP, pero no sabemos si todos los problemas en NP están en P. Para demostrar que $P \neq NP$ habría que encontrar un problema en NP no presente en P, y para demostrar que $P = NP$ habría que encontrar un algoritmo de tiempo polinómico que resuelva cada problema en NP.

Vamos a suponer que, dados dos problemas de decisión A y B, tenemos un algoritmo que resuelve B y queremos resolver A. Podremos resolver A usando el algoritmo de B si encontramos un algoritmo que, para toda entrada del problema A, construya una del problema B de forma que la solución de B con esa entrada ("sí" o "no") sea la misma que la esperada del problema A con la entrada original. Este algoritmo se denomina un algoritmo de transformación.

La reducibilidad polinómica es una relación entre dos problemas de decisión A y B que indica que existe un algoritmo de transformación de tiempo polinómico del problema A al B. $A \leq^p B$ denota que A es reducible polinómicamente a B. Si $B \in P$ y $A \leq^p B$ entonces $A \in P$.

Un problema B es NP-difícil si para cualquier problema A en NP se cumple $A \leq^p B$. No es necesario que B sea un problema de decisión y esté en NP para que sea un problema NP-difícil. En cambio, un problema es NP-completo si está en NP y es NP-difícil. No es necesario demostrar que todos los problemas en NP son reducibles a un problema C para demostrar que C es NP-completo; es suficiente con demostrar que está en NP y que existe un problema NP-difícil B ya conocido reducible polinómicamente a C. Sabemos que esto es cierto porque todos los problemas en NP son reducibles polinómicamente a B y la relación de reducibilidad es transitiva. El Problema del Viajante y el Problema del Cliqué son ejemplos de problemas NP-difíciles, y tienen problemas de decisión asociados que son NP-completos (por

ejemplo, averiguar si existe un cliqué de un cierto tamaño $k \in \mathbb{N}$ dado en un grafo).

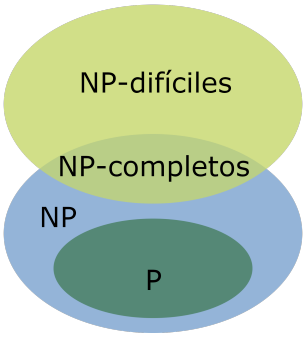


Figura 1.1: Representación de la relación entre clases de complejidad (P, NP, NP-completos, NP-difíciles)

Ahora que conocemos el conjunto de problemas NP-completos, podríamos probar que $P = NP$ encontrando un único problema NP-completo que pertenezca a P, es decir, encontrando un algoritmo de tiempo polinómico que resuelva un problema NP-completo. No se ha encontrado ninguno ni se ha demostrado que exista un problema NP no perteneciente a P, por lo que la cuestión $P \stackrel{?}{=} NP$ sigue abierta.

Para resolver problemas NP-difíciles, se pueden usar algoritmos exactos o heurísticos. Como no se ha encontrado ningún algoritmo de tiempo polinómico que resuelva un problema NP-difícil, los algoritmos exactos encuentran siempre la solución correcta pero en tiempo no polinómico respecto a la entrada, por lo que en muchas situaciones son inviables. Los algoritmos heurísticos encuentran en menos tiempo una solución subóptima.

1.2.2. Complejidad espacial

Vamos a trasladar los conceptos de la complejidad computacional del tiempo que hemos visto al espacio.

De forma similar a la complejidad temporal, la complejidad espacial de un algoritmo que resuelve un problema es la medida del espacio en memoria que requiere para resolver el problema respecto al tamaño de la entrada y un algoritmo de espacio polinómico es uno cuya complejidad espacial está acotada superiormente en el caso peor por un polinomio del tamaño de la entrada. Es decir, para una entrada de tamaño n, el algoritmo no requerirá en ningún momento una cantidad de espacio superior a un polinomio de n.

PSPACE es el conjunto de problemas de decisión que pueden ser resueltos por un algoritmo determinista de espacio polinómico.

Un problema B es PSPACE-difícil si para cualquier problema A en PSPACE se cumple $A \leq^p B$, es decir, si todos los problemas en PSPACE son reducibles en tiempo polinómico respecto al tamaño de la entrada a B. Finalmente, un problema es PSPACE-completo si es PSPACE-difícil y pertenece a PSPACE. Para demostrar

que un problema es PSPACE-completo será suficiente con demostrar que pertenece a PSPACE y que existe otro problema PSPACE-difícil reducible polinómicamente a él. El problema TQBF, que definiremos más adelante, es un ejemplo de problema PSPACE-completo, y lo utilizaremos para demostrar la PSPACE-dificultad del problema de expansión empresarial que vamos a estudiar. Se ha demostrado que $P \subseteq NP \subseteq PSPACE$, pero $P \stackrel{?}{=} PSPACE$ es otro de los problemas no resueltos más importantes de las ciencias de la informática teórica.

1.3. Algoritmo minimax

El algoritmo minimax es uno de los algoritmos más importantes de la teoría de juegos. Vamos a utilizarlo tanto de forma teórica para demostrar la pertenencia a PSPACE de nuestro problema como en la resolución práctica. Este algoritmo recursivo se utiliza en juegos con adversario por turnos y donde los jugadores tienen información perfecta, como el ajedrez, para decidir el movimiento que minimice la pérdida máxima esperada. El algoritmo minimax busca el mejor movimiento para un jugador suponiendo que en los siguientes turnos todos los jugadores jugarán óptimamente; el jugador original escogerá el mejor movimiento para él (el que minimice su pérdida) y los otros el movimiento que más le perjudique (el que maximice su pérdida). En la mayoría de juegos en los que se aplica este algoritmo hay dos jugadores, como en el nuestro, que definiremos más adelante. Por ello, vamos a estudiar el algoritmo minimax de dos jugadores.

El algoritmo minimax se suele representar con un árbol donde cada nodo representa una configuración del juego. Los nodos que representan configuraciones en las que es el turno del primer jugador se denominan nodos MAX y los que representan configuraciones en las que es el turno del segundo nodos MIN. Los nodos hijos de un nodo representarán las configuraciones a las que se puede llegar con cada acción del nodo padre. La paridad de la profundidad de un nodo indicará, por lo tanto, a qué jugador le tocará realizar una acción desde la configuración asociada a ese nodo.

Para decidir qué movimientos son más óptimos es necesario dar una puntuación a los nodos. Para ello usaremos una función heurística para valorar los nodos terminales, que representan las configuraciones finales del juego, y propagaremos las puntuaciones hacia arriba en el árbol. En muchos juegos, la heurística de los nodos terminales es trivial porque solo existe un dato, de naturaleza binaria, relevante; el primer jugador ha ganado y el segundo ha perdido o viceversa. En estos casos se suelen puntuar los nodos terminales con 1 o 0.

El algoritmo usará un recorrido de profundidad. Para propagar las puntuaciones hacia arriba, se asignará a los nodos MAX la puntuación máxima asignada a uno de sus nodos hijos, y a los MIN la puntuación mínima. Es decir, el primer jugador elegirá la acción que más le favorezca y el segundo la que menos favorezca al primer jugador.

La Figura 1.2 ilustra el funcionamiento del algoritmo. El primer nodo terminal al que llega el recorrido en profundidad es valorado por la heurística con una puntuación de 4. Este valor es propagado hacia arriba y como el primer nodo MIN no tenía una

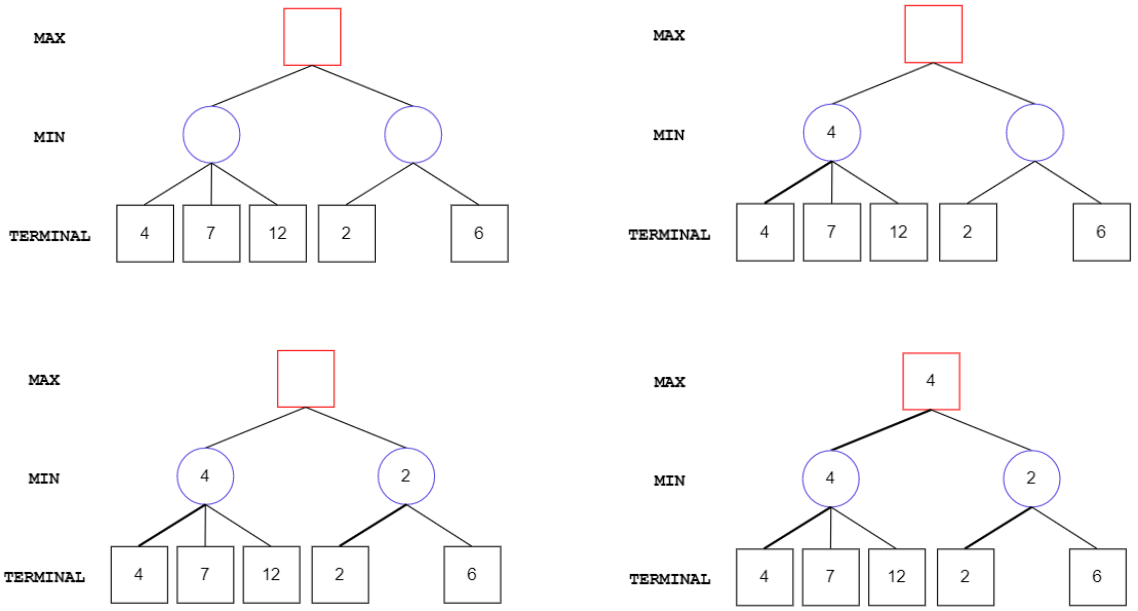


Figura 1.2: Ejemplo de aplicación del algoritmo minimax [7]

puntuación asignada, obtiene también la puntuación de 4 (es trivialmente mejor que la que tenía). El recorrido continúa con los otros dos hijos, dándoles las puntuaciones 7 y 12 calculadas por la heurística. Estas puntuaciones son propagadas hacia arriba, pero como es un nodo MIN no mejoran a 4. De forma similar, el otro nodo MIN obtiene la puntuación más baja de sus hijos y el nodo MAX la más alta. En este caso, el movimiento elegido por el algoritmo minimax es el que le lleva al primer nodo MIN porque le garantiza que, independientemente de cómo juegue el rival, podrá conseguir una puntuación de 4 y ningún otro movimiento le garantiza una puntuación mayor que 4.

Chapter 2

Introduction

In this introduction of the study we are going to start by explaining why we have chosen this topic. We will continue with an explanation of intractability and the minimax algorithm, two concepts that we are going to use in the theoretical demonstrations and in the practical resolution of our study.

2.1. Motivation

One of the main motivations for this study of business expansions is the academic development of the author in the area of computational complexity.

In this study, we will focus on the application of the problem that we will define later in business expansion situations, but it is defined very generically and could also be applied in other expansion situations, like in empire simulation games and in advertising, where the expansions could represent sponsorships or purchases of advertising space in media. Therefore, demonstrating the intractability of this problem and developing a heuristic tool capable of helping in these situations is another reason why we have chosen this topic.

Finally, similarly to how we have consulted other Final Degree Projects [6, 7] to aid us in our study, we hope that this work can be used as a reference in future studies about computational complexity and, specifically, demonstrations of PSPACE-completeness.

2.2. Intractable problems

In this section we will introduce some important concepts of computational complexity of time and space like NP-completeness and PSPACE-completeness [8, 10, 14]. These concepts will be of vital importance in the rest of our study.

2.2.1. Time complexity

The time complexity of an algorithm that solves a problem is the measure of the time it takes to solve that problem in relation to the size of the problem's input. This measure is used to compare the efficiency of algorithms that solve the same problem and to study the viability of an algorithm in a given situation.

A polynomial time algorithm is an algorithm whose time complexity has a polynomial of the size of the input as an upper bound.

A decision problem is a type of problem that has "yes" and "no" as its only possible outputs.

The class P is the set of decision problems that can be solved by a deterministic polynomial-time algorithm.

A non-deterministic polynomial-time algorithm is an algorithm that can be executed in polynomial time by a non-deterministic Turing machine. NP is the set of decision problems that can be solved by a non-deterministic polynomial time algorithm, which doesn't necessarily mean that those problems need to be solvable by deterministic polynomial-time algorithms (hereinafter, we will refer to deterministic algorithms when we do not specify their type).

$P \stackrel{?}{=} NP$ is one of the most important unsolved problems in theoretical computer science. All problems in P are in NP, but we do not know if all problems in NP are in P. To prove $P \neq NP$ we would have to find a problem in NP not present in P, and to prove $P = NP$ we would have to find for each problem in NP a polynomial-time algorithm that solves it.

We are going to assume that, given two decision problems A and B, we have an algorithm that solves B and we want to solve A. We could solve A using B's algorithm if we find an algorithm that, for every possible input of problem A, constructs one of problem B such that the output of B with that input ("yes" or "no") is the same as the expected output of A with the original input. This algorithm is called a transformation algorithm.

Polynomial reducibility is a relation between two decision problems A and B that indicates that there exists a polynomial-time transformation algorithm from the problem A to the problem B. $A \leq^p B$ denotes that A is polynomially reducible to B. If $B \in P$ y $A \leq^p B$ then $A \in P$.

A problem B is said to be NP-hard if for every problem A in NP $A \leq^p B$. It is not necessary for B to be a decision problem or a NP problem for it to be NP-hard. A problem is NP-complete if it is in NP and it is NP-hard. We do not need to prove that every NP problem is polynomially reducible to C to prove that C is NP-complete; it is enough to show that it is in NP and that there exists an already known NP-hard problem B polynomially reducible to C. We know that this is true because every NP problem is polynomially reducible to B and the reducibility relation is transitive. The Travelling Salesman Problem and the Clique Problem are well-known examples of NP-difficult problems, and they have NP-complete decision problems associated with them (for example, finding out whether there exists a clique of a given size $k \in \mathbb{N}$ in a graph).

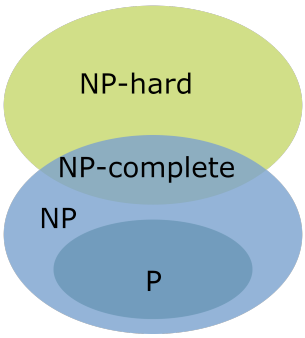


Figure 2.1: Representation of the relationship between complexity classes (P, NP, NP-complete, NP-hard)

Now that we know the NP-complete problem set, we can prove that $P = NP$ by finding a NP-complete problem that is in P, that is, by finding a polynomial-time algorithm that solves a NP-complete problem. So far, none has been found nor has it been proven that there exists a NP problem that is not a P problem, so the question $P \stackrel{?}{=} NP$ remains open.

To solve NP-hard problems, we can use exact or heuristic algorithms. As no polynomial-time algorithm that solves any NP-hard problem has been found, exact algorithms always find the correct solution but in non-polynomial with respect to the input, making them infeasible in many situations. Heuristic algorithms find in less time a suboptimal solution.

2.2.2. Spatial complexity

We are going to translate the concepts of computational time complexity that we have explained to space.

Similar to time complexity, the spatial complexity of an algorithm that solves a problem is the measure of the amount of space in memory it requires to solve the problem relative to the input size, and a polynomial space algorithm is one whose spatial complexity has, in the worst case, a polynomial of the size of the input as an upper bound. That is, for an input of size n, the algorithm will not require in any moment of the execution an amount of space greater than a polynomial of n.

PSPACE is the set of decision problems that can be solved by a deterministic polynomial-space algorithm.

A problem B is said to be PSPACE-hard if for every problem A in PSPACE $A \leq^p B$, that is, if all PSPACE problems are polynomial-time reducible with respect to the size of the input to B. Finally, a problem is PSPACE-complete if it is PSPACE-hard and belongs to PSPACE. To prove that a problem is PSPACE-complete it will be sufficient to prove that it belongs to PSPACE and that there exists another PSPACE-hard problem polynomially reducible to it. The problem TQBF, that we will explain later, is a well-known PSPACE-complete problem, and we will use it to demonstrate the PSPACE-hardness of the business expansion problem that we are

going to study. It has been proven that $P \subseteq NP \subseteq PSPACE$, but $P \stackrel{?}{=} PSPACE$ is another of the most important unsolved problems in computer science.

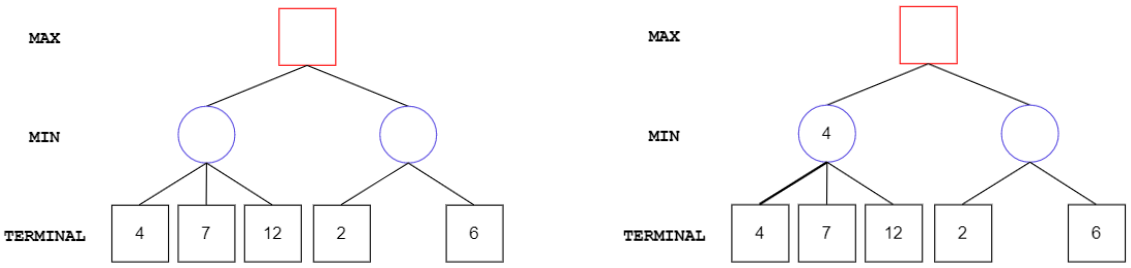
2.3. Minimax algorithm

The minimax algorithm is one of the most important algorithms in game theory. We are going to use it both theoretically to prove that our problem belongs in PSPACE and in the practical resolution. This recursive algorithm is used in turn-based games with an opponent where the players have perfect information, such as chess, to decide the move that minimizes the maximum expected loss. The minimax algorithm searches for the best move for a player assuming that in the following turns all players will play optimally; the original player will choose his best possible move (the one that minimizes his loss) and the others the move that hurts him the most (the one that maximizes his loss). In most games in which this algorithm is applied there are two players, like in ours, that we will define later. Therefore, we are going to study the minimax algorithm for two players.

The minimax algorithm is commonly represented with a tree where every node represents a game configuration. The nodes that represent configurations where it is the turn of the first player are called MAX nodes and those representing configurations where it is the second player's turn are called MIN nodes. The child nodes of a node will represent the configurations that are reachable with each possible action of the parent node. The parity of the depth of a node will, therefore, indicate which player has to perform an action from the configuration associated with that node.

To decide which moves are the most optimal it is necessary to score the nodes. To do this we will use a heuristic function to score the terminal nodes, which represent the final configurations of the game, and we will propagate the scores up the tree. In many games, the heuristic of the terminal nodes is trivial because there is only one binary piece of information relevant; the first player has won and the second has lost or vice versa. In these cases it is common to score terminal nodes with 1 or 0.

The algorithm will follow a depth-first path. To propagate the scores upwards, MAX nodes will be assigned the maximum score assigned to one of their child nodes, and MIN nodes will be assigned the minimum score. That is, the first player will choose the action that benefits him the most and the second the one that benefits the first player the least.



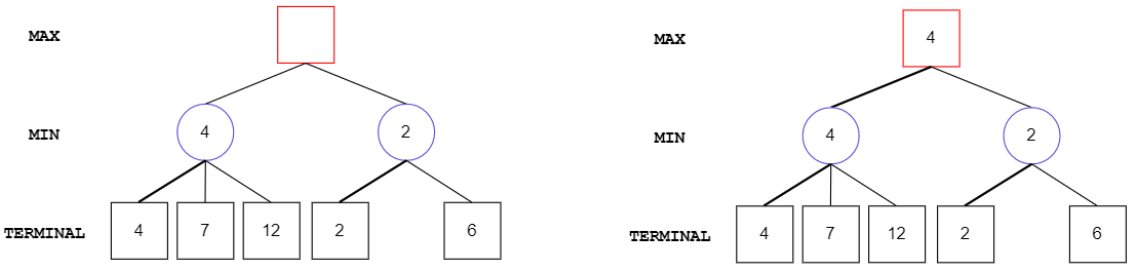


Figure 2.2: Example of the application of the minimax algorithm [7]

The Figure 2.2 illustrates the behavior of the algorithm. The first terminal node reached by the depth-first traversal is rated by the heuristic with a score of 4. This value is propagated upwards and, as the first MIN node did not have a score assigned to it, it also gets the score of 4 (it is trivially better than his previous one). The depth-first traversal continues with the other two child nodes, giving them the scores 7 and 12 calculated by the heuristic. These scores are propagated upwards, but their parent is a MIN node so they do not improve 4. Similarly, the other MIN node gets the lowest score of its children and the MAX node the highest. In this case, the movement chosen by the minimax algorithm is the one that leads to the first MIN node because it guarantees that, regardless of how the opponent plays, it will be able to get at least a score of 4, and no other move guarantees a score higher than that.

Capítulo 3

Definición del problema

Vamos a estudiar una competición entre dos empresas o jugadores (J_1 y J_2) que se expanden por un territorio efectuando edificaciones comerciales en distintos terrenos disponibles. Denominaremos a esta competición como el *juego de expansión* (JE). Cada edificación tiene un coste y cada propiedad comercial genera una cantidad de ganancias fija por unidad de tiempo. Además, los terrenos pueden estar relacionados: poseer ciertos terrenos en conjunto generará una cantidad de ganancias adicional. Por ejemplo, disponer de todos los terrenos disponibles en una zona puede dar lugar a mayores beneficios por estar en una situación de monopolio.

Los jugadores realizarán compras de terrenos y edificaciones en los mismos por turnos. A partir de ahora diremos que un jugador ha comprado un terreno cuando éste haya realizado tanto la compra como la edificación posterior.

Los *turnos* son segmentos de tiempo consecutivos $(t_1, t_2, ..., t_n)$, cada uno de duración igual a la unidad de tiempo referencia, es decir, la cantidad de tiempo que tiene que pasar para que se obtenga una cantidad de dinero igual al valor de ganancia actual. En los turnos impares, J_1 obtendrá una cantidad de dinero igual a su ganancia total y podrá edificar tantas veces como quiera con el dinero que tenga disponible o decidir no hacerlo. La mecánica será idéntica para J_2 en los turnos pares.

Llamaremos T, E_1 y E_2 a los conjuntos de terrenos no edificados, edificados por J_1 y por J_2 respectivamente, donde un terreno $n \in T \cup E_1 \cup E_2$ es una terna $n = (id, c, g)$, id es su identificador único y $c, g \in \mathbb{Q}^+ \cup \{0\}$ son respectivamente su coste y su ganancia. Siempre se cumplirá que $(E_1 \cup E_2) \cap T = \emptyset$ y que $E_1 \cap E_2 = \emptyset$.

La información sobre las ganancias adicionales por tener varios terrenos relacionados simultáneamente denota el conjunto M. $\forall R \in M, R = (g, ID)$, donde R es una *relación*, $g \in \mathbb{Q}^+ \cup \{0\}$ es su ganancia adicional correspondiente y donde ID es el conjunto de los identificadores de los terrenos relacionados que deben poseerse simultáneamente para recibir la ganancia adicional. Diremos que un jugador ha completado una relación R cuando éste haya acabado de comprar todos los terrenos cuyos índices están en $R.ID$ (para toda tupla $t = (c_1, ..., c_i, ..., c_n)$, asumiremos que $t.c_i = c_i$).

El parámetro de entrada de JE es la tupla $(T_{ini}, M, d_{ini,1}, d_{ini,2})$, siendo $d_{ini,1}$,

$d_{ini,2} \in \mathbb{Q}^+ \cup \{0\}$ la cantidad de dinero inicial con la que comienzan J_1 y J_2, respectivamente. El conjunto T_{ini} tiene que contener al menos un terreno.

La *configuración* de un juego es la descripción completa del estado del juego en un momento dado (en nuestro caso, un turno). Definiremos una configuración de JE como $(T, E_1, E_2, M, d_1, d_2, g_1, g_2, t)$, donde t es el turno actual, $d_1, d_2 \in \mathbb{Q}$ denotan la cantidad de dinero de J_1 y J_2, y $g_1, g_2 \in \mathbb{Q}$ sus ganancias totales. Al inicio del juego, se genera la configuración inicial de JE a partir de la entrada de la siguiente forma: $T = T_{ini}$, $E_1 = E_2 = \emptyset$, $d_1 = d_{ini,1}$, $d_2 = d_{ini,2}$, $g_1 = g_2 = 0$ y $t = 0$.

J_1 comienza su turno sumando a su dinero (d_1) su valor de ganancia (g_1). Prosigue comprando tantos terrenos como quiera con el dinero que dispone. Mientras exista un terreno $n \in T$ con $n.c \leq d_1$ que J_1 quiera comprar, el proceso de compra es el siguiente:

- $n.c$ es restado a d_1,
- $n.g$ es sumado a g_1,
- n es eliminado de T,
- n es añadido a E_1, y
- Si $\exists R = (g, ID) \in M : ((\forall id \in ID, id \in E_1) \wedge (\exists id \in ID : id = n.id))$ entonces hacemos $g_1 \leftarrow g_1 + g$. Es decir, se añade la ganancia adicional correspondiente a una relación a la ganancia (g_1) del primer jugador (J_1) cuando J_1 compra un terreno contenido en la relación y ya ha comprado el resto de terrenos contenidos en la relación (E_1 contiene todos los terrenos en la relación).

Cuando J_1 no tiene más terrenos que quiera comprar, comienza el turno de J_2, que sigue la misma mecánica.

Denominaremos a estas secuencias de compras de terrenos que tienen lugar en un turno determinado como *acciones*. Una *estrategia* es una manera de jugar que nos indica qué se debe hacer en cada situación posible del juego (en cada configuración) a la que nos hayan conducido nuestras acciones y las de nuestro rival.

El problema de *optimización del juego de expansión* (OJE) es el siguiente: dada una entrada de JE, ¿cuál es el valor máximo de g_1 que se puede obtener en el caso peor, es decir, al que se puede llegar con una estrategia de J_1 y que sea alcanzable contra todas las posibles estrategias de J_2?

Una *instancia* de un problema es la descripción completa de un caso del problema que se pretende resolver. Si disponemos de un programa capaz de resolver ese problema, la instancia del problema que queremos resolver será equivalente a una entrada del programa. El *éxito del juego de expansión* (EJE) es el problema definido a continuación: dada una entrada de JE y un número $o \in \mathbb{Q}$ (por lo tanto, una instancia de EJE será la tupla $(T_{ini}, M, d_{ini,1}, d_{ini,2}, o)$), ¿existe una estrategia para J_1 desde la configuración inicial del juego generada a partir de entrada que garantice llegar a un valor de g_1 mayor o igual que o antes de que lo haga g_2 para cualquier estrategia de J_2? Es decir, EJE finalizará con la victoria de la primera empresa que consiga llegar a un valor objetivo de ganancia (por unidad de tiempo). Si no quedan

terrenos por comprar y ningún jugador ha ganado, se le dará la victoria al jugador que ha empezado en segundo lugar (J_2). Si gana J_1, el resultado de la instancia de EJE que se está estudiando será $\top$ y si gana J_2 será $\bot$.

Capítulo 4

Análisis de complejidad

En este capítulo demostraremos que el problema EJE es PSPACE-completo. Para ello, primero probaremos que pertenece a PSPACE y completaremos la demostración reduciendo polinómicamente el conocido problema PSPACE-completo TQBF a EJE.

4.1. Pertenencia a PSPACE

Comencemos demostrando la pertenencia de EJE a PSPACE usando un algoritmo minimax con profundidad igual al número máximo de turnos que lo resuelva en espacio polinómico respecto al tamaño de la entrada, que explicaremos en profundidad en el capítulo *Resolución práctica*. Este algoritmo comprobará todas las secuencias de acciones posibles, empezando en la raíz del árbol minimax con la configuración de juego inicial definida a partir de la entrada del problema EJE a resolver. Los nodos hijo de un nodo padre del árbol son las configuraciones resultantes de todas las posibles acciones desde la configuración del nodo padre. El número máximo de hijos de cada nodo no es relevante respecto al espacio que se necesita para resolver EJE. No es necesario guardar en memoria simultáneamente todos los nodos (configuraciones de la instancia del problema) procesados del árbol; en cualquier momento del procesamiento del árbol es información suficiente para construir el resto de nodos el nodo que está siendo procesado y todos sus nodos ascendientes. Los nodos guardarán como información adicional un número $a \in \mathbb{N}$ menor que el número de acciones máximas desde cualquier configuración que representa la acción que tendría que hacer desde su configuración para llegar a la de su último nodo hijo que ha sido procesado (las acciones posibles están ordenadas). Mediante un recorrido en profundidad, desde cualquier nodo se puede llegar al siguiente a partir de la acción siguiente a la que indica a si existe o, si no quedan más hijos por recorrer, volviendo al padre y repitiendo el proceso.

Como no puede haber más de un nodo por cada nivel del árbol, el número máximo de nodos que tienen que ser guardados en memoria simultáneamente es igual a la profundidad máxima del árbol (el número de turnos máximo). El problema EJE sin ninguna restricción no tiene ningún límite al número máximo de turnos. Ambos jugadores pueden alternarse pasando en todos los turnos sin comprar ningún terreno

de forma indefinida. Para que EJE pertenezca a PSPACE, vamos a tener que añadir como restricción un número límite $l \in \mathbb{N}$ de turnos consecutivos que puede pasar el turno un mismo jugador sin perder la partida automáticamente como constante conocida del problema, es decir, sin ser parte de las instancias del problema. l no puede ser parte de la entrada porque entonces el valor numérico de l escalaría de forma exponencial respecto al espacio en memoria que ocupa.

Habiendo establecido este límite, el número de turnos será menor o igual a $(l + 1)(|T|+1)$. Es decir, los jugadores podrán maximizar el número de turnos empleados en la compra de cada terreno pasando el máximo número de turnos consecutivos sin comprar terrenos ($l \cdot |T|$) antes de la compra de cada terreno ($|T|$). En el caso de que el número de terrenos disponibles sea impar, J_2 tendrá que esperar n turnos adicionales sin comprar después de su última compra hasta que J_1 acabe. Se puede comprobar que esta estrategia maximiza el número de turnos viendo que no hay opción de añadir una acción de pasar el turno entre dos acciones de las estrategias de ambos jugadores; no se puede añadir después de la última compra (porque por definición el juego ya habría acabado al no quedar ningún terreno en T) y no se puede añadir antes de ninguna compra porque, como ya hay l pases de turno de cada jugador antes de todas sus compras de terreno, el juego acabaría prematuramente (o al mismo tiempo si justo antes de la compra final se pasa el turno otra vez).

Recordemos que habíamos definido las configuraciones de JE como tuplas $(T, E_1, E_2, M, d_1, d_2, g_1, g_2, t)$ que describen de forma total el estado de JE tras una secuencia de acciones de J_1 y J_2, pero como hemos tenido que añadir la limitación del número de pases consecutivos que puede hacer un jugador al problema EJE tenemos que añadir a las configuraciones los números enteros p_1 y p_2. Serán el número de turnos consecutivos que han pasado J_1 y J_2, respectivamente. Por lo tanto, usaremos como configuraciones del problema JE las tuplas $(T, E_1, E_2, M, d_1, d_2, g_1, g_2, t, p_1, p_2)$ y en la configuración inicial $p_1 = p_2 = 0$.

El espacio ocupado por T, E_1 y E_2 en cualquier configuración de EJE es polinómico con el espacio ocupado por T_{ini} en la entrada. Como $T \cup E_1 \cup E_2 = T_{ini}$, $(E_1 \cup E_2) \cap T = \emptyset$ y $E_1 \cap E_2 = \emptyset$ (siempre que se elimina un terreno de T se añade a E_1 o E_2, no se pueden añadir terrenos a E_1 o E_2 sin haber sido eliminados previamente de T, no se pueden eliminar terrenos de E_1 y E_2 y no se pueden añadir terrenos a T), la complejidad en espacio de T, E_1 y E_2 respecto a T_{ini} es $O(n)$.

El espacio ocupado por d_1, d_2, g_1, g_2, t, p_1 y p_2 es polinómico con la entrada. El valor de t $((l + 1)(|T| + 1))$ cumple esta propiedad porque l ocupa en memoria una cantidad constante de bits independientemente de la entrada y $|T|$ ocupa trivialmente igual o menos que T_{ini}. Los valores de p_1 y p_2 no pueden ser mayores que t porque están inicializados a 0 y pueden aumentar como máximo en incrementos de 1 cada turno. El valor de g_1 y g_2 máximo es igual a la suma de la ganancia de todos los terrenos en T más la ganancia de todas las relaciones en M. Cada una de esas ganancias están definidas explícitamente en la entrada (en T_{ini} y M). Como el número de bits que ocupa en memoria la suma de dos números es como mucho uno más que el número de bits del mayor de los dos números, el espacio ocupado por g_1 y g_2 es menor o igual que el ocupado por todas las ganancias en la entrada. Los valores de d_1 y d_2 máximos serán menores o iguales que la ganancia máxima por el

número de turnos máximo ($(l+1)(|T|+1)$). Como los valores máximos tanto de la ganancia como del número de turnos son polinómicos con la entrada y como el producto de dos polinomios es un polinomio de grado igual a la suma de los grados de los factores, su producto también lo será.

Finalmente, como el espacio en memoria máximo que ocupa el problema EJE al ser resuelto por un algoritmo minimax es igual al producto del número de configuraciones que tienen que poder ser guardadas simultáneamente (que es igual al número máximo de turnos del juego) y el espacio máximo ocupado por cada configuración, y ambos son polinómicos con la entrada, EJE pertenece a PSPACE.

4.2. Reducción de TQBF

El *Boolean satisfiability problem* (SAT) es el problema de encontrar una valoración de las variables de una fórmula lógica de primer orden que evalúe a $\top$. Por ejemplo, la expresión $(x_1 \wedge x_2) \wedge (\neg x_1 \vee x_2)$ es verdadera con los valores $x_1 = \top, x_2 = \top$, mientras que la expresión $(x_1 \wedge \neg x_2) \wedge (\neg x_1 \vee x_2)$ no es verdadera $\forall x_1, x_2 \in \{\top, \bot\}$.

El problema *true quantified Boolean formula* (TQBF) es una generalización de SAT en el que las variables están cuantificadas con cuantificadores existenciales ($\exists$) o universales ($\forall$). Las instancias de SAT son equivalentes a instancias de TQBF (llamadas QBF) en las que todas las variables están cuantificadas existencialmente. TQBF es el problema canónico PSPACE-completo de la teoría de la complejidad computacional y es frecuentemente usado en demostraciones de PSPACE-completitud. En las demostraciones, es habitual asumir que TQBF usa fórmulas en forma normal prenexa sin que deje de ser PSPACE-completo.

La *forma normal prenexa* (PNF) es una manera de expresar fórmulas lógicas de primer orden como un *prefijo* formado por los cuantificadores existenciales o universales y una *matriz* sin cuantificadores. Las instancias de esta versión de TQBF con esta condición se pueden escribir de la siguiente forma: $Q_1 x_1, ..., Q_n x_n \phi$, donde $\forall i \in [1, n]$, $Q_i \in \{\exists, \forall\}$ y x_i son variables proposicionales, y ϕ es una expresión booleana que no tiene ninguna variable no cuantificada. Un ejemplo de una fórmula en PNF (y de una instancia de TQBF) es $\forall x_1 \exists x_2 \exists x_3 : (x_1 \vee x_2) \vee (\neg x_1 \wedge \neg x_2 \wedge \neg x_3)$.

Cualquier instancia $Q_1 x_1, ..., Q_n x_n \phi$ de TQBF puede ser reescrita en la forma $Q_1 x_1, ..., Q_m x_m \phi$, donde $m \geq n$ y los cuantificadores $\exists$ y $\forall$ se alternan, es decir, $\forall i \in [1, m] : ((i \ mod \ 2 = 1) \Rightarrow (Q_i = \exists) \wedge (i \ mod \ 2 = 0) \Rightarrow (Q_i = \forall))$. Esto se puede conseguir en tiempo lineal respecto a la entrada añadiendo los cuantificadores correctos que faltan entre cada par de cuantificadores iguales consecutivos en el prefijo, cuantificando variables auxiliares no presentes en la matriz.

Además, la matriz es una fórmula lógica y puede ser convertida en forma normal disyuntiva (FND), es decir, disyunciones de cláusulas conjuntivas utilizando únicamente los operadores $\wedge$, $\vee$ y $\neg$. Esta versión de TQBF con las fórmulas en forma normal disyuntiva sigue siendo PSPACE-completa [3]. Utilizaremos en la reducción, sin pérdida de generalidad, instancias de TQBF con cuantificadores ordenados empezando por $\exists$ y con la matriz en FND.

4.2.1. Introducción informal a la transformación

Vamos a ilustrar esta introducción a la transformación de una instancia de TQBF a una de EJE usando como ejemplo la instancia $\exists x_1 \forall x_2 : (x_1 \wedge x_2) \vee (\neg x_1)$. Usaremos rectángulos para representar los terrenos y representaremos las relaciones uniendo mediante líneas los terrenos que forman parte de una misma relación.

Nuestro objetivo es encontrar una manera de crear una instancia de EJE a partir de una de TQBF cuyo resultado sea el mismo. Vamos a utilizar terrenos que representen las decisiones de valoración de las variables. Por cada variable en el prefijo, tendremos dos terrenos. Los llamaremos *terrenos normales*. El primero representa la valoración de la variable correspondiente como $\top$ y el segundo como $\bot$. Es decir, si el primer jugador compra el primer terreno del par de terrenos asociados a una variable, decidirá su valor como $\top$, y si compra el segundo lo decidirá como $\bot$.

Figura 4.1: Terrenos normales

Tenemos que asegurarnos de que al final de la partida el primer jugador ha comprado exactamente un terreno de cada par (no puede no haber decidido o haber decidido dos veces el valor de una variable). Además, como las variables en posiciones impares del predicado están cuantificadas existencialmente y las que están en posiciones pares universalmente, tenemos que ingeniar una forma de hacer que el primer jugador pueda elegir libremente qué terreno comprar de los pares asociados a variables en posiciones impares, pero que en los pares asociados a variables en posiciones pares el segundo jugador (cuya intención es hacer que el primero pierda) le obligue a comprar un terreno determinado.

Para poder satisfacer estas dos condiciones, vamos a hacer que los terrenos se compren de forma ordenada. El primer jugador empezará comprando el terreno asociado a la primera variable del predicado que quiera y el segundo continuará comprando el terreno restante asociado a esa variable. Después, le tocará al segundo jugador elegir una variable asociada a la segunda variable del predicado primero, y el primer jugador comprará el terreno restante. Continuarán alternándose de esta forma en la elección de terrenos de cada variable, con el primer jugador eligiendo primero los terrenos asociados a variables en posiciones impares del prefijo y el segundo eligiendo primero los asociados a variables en posiciones pares y sin que ningún jugador compre terrenos asociados a variables situadas más a la derecha en el prefijo que otras variables cuyos terrenos asociados todavía no hayan sido comprados. Esta mecánica simulará la alternación de cuantificadores en el prefijo.

Tenemos que obligar a los jugadores a seguir este orden de compra haciendo que, en cada momento en el juego, el resto de acciones que tengan disponibles que se salgan del orden establecido hagan que les sea imposible ganar independientemente de lo que hagan a partir de ese punto si el otro jugador juega correctamente. Para ello, vamos a hacer que los dos terrenos asociados a una variable tengan el mismo

coste que la posición de la variable en el prefijo, empezando por uno, y su ganancia será 1. De esta forma, los jugadores no podrán ahorrar dinero entre turnos ya que tendrán que gastar en cada turno todo el dinero que disponen para comprar el siguiente terreno.

Figura 4.2: Costes y ganancias de los terrenos normales

Además, vamos a añadir otros terrenos, que llamaremos *terrenos auxiliares*. Cada variable en el prefijo tendrá asociado un terreno auxiliar. Estos terrenos tendrán un coste igual a la posición en el prefijo de la variable asociada, como los terrenos normales, y su ganancia será 0. Servirán para lograr la alternación de elección entre los jugadores. Después de que los dos terrenos normales asociados a una variable hayan sido comprados, el jugador del que sea el turno (que es el jugador que tuvo la última elección de un terreno normal y que no tuvo que conformarse con el restante) tendrá que comprar el terreno auxiliar con un coste igual a la posición de la siguiente variable, dejando al otro jugador la elección en el siguiente par de terrenos normales.

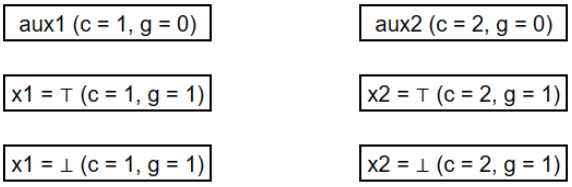


Figura 4.3: Terrenos auxiliares

Para obligar a los jugadores a comprar un terreno auxiliar cuando les toque, vamos a añadir unas *relaciones auxiliares* con una ganancia lo suficientemente alta como para garantizar a quien las complete la victoria inmediata que darán la victoria a un jugador que compre un terreno auxiliar que debería haber comprado el otro jugador. Para distinguir entre los jugadores, vamos a añadir tres terrenos iniciales que deberán ser comprados en los tres primeros turnos. Además de identificar a los jugadores, la compra de estos terrenos hará que ambos jugadores tengan 1 de ganancia y que el primer jugador tenga la elección del primer terreno normal.

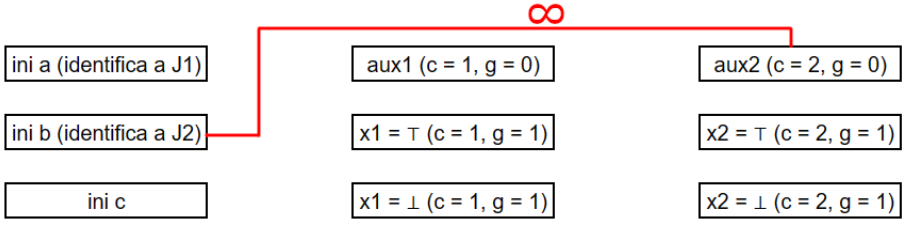


Figura 4.4: Una relación auxiliar (incompleta)

De esta forma, los jugadores no podrán saltarse un terreno auxiliar si quieren ganar. Pero, ¿qué pasaría si un jugador decide no comprar un terreno normal y decide ahorrar para comprar el siguiente turno, por ejemplo, un terreno auxiliar que no deba? Para hacer que un movimiento así dé lugar a una derrota inmediata, vamos a hacer que haya *relaciones intermedias* que den la victoria a quien consiga hacerse con los dos terrenos normales asociados a una variable. Tanto estas relaciones como las que dan la victoria a quien compre un terreno auxiliar que debe comprar el otro jugador incluyen también otros terrenos para asegurarnos de que únicamente podrán ser completadas por jugadores que han seguido la pauta establecida.

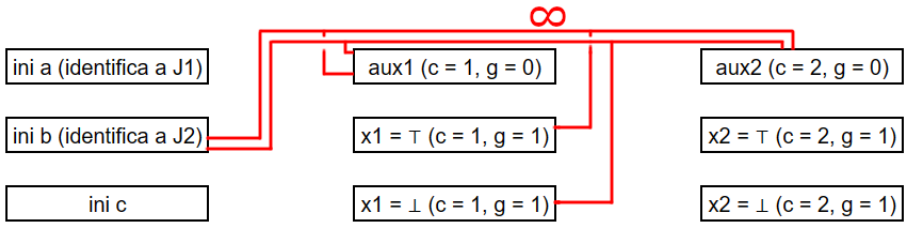


Figura 4.5: Dos relaciones auxiliares

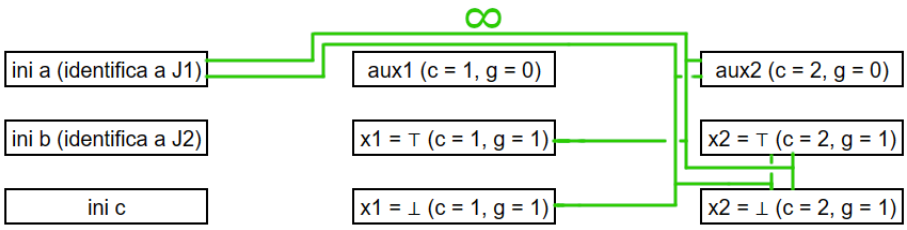


Figura 4.6: Dos relaciones intermedias

Finalmente, tenemos que encontrar una forma de, al final de la partida, comprobar si el primer jugador ha comprado unos terrenos normales cuyas decisiones asociadas satisfacen la QBF original. Para ello, como la matriz de QBF es una disyunción de conjunciones, vamos a añadir dos *terrenos finales*, otro terreno auxiliar y una *relación final* por cada conjunción en la matriz que incluya todos los terrenos que representen las decisiones que hacen que la conjunción sea cierta, un terreno final para asegurarnos de que se han tomado todas las decisiones y otros terrenos para asegurarnos de que el primer jugador ha seguido la pauta establecida. Los terrenos finales tendrán los valores de coste y ganancia que, si hubiese una variable más en el prefijo, tendrían los terrenos asociados a esa variable. Completar una de estas relaciones será suficiente para que el primer jugador gane la partida.

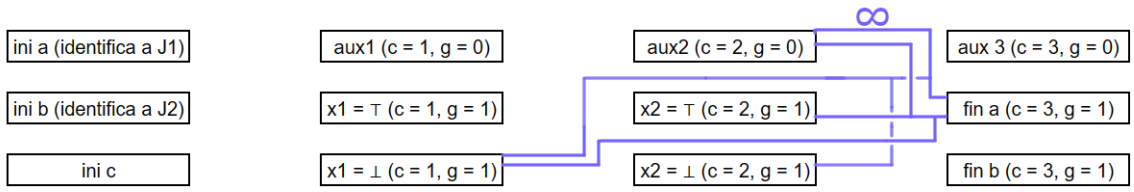


Figura 4.7: Dos relaciones finales

En la siguiente sección vamos a explicar en detalle y de manera formal la transformación de la entrada de TQBF que hemos introducido informalmente.

4.2.2. Transformación de la entrada de TQBF a una de EJE

Para reducir polinómicamente TQBF a EJE vamos a transformar la entrada de TQBF a una de EJE de forma que la respuesta para EJE sea la misma que la que se daría con la entrada original TQBF. Recordemos que una entrada o instancia de EJE es la tupla $(T_{ini}, M, d_{ini,1}, d_{ini,2}, o)$.

Para construir el conjunto T_{ini} a partir de la entrada de TQBF, es decir, la fórmula lógica en la forma especificada previamente, se crea un conjunto vacío y se le añaden:

- Por cada variable x_i en el prefijo (siendo i la posición en la que es cuantificada, empezando por 1), los *terrenos normales* $n_i = (idn_i, c_i, g)$ y $m_i = (idm_i, c_i, g)$, donde $c_i = i$ y $g = 1$.
- De nuevo, por cada variable x_i, el *terreno auxiliar* $aux_i = (idaux_i, c_i, g)$, donde $c_i = i$ y $g = 0$.
- El *terreno auxiliar* $aux_f = (idaux_f, c_f, g)$, donde $c_f = f$ y $g = 0$.
- Los *terrenos iniciales* $ini_a = (idini_a, c_a, g_a)$, $ini_b = (idini_b, c_b, g_b)$ y $ini_c = idini_c, c_c, g_c)$, donde $c_a = 1/100, g_a = 3/100$, $c_b = 2/100, g_b = 1$ y $c_c = 3/100, g_c = 0$. ini_a e ini_b son los terrenos iniciales *identificativos*.
- Los *terrenos finales* $fin_a = (idfin_a, c_f, g)$ y $fin_b = (idfin_b, c_f, g)$, donde $c_f = f$ y $g = 1$.

Denotaremos como *terrenos auxiliares correspondientes* a un jugador al conjunto de terrenos auxiliares que éste tiene que comprar si sigue la estrategia que describiremos en *Demostración de la validez de la entrada transformada*, que serán los terrenos auxiliares con un valor de coste par para J_1 e impar para J_2.

Cada terreno en T_{ini} tiene un *id* asociado y no puede haber dos terrenos con el mismo identificador. f es igual al número de variables en el prefijo más 1 e $INF = 3\sum_{i=1}^{f} i + 1$. Si un jugador obtiene INF dinero, puede comprar todos los terrenos en T_{ini}. Llamaremos nivel i al conjunto de terrenos (en la mayoría de casos, n_i, m_i y aux_i) de coste i en T_{ini}. Los terrenos iniciales identificativos (ini_a e ini_b) se utilizan

para distinguir a J_1 y J_2, los finales fin para verificar si el problema TQBF daría éxito y los terrenos auxiliares aux para alternar la elección de una variable o su negación (x o $\neg x$) entre J_1 y J_2. ini_c es usado para hacer que J_1 tenga la primera elección. Podría omitirse quitando aux_1 y cambiando la ganancia de ini_a a 1, pero hemos elegido incluirlo para no tener que añadir un caso especial en el nivel 1.

Para construir M a partir de la entrada de TQBF, se crea un conjunto vacío y se le añaden las siguientes relaciones $R = (g, ID)$:

- La *relación inicial* R_{ini} donde $R_{ini}.g = 97/100$ y su ID tiene los id de ini_a e ini_c. En total, J_1 obtiene una ganancia de 1 con la compra de ini_a e ini_c.

- Por cada variable en posición i en el prefijo, las cuatro *relaciones intermedias* del nivel i $\forall y \in \mathbb{N}, 1 \leq y \leq 4 : R_{inter,y,i}$, donde sus ID son un conjunto de identificadores que contiene los identificadores id de los terrenos n_i y m_i asociados a la variable y $g = INF$. Además:

  - $R_{inter,1,i}.ID$ contiene los id de n_{i-1} (si $i \neq 1$), ini_a y aux_i (si $i \ mod \ 2 = 0$).
  - $R_{inter,2,i}.ID$ contiene los id de m_{i-1} (si $i \neq 1$), ini_a y aux_i (si $i \ mod \ 2 = 0$).
  - $R_{inter,3,i}.ID$ contiene los id de n_{i-1} (si $i \neq 1$), ini_b y aux_i (si $i \ mod \ 2 = 1$).
  - $R_{inter,4,i}.ID$ contiene los id de m_{i-1} (si $i \neq 1$), ini_b y aux_i (si $i \ mod \ 2 = 1$).

  Es decir, para que una de estas relaciones sume su ganancia adicional a los jugadores, éstos necesitarán haber comprado los dos terrenos normales de un nivel i y uno del nivel anterior (si existe), uno de los terrenos iniciales identificativos y el terreno auxiliar del nivel i si les corresponde.

- $\forall i \in \mathbb{N}, 1 \leq i \leq f$, las *relaciones auxiliares* del nivel i $R_{aux,1,i}$ y $R_{aux,2,i}$ donde $g = INF$ y $\forall x \in \mathbb{N}^+ : 1 \leq x \leq i-1$ tal que $x \ mod \ 2 \neq i \ mod \ 2$, su ID contiene los id de los terrenos auxiliares aux_x, el de aux_i y el de ini_a si $i \ mod \ 2 = 1$ o ini_b si $i \ mod \ 2 = 0$. Además, $R_{aux,1,i}.ID$ contiene el id de n_{i-1} y $R_{aux,2,i}.ID$ el de m_{i-1} si $i > 1$ (de no ser así, n_{i-1} y m_{i-1} no existirían).

  Es decir, para que una de estas relaciones sume su ganancia adicional a los jugadores, éstos necesitarán haber comprado un terreno auxiliar aux_i que no les corresponda (uno que no podrían adquirir si el rival jugase de la manera estándar planteada por la construcción), todos los terrenos auxiliares de niveles inferiores que les corresponden, uno de los terrenos iniciales identificativos y un terreno normal del nivel anterior. Nótese que $R_{aux,1,1}$ y $R_{aux,2,1}$ son equivalentes, pero vamos a mantener ambas relaciones en M por simplicidad.

- Por cada conjunción en la matriz (recordemos que la matriz es una disyunción de conjunciones), incluso si es de una sola variable, las cuatro *relaciones finales* $\forall y \in N, 1 \leq y \leq 4 : R_{fin,y,i}$ donde i es la posición, empezando por uno, en la que está situada la conjunción en la matriz, y sus ID contienen los id de los terrenos n_i y m_i asociados a las variables presentes en dicha conjunción no precedidas (x_i) y precedidas ($\neg x_i$) por una negación, respectivamente, y sus ganancias son INF. Además:

- $R_{fin,1,i}.ID$ contiene los id de n_{f-1}, fin_a y aux_{f-1} (si $f-1\ mod\ 2=0$) o aux_f (si no).
- $R_{fin,2,i}.ID$ contiene los id de m_{f-1}, fin_a y aux_{f-1} (si $f-1\ mod\ 2=0$) o aux_f (si no).
- $R_{fin,3,i}.ID$ contiene los id de n_{f-1}, fin_b y aux_{f-1} (si $f-1\ mod\ 2=0$) o aux_f (si no).
- $R_{fin,4,i}.ID$ contiene los id de m_{f-1}, fin_b y aux_{f-1} (si $f-1\ mod\ 2=0$) o aux_f (si no).

Es decir, para que una de estas relaciones sume su ganancia adicional a un jugador, éste necesitará haber elegido correctamente los terrenos asociados a todas las variables presentes en una conjunción de la matriz (la valoración de las variables que ha elegido hace que la fórmula sea verdadera) y, para asegurarnos de que se ha llegado al final de la partida siguiendo la estrategia que describiremos en *Demostración de la validez de la entrada transformada* y de que únicamente J_1 pueda tener acceso a la ganancia de las relaciones finales, haber comprado uno de los terrenos finales, un terreno normal del penúltimo nivel y el último terreno auxiliar que le corresponde a J_1 (aux_{f-1} o aux_f).

Finalmente, $d_{ini,1} = 1/100$, $d_{ini,2} = 2/100$ y $o = INF$. No es necesario que el valor de o sea tan elevado. Podría reducirse a $o = f + 2$ (pudiendo cambiar la ganancia de las relaciones finales a 1) y toda la demostración seguiría siendo válida, pero hemos decidido dejarlo en INF para que sea más fácil ver que no es posible que, si ambos jugadores están siguiendo la estrategia definida en las siguientes páginas, uno llegue al valor o de ganancia antes de haber comprado un terreno final.

Nótese que las relaciones que incluyen en sus ID el id de:

- n_i o m_i son las relaciones intermedias de los niveles i e $i+1$, las auxiliares del nivel $i+1$ y las finales.
- aux_i (donde $i \neq f$) son dos de las cuatro intermedias del nivel i, las auxiliares del nivel i, la mitad de las de niveles superiores ($\forall z \in \mathbb{N}, i < z < f, z\ mod\ 2 \neq i\ mod\ 2 : R_{aux,1,z}, R_{aux,2,z}$) y las finales.
- aux_f son las auxiliares del nivel f y las finales.
- fin_a o fin_b son las finales.

Vamos a tener esto en mente durante la demostración.

4.2.3. Estrategia de EJE

Queremos que los jugadores se vean obligados (si no quieren perder trivialmente la partida) a comprar un terreno determinado por turno, de acuerdo con el siguiente orden:

En los turnos 1, 2 y 3, los jugadores compran los terrenos iniciales. El terreno ini_a identificará a J_1 e ini_b a J_2. Tras haberlos comprado, los jugadores tendrán una ganancia de 1.

- Turno 1 $\Rightarrow$ J_1 compra ini_a.
- Turno 2 $\Rightarrow$ J_2 compra ini_b.
- Turno 3 $\Rightarrow$ J_1 compra ini_c.

k es el primer turno en el que se comprará un terreno del último nivel y es igual a 4 más tres veces el número de variables en el prefijo (es decir, el número de terrenos en total menos dos). Desde el turno 4 hasta el turno $k-1$, los jugadores comprarán de forma ordenada, en secuencias de tres turnos, los terrenos de cada nivel (hasta el penúltimo). El terreno auxiliar es el primero que tiene que ser comprado de cada nivel y su propósito es hacer perder un turno al jugador que tenga que comprarlo, de forma que se alterne el jugador al que le toca elegir primero un terreno normal (los que deciden los valores de las variables) del nivel actual. Así, simularemos la alternación de cuantificadores existenciales (elige primero J_1) y universales (elige primero J_2). Que un jugador no elija un terreno auxiliar cuando le toca conllevará a su derrota. Cuando los jugadores finalizan un nivel, su ganancia habrá sido aumentada en 1 y podrán permitirse los terrenos del siguiente nivel. Si el juego transcurre según la estrategia, los jugadores no van a poder ahorrar; en cada turno tendrán que gastar todo el dinero que disponen.

- Turno 4 $\Rightarrow$ J_2 compra aux_1.
- Turno 5 $\Rightarrow$ J_1 compra n_1 o m_1 (decidiendo así el valor de x_1).
- Turno 6 $\Rightarrow$ J_2 compra n_1 o m_1 (el que J_1 no haya comprado).
- Turno 7 $\Rightarrow$ J_1 compra aux_2.
- Turno 8 $\Rightarrow$ J_2 compra n_2 o m_2 (decidiendo así el valor de x_2).
- Turno 9 $\Rightarrow$ J_1 compra n_2 o m_2 (el que J_2 no haya comprado).
- Los siguientes turnos son similares a los turnos del 4 al 6 y del 7 al 9.

En el turno k, el jugador al que le toque (dependiendo de la paridad del número de variables del problema TQBF original) comprará el terreno auxiliar del último nivel (f). Ya no quedan variables por cuantificar y es irrelevante quién elija primero si comprar fin_a o fin_b, pero este último terreno auxiliar ayuda a que los jugadores no puedan salirse de la estrategia exitosamente. Finalmente, los jugadores comprarán los terrenos finales, que darán la victoria a J_1 si se han elegido valores que satisfagan la fórmula booleana. Si no, el juego acabará con la victoria de J_2 porque no quedan terrenos por comprar y J_1 no ha obtenido la ganancia requerida.

- Turno k, J_1 o J_2 (a quien le toque) compra aux_f.
- Turno $k+1$, J_1 o J_2 compra fin_a o fin_b.
- Turno $k+2$, J_1 o J_2 compra fin_a o fin_b (el que no haya sido comprado).

Formalmente, la estrategia será:
$\forall t \in \mathbb{N}^+, k = 3nvar + 4, i = (t-1)\ div\ 3:$
$(t = 1 \Rightarrow compra(J_1, ini_a)),$
$(t = 2 \Rightarrow compra(J_2, ini_b)),$
$(t = 3 \Rightarrow compra(J_1, ini_c)),$
$(t = k+1 \wedge nvar\ mod\ 2 = 1 \Rightarrow elige(J_1, fin_a, fin_b)),$
$(t = k+1 \wedge nvar\ mod\ 2 = 0 \Rightarrow elige(J_2, fin_a, fin_b)),$
$(t = k+2 \wedge nvar\ mod\ 2 = 1 \Rightarrow restante(J_2, fin_a, fin_b)),$
$(t = k+2 \wedge nvar\ mod\ 2 = 0 \Rightarrow restante(J_1, fin_a, fin_b)),$
$(3 < t \leq k \wedge i\ mod\ 2 = 1 \wedge t\ mod\ 3 = 1 \Rightarrow compra(J_2, aux_i))$
$(3 < t < k \wedge i\ mod\ 2 = 1 \wedge t\ mod\ 3 = 2 \Rightarrow elige(J_1, n_i, m_i)),$
$(3 < t < k \wedge i\ mod\ 2 = 1 \wedge t\ mod\ 3 = 0 \Rightarrow restante(J_2, n_i, m_i)),$
$(3 < t \leq k \wedge i\ mod\ 2 = 0 \wedge t\ mod\ 3 = 1 \Rightarrow compra(J_1, aux_i))$
$(3 < t < k \wedge i\ mod\ 2 = 0 \wedge t\ mod\ 3 = 2 \Rightarrow elige(J_2, n_i, m_i)),$
$(3 < t < k \wedge i\ mod\ 2 = 0 \wedge t\ mod\ 3 = 0 \Rightarrow restante(J_1, n_i, m_i)),$
donde t es el turno, $nvar$ es el número de variables en el prefijo, div es el operador de división entera, $compra(J, n)$ significa que el jugador J compra el terreno n, $elige(J, n, m)$ significa que J compra n o m a su elección, $restante(J, n, m)$ significa que J compra, de entre n y m, el que no haya sido comprado por el otro jugador, y $pasa(J)$ significa que J no compra ningún terreno en su turno.

4.2.4. Explicación informal mediante un ejemplo

Antes de ver la corrección de la reducción propuesta, en este apartado vamos a mostrar una transformación de una entrada concreta de TQBF a una de EJE y vamos a resolver informalmente el problema EJE obtenido. Ilustraremos el proceso usando tablas donde cada casilla representa un terreno ("ini a" representa el terreno ini_a) y su color indica su pertenencia (si es azul está en E_1, si es roja a E_2 y si es blanca sigue en T).

Entrada de TQBF:
$\exists x1 \forall x2 : (x1 \wedge x2) \vee (\neg x1)$
Entrada transformada:
$T = \{ini_a, ini_b, ini_c, n_1, m_1, n_2, m_2, aux_1, aux_2, aux_3, fin_a, fin_b\},$
$M = \{$
$R_{ini},$
$R_{inter,1,1}, R_{inter,2,1}, R_{inter,3,1}, R_{inter,4,1}, R_{inter,1,2}, R_{inter,2,2}, R_{inter,3,2}, R_{inter,4,2},$
$R_{aux,1,1}, R_{aux,2,1}, R_{aux,1,2}, R_{aux,2,2}, R_{aux,1,3}, R_{aux,2,3},$
$R_{fin,1,1}, R_{fin,2,1}, R_{fin,3,1}, R_{fin,4,1}, R_{fin,1,2}, R_{fin,2,2}, R_{fin,3,2}, R_{fin,4,2}$
$\},$
$d_{ini,1} = 1/100,\ d_{ini,2} = 2/100,\ o = 19$

En el primer turno, J_1 tiene que comprar el terreno ini_a si no quiere perder. Obtiene $g_1 = 3/100$. Con $d_1 = 1/100$ no puede permitirse ningún otro terreno y tendría como alternativa pasar. Si pasa, J_1 podría aprovechar para comprar ini_a, impidiendo a J_1 comprar en ningún otro turno porque $g_1 = 0$.

Cuando $t = 2$, $d_2 = 2/100$ y J_2 compra ini_b. Su única alternativa es pasar, y J_1 podría comprar ini_b dejando a J_2 sin opciones.

ini a	aux 1	aux 2	aux 3
ini b	n 1	n 2	fin a
ini c	m 1	m 2	fin b

Cuando $t = 3$, $d_1 = 3/100$ y J_1 compra ini_c. Su única alternativa es pasar, pero si J_2 compra después ini_c, J_1 no podría comprar ningún terreno durante varios turnos porque $g_1 = 3/10$ y el terreno más barato que quedaría en T cuesta 1. Antes de que pueda permitirse comprar un terreno, J_2 podría comprar n_1 y m_1, sumando INF a su ganancia por completar la relación intermedia del nivel 1 y ganando inmediatamente.

Cuando $t = 4$, $d_2 = 1$ y J_2 compra aux_1. Si decide comprar otro terreno o pasar, J_1 podría comprar aux_1, completando la relación auxiliar del nivel 1, sumando INF a su ganancia y ganando inmediatamente.

ini a	aux 1	aux 2	aux 3
ini b	n 1	n 2	fin a
ini c	m 1	m 2	fin b

Cuando $t = 5$, $d_1 = 1$ y J_1 elige comprar m_1, cuantificando la variable x_1 a $\perp$. Comprar n_1 está permitido, pero de esa forma no podría satisfacer la fórmula booleana $\forall x_2 \in \{\top, \perp\}$. Si decide pasar, estaría perdiendo la posibilidad de elegir primero y J_2 podría comprar n_1 el siguiente turno, impidiendo a J_1 satisfacer la fórmula. Incluso si, en un hipotético caso, el valor de la variable x_1 no fuese relevante para poder satisfacer la fórmula (por ejemplo, si x_1 no estuviera presente en la matriz), la acción de J_1 de pasar en lugar de elegir un valor hará que J_2 pueda comprar primero un terreno normal del nivel 1 y en su turno siguiente el restante del nivel 1 o el auxiliar del nivel 2, sumando INF a su ganancia y ganando.

Cuando $t = 6$, $d_2 = 1$ y J_2 compra el terreno normal restante del nivel 1 (n_1). No puede permitirse ningún otro terreno y su única alternativa sería pasar. Si hubiese pasado, J_1 podría comprar n_1, sumando INF a su ganancia por haber completado la relación intermedia del nivel 1 y ganando inmediatamente.

ini a	aux 1	aux 2	aux 3
ini b	n 1	n 2	fin a
ini c	m 1	m 2	fin b

Cuando $t = 7$, $d_1 = 2$ y J_1 compra aux_2. Si en su lugar compra un terreno normal del nivel 2, J_2 puede comprar aux_2, impidiendo que J_1 gane porque para obtener la ganancia adicional de las relaciones finales necesita haber comprado el último terreno auxiliar que le corresponde (aux_2).

Cuando $t = 8$, $d_2 = 2$ y J_2 compra n_2. En este caso su decisión es irrelevante porque el valor de x_2 no influye en la satisfacibilidad de la fórmula booleana una vez el valor de x_1 ha sido correctamente fijado. Si decide pasar, J_1 puede comprar un terreno normal del nivel 2. Independientemente de lo que haga J_2 el siguiente turno, J_1 podrá después comprar el terreno normal del nivel 2 restante o aux_3, ganando la partida tras completar una relación intermedia del nivel 2 o una auxiliar del nivel 3, respectivamente.

ini a	aux 1	aux 2	aux 3
ini b	n 1	n 2	fin a
ini c	m 1	m 2	fin b

Cuando $t = 9$, $d_1 = 2$ y J_1 compra m_2. Si en su lugar decide pasar, J_2 podría comprar m_2, ganando la partida por completar la relación intermedia del nivel 2.

Cuando $t = 10$, $d_2 = 3$ y J_2 compra aux_3. Si en su lugar decide comprar un terreno final o pasar, J_1 podría comprar aux_3, ganando la partida inmediatamente por haber completado la relación auxiliar del nivel 3 con una ganancia adicional de INF, aunque en este ejemplo esto no alteraría el resultado final ya que J_1 va a ganar igualmente porque la fórmula evalúa a $\top$.

ini a	aux 1	aux 2	aux 3
ini b	n 1	n 2	fin a
ini c	m 1	m 2	fin b

Cuando $t = 11$, $d_1 = 3$ y J_1 compra fin_a. Gana la partida porque su ganancia ha aumentado en 1 (ganancia de fin_a) más 1 por haber completado una de las relaciones finales referentes a la conjunción ($\neg x1$), llegando a $g_1 = 5 \geq o$.

ini a	aux 1	aux 2	aux 3
ini b	n 1	n 2	fin a
ini c	m 1	m 2	fin b

4.3. Demostración de la validez de la entrada transformada

Vamos a demostrar de manera formal que la solución de EJE para una entrada transformada a partir de cualquier entrada válida de TQBF es igual a la solución de TQBF para su entrada original.

El propósito de transformar la entrada de TQBF en una de EJE de esta forma es forzar en cada decisión que los jugadores elijan únicamente entre las acciones que acabamos de definir. En caso de que un jugador actúe de forma diferente, tenemos que asegurar que nunca va a ser más favorable para él respecto a la estrategia definida.

En la mayoría de los casos, esto va a significar que si un jugador hace un movimiento fuera de la estrategia, éste perderá independientemente de lo que haga a partir de ese punto si el otro jugador juega correctamente, pero en la compra de terrenos del último nivel nos vamos a encontrar acciones fuera de la estrategia que, aunque no lleven directamente a la derrota, no aportan ningún beneficio respecto a la estrategia. Vamos a demostrar por inducción que esto se verifica $\forall t \in \mathbb{N}^+$ demostrando que se verifica para los casos base y probando que si se verifica en un turno t se verifica también el siguiente turno $t+1$.

En la demostración, J_a representará a un jugador no determinado (J_1 o J_2), y J_b al otro, y g_a, g_b, d_a, d_b a las ganancias y cantidades de dinero de J_a y J_b. Además, para facilitar la demostración, usaremos el prefijo "¬"precediendo expresiones como $compra(J_1, ini_a)$ para indicar que la compra descrita por la expresión no ha sido realizada. Finalmente, hay que tener en cuenta que INF es mayor que $3f$ ya que es dinero suficiente para comprar todos los terrenos en T y hay tres terrenos en el nivel f.

4.3.1. Terrenos iniciales

Si $t = 1$ y $\neg compra(J_1, ini_a)$ entonces J_1 no puede comprar ningún otro terreno porque $d_1 = 1/100$ y los otros terrenos tienen un coste mayor, y tendría que pasar. Si en $t = 2$ $compra(J_2, ini_a)$ entonces en $t = 3$ J_1 sigue con $d_1 = 1/100$ porque $g_1 = 0$. J_1 no puede comprar más terrenos ni por lo tanto alcanzar $g_1 \geq o$. J_2 puede comprar en los siguientes turnos ini_b, n_1 y m_1, completando la relación intermedia del nivel 1, sumando INF a su ganancia y ganando la partida porque $g_2 \geq INF = o$.

Si $t = 2$ y $\neg compra(J_2, ini_b)$ entonces J_2 tendría que pasar porque el resto de nodos no comprados tienen un coste mayor que $d_1 = 2/100$. El siguiente turno, J_1 podría comprar ini_b, impidiendo a J_2 comprar ningún otro terreno. A partir de este punto, J_2 tendrá que pasar en todos sus turnos porque $g_2 = 0$ y no puede permitirse ningún terreno en T. J_1 puede comprar en los siguientes turnos n_1 y m_1, completando la relación intermedia del nivel 1 y ganando la partida.

Si $t = 3$ y $\neg compra(J_1, ini_c)$ entonces J_1 no puede comprar ningún otro terreno con $d_1 = 3/100$. Sabemos que tiene esa cantidad de dinero porque en el primer turno gastó todo su dinero en ini_a que tiene una ganancia de $3/100$. Si J_1 pasa y el turno siguiente $compra(J_2, ini_c)$, cuando $t = 5$ J_1 tendrá $d_1 = 6/100$ y cuando $t = 7$ J_1 tendrá $d_1 = 9/100$ (el doble y el triple de g_1) y únicamente podrá pasar de nuevo. Mientras tanto, en sus dos turnos siguientes J_2 podrá comprar n_1 y m_1, ganando la partida.

4.3.2. Terrenos intermedios

Vamos a tener como invariante que si no ha habido ninguna acción fuera de nuestra estrategia en los turnos anteriores, entonces en los turnos de J_1 y J_2 en los que se espera que compren su primer terreno de cada nivel i todos los terrenos de niveles inferiores han sido comprados, no se ha comprado ningún terreno de ese nivel o niveles superiores y $g_1 = g_2 = d_1 = d_2 = i$. Además, el primer turno de cada nivel i (en el que se tiene que comprar aux_i) es de J_1 si i es par y de J_2 si es impar.

Al final del turno 3, $g_1 = ini_a.g + R_{ini}.g = g_2 = ini_b.g = 1$ y $d_1 = d_2 = 0$ (en los tres primeros turnos, se gastaron todo el dinero que disponían). En el turno 4, $i = (t-1)\ div\ 3 = 1$, y al sumar las ganancias al dinero de los jugadores, $g_1 = g_2 = d_1 = d_2 = 1 = i$. Se han comprado únicamente los terrenos iniciales, que pertenecen a niveles inferiores a 1. $i = 1$ es impar y como en el turno 3 $compra(J_1, ini_c)$ en el turno 4 le tocará jugar a J_2. Por lo tanto, la invariante se cumple en el primer nivel.

Si los jugadores usan la estrategia que hemos definido hasta el turno t y es el primer turno en el nivel i donde $i = (t-1)\ div\ 3$ e $i \neq f$ (hay al menos un nivel más), $g_a = g_b = d_a = d_b = i$. Si desde este punto siguen usando esta estrategia, en los turnos t, $t+1$ y $t+2$ $compra(J_a, aux_i)$, $elige(J_b, n_i, m_i)$ y $restante(J_a, n_i, m_i)$, respectivamente. En el turno $t+3$, en el cual se comprará el primer nodo del nivel $i+1$ (porque $(t+3-1)\ div\ 3$ es igual a $i+1$), $g_a = i + aux_i.g + n_i.g = i + aux_i.g + m_i.g = i+1$ y $g_b = i + n_i.g = i + m_i.g = i+1$ porque todas las relaciones que incluyen en su ID los id de n_i, m_i o aux_i (las intermedias de los niveles i e $i+1$, las auxiliares de los niveles i, $i+1$ y las de niveles superiores a $i+1$ no correspondientes al jugador que ha comprado aux_i, y las finales) tienen en sus ID los id de los siguientes terrenos:

- Para las relaciones intermedias del nivel i, tanto n_i como m_i. Como J_1 y J_2 tienen uno cada uno, no pueden obtener la ganancia de estas relaciones.
- Para las relaciones intermedias del nivel $i+1$, los terrenos n_{i+1} y m_{i+1}. El invariante nos asegura que no se habían comprado previamente terrenos de niveles superiores a i y estamos suponiendo que en los turnos t, $t+1$ y $t+2$ tampoco, por lo que J_1 y J_2 no pueden obtener la ganancia de estas relaciones.
- Para las relaciones auxiliares del nivel i, el terreno inicial identificativo del jugador al que no le corresponda aux_i (en el caso de J_a, porque aunque compró aux_i en el turno t, sabemos que lo hizo siguiendo la estrategia, por lo que le corresponde) o aux_i (en el caso de J_b, porque sabemos que no lo puede tener).
- Para las relaciones auxiliares de niveles superiores a i, terrenos auxiliares de niveles superiores a i. No se han comprado terrenos de niveles superiores a i, por lo que J_1 y J_2 no pueden obtener la ganancia de estas relaciones.
- Para las relaciones finales, fin_a o fin_b. No se han comprado terrenos de niveles superiores a i e $i < f$, por lo que J_1 y J_2 no pueden obtener la ganancia de estas relaciones.

Además, se verifica trivialmente que en el turno $t+3$ (el primero del nivel $i+1$), como no se han comprado terrenos de niveles superiores a i entre los turnos t y $t+2$

y antes del turno t no se habían comprado terrenos de niveles superiores a $i-1$, no se ha comprado ningún terreno de nivel superior o igual a $i+1$. De forma similar, como se han comprado los 3 nodos del nivel i (n_i, m_i, aux_i) y antes del turno t se habían comprado todos los terrenos de niveles inferiores a i, se han comprado todos los terrenos de niveles inferiores a $i+1$. Finalmente, como el jugador que juega el primer turno de cada nivel depende de la paridad del nivel, si J_a juega en el turno t (nivel i), J_b tendrá que jugar en $t+3$ (nivel $i+1$). Como en $t+2$ $restante(J_a, n_i, m_i)$, en $t+3$ le tocará hacer una acción a J_b.

4.3.2.1. Terrenos intermedios auxiliares

Si $(3 < t < k \wedge i \ mod\ 2 = 1 \wedge t \ mod\ 3 = 1)$ o $(3 < t < k \wedge i \ mod\ 2 = 0 \wedge t \ mod\ 3 = 1)$ (J_a tendría que comprar el terreno auxiliar del nivel i e $i \neq f$) y $\neg compra(J_a, aux_i)$, como $d_a = i$ y el invariante nos asegura que no quedan terrenos de coste menor que i, J_a tiene dos opciones, $pasa(J_a)$ y $elige(J_a, n_i, m_i)$.

Si no había alcanzado $g_a \geq o$ antes, es trivial que si pasa tampoco lo alcanzará en el turno t. Si compra un terreno normal del nivel i, suma 1 a su ganancia porque $n_i = m_i = 1$ y no puede completar ninguna relación que contenga n_i o m_i ya que el invariante nos asegura que, como ningún jugador ha podido comprar un terreno del nivel i o superiores antes del turno t, no se han podido comprar los siguientes terrenos:

- Para las relaciones intermedias del nivel i, un terreno normal del nivel i.
- Para las relaciones intermedias del nivel $i+1$, un terreno normal del nivel $i+1$.
- Para las relaciones auxiliares del nivel $i+1$, el terreno auxiliar del nivel $i+1$.
- Para las relaciones finales, un terreno final.

Por lo tanto, J_b puede hacer $compra(J_b, aux_i)$ en $t+1$, sumando INF a su ganancia por haber completado una de las dos relaciones auxiliares del nivel i, ya que ha comprado todos los terrenos cuyos id están contenidos en sus ID, es decir, el terreno inicial identificativo del jugador al que no le corresponde el terreno aux_i (ini_a si $i \ mod\ 2 = 1$ o ini_b si $i \ mod\ 2 = 0$), aux_i, los terrenos auxiliares anteriores correspondientes a J_b y n_{i-1} (en el caso de $R_{aux,1,i}$) o m_{i-1} (en el caso de $R_{aux,2,i}$). Sabemos que ha comprado esos terrenos porque, por la hipótesis inductiva, no ha hecho ninguna acción inesperada en turnos anteriores a i y J_b ha comprado el terreno inicial identificativo correcto porque según la estrategia aux_i le correspondía a J_a. J_b gana por haber alcanzado $g_1 \geq INF = o$.

4.3.2.2. Primeros terrenos intermedios normales

Si $(3 < t < k \wedge i \ mod\ 2 = 1 \wedge t \ mod\ 3 = 2)$ o $(3 < t < k \wedge i \ mod\ 2 = 0 \wedge t \ mod\ 3 = 2)$ (J_a tendría que comprar el primer terreno normal del nivel i) y $\neg elige(J_a, n_i, m_i)$, J_a no puede comprar ningún terreno no esperado porque nuestra hipótesis inductiva nos asegura que aux_i ha sido comprado el turno anterior, y además el invariante

nos asegura que $g_a = d_a = i$ y que no quedan terrenos en T de niveles inferiores, por lo que los únicos terrenos que puede comprar son los normales del nivel i. Si $pasa(J_a)$, J_b puede $compra(J_b, n_i)$ porque como el invariante nos asegura que cuando J_b compró su primer terreno del nivel i (aux_i) tenía $d_b = g_b = i$, sabemos que en el turno $t+1$ tendrá los mismos valores de d_b y g_b.

En el turno $t+2$, $d_a = 2g_a = 2i$. Solo queda un terreno de coste i, m_i, y el resto tienen un coste mayor por lo que J_a puede comprar un solo terreno o pasar de nuevo. Las opciones de compra que tiene son el terreno restante del nivel i ($compra(J_a, m_i)$), de niveles superiores ($\forall x \in \mathbb{N}, x \leq 2i, x \geq i+1 : elige(J_a, n_x, m_x) \vee compra(J_a, aux_x)$), incluidos los finales (si $2i \geq f$, $elige(J_a, fin_a, fin_b)$). En $t+3$, $g_b = d_b = i+1$ y J_b puede $compra(J_b, m_i)$ si no la ha comprado J_a el turno anterior o $compra(J_b, aux_{i+1})$ en caso contrario.

Si $compra(J_b, m_i)$, $g_b \geq INF$ porque hay una relación intermedia del nivel i (una de las relaciones $\forall y \in N, 1 \leq y \leq 4 : R_{inter,y,i}$) con ganancia INF en M cuyo ID tiene exclusivamente los id de n_i y m_i, n_{i-1} o m_{i-1} (si $i \neq 1$), y, o bien ini_a y, si $i \ mod \ 2 = 0$, aux_i o ini_b y, si $i \ mod \ 2 = 1$, aux_i.

Sabemos que n_i y m_i pertenecen a J_b porque los ha comprado en los turnos $t+1$ y $t+3$. Como cuando $t \ mod \ 3 = 2$ un jugador elige el primer nodo normal de un nivel y cuando $t \ mod \ 3 = 0$ el otro jugador elige el restante, la estrategia que hemos definido obliga a que un jugador compre un terreno normal de cada nivel y J_b debe haber comprado n_{i-1} o m_{i-1}. Finalmente, aux_i ha debido ser comprado por J_b en el turno $t-1$ porque estamos suponiendo que J_a tendría que comprar el primer terreno normal del nivel i (por lo que $t \ mod \ 3 = 2$) y $(t-1) \ mod \ 3 = 1$.

Si $compra(J_b, aux_{i+1})$, $g_b \geq INF$ porque hay una relación auxiliar del nivel $i+1$ con ganancia INF en M cuyo ID tiene los id de los terrenos auxiliares correspondientes a J_b, ini_a si $(i+1) \ mod \ 2 = 1$ o ini_b si $(i+1) \ mod \ 2 = 0$ (el terreno inicial identificativo del jugador al que, según la estrategia, no le corresponde comprar el terreno aux_i) y n_i (un terreno normal del nivel anterior).

Sabemos que $compra(J_b, n_i)$ en el turno $t+1$ y que los id de sus terrenos auxiliares correspondientes pertenecen a J_b porque antes del turno t no han habido acciones inesperadas. Como en el turno t la estrategia indicaba que J_a tendría que haber comprado el primer terreno normal del nivel i, J_b ha tenido que haber comprado el terreno auxiliar del nivel i en el turno $t-1$ ($t \ mod \ 3 = 2$ luego $(t-1) \ mod \ 3 = 1$). El invariante nos asegura que el primer turno de cada nivel i (en el que se tiene que comprar aux_i) es de J_1 si i es par y de J_2 si es impar, por lo que el jugador al que le corresponde aux_{i+1} no puede ser J_b.

Por consiguiente, J_b puede alcanzar $g_b \geq INF \geq o$ en el turno $t+3$, ganando la partida. Tenemos que asegurarnos de que J_a no puede ganar en el turno $t+2$ llegando a $g_a \geq o = INF$. Si en el turno $t+2$:

(a) $compra(J_a, m_i)$, tendrá $g_1 = i + m_i.g = i+1$ porque no ha comprado todos los terrenos cuyos id están en los ID de las relaciones en M que incluyen m_i (las intermedias de los niveles i e $i+1$, las auxiliares del nivel $i+1$ y las finales).

- Para las intermedias del nivel i ($\forall x \in \mathbb{N}, x \in \{2, 4\} : R_{inter,x,i}$), no ha comprado n_i.

- Para las intermedias del nivel $i+1$ ($\forall x \in \mathbb{N}, x \in \{2,4\} : R_{inter,x,i+1}$), no ha comprado n_{i+1} ni m_{i+1}.
- Para las auxiliares del nivel $i+1$ ($R_{aux,1,i+1}$ y $R_{aux,2,i+1}$), no ha comprado aux_{i+1}.
- Para las relaciones finales, no ha comprado fin_a ni fin_b.

La imagen siguiente muestra el estado final de la partida si J_a elige este camino.

aux i	aux i+1	...	aux f
n i	n i+1	...	fin a
m i	m i+1	...	fin b

(b) $elige(J_1, n_x, m_x)$, tendrá $g_1 = i+1$ porque $i < x \leq 2i$ y porque no ha comprado todos los terrenos cuyos id estén en los ID de las relaciones en M que incluyen n_x o m_x.

- Para las relaciones intermedias del nivel x, no ha comprado n_x o m_x (el terreno restante).
- Para las relaciones intermedias del nivel $x+1$, no ha comprado n_{x+1} ni m_{x+1}.
- Para las relaciones auxiliares del nivel $x+1$, no ha comprado aux_{x+1}.
- Para las relaciones finales, no ha comprado fin_a ni fin_b.

aux i	...	aux x	...
n i	...	n x	...
m i	...	m x	...

(c) $compra(J_1, aux_x)$, en $t+4$ tendrá $g_1 = i$ y $d_1 = 3i - x \leq 2i - 1$ porque no ha comprado todos los terrenos cuyos id estén en los ID de las relaciones en M que incluyen aux_x ($R_{aux,1,x}$, $R_{aux,2,x}$ y $\forall z \in \mathbb{N}, x < z < f, z \ mod \ 2 \neq x \ mod \ 2 : R_{aux,1,z}, R_{aux,2,z}$). Es decir, no ha comprado terrenos auxiliares de niveles superiores a x (descartando $\forall z \in \mathbb{N}, x < z < f, z \ mod \ 2 \neq x \ mod \ 2 : R_{aux,1,z}, R_{aux,2,z}$) ni los terrenos n_{x-1} o m_{x-1} (descartando $R_{aux,1,x}$ y $R_{aux,2,x}$).

aux i	...	aux x	...
n i	...	n x	...
m i	...	m x	...

(d) $elige(J_1, fin_a, fin_b)$, en $t+4$ tendrá $g_1 = i+1$ y $d_1 = 3i - f + 2 \leq 2i$ porque $i < f \leq 2i$ y porque no ha comprado todos los terrenos cuyos id estén en los ID de las relaciones finales ya que no ha comprado n_{f-1} ni m_{f-1}.

aux i	...	aux f-1	aux f
n i	...	n f-1	fin a
m i	...	m f-1	fin b

(e) $pasa(J_1)$, en $t+4$ tendrá trivialmente $g_1 = i$ y $d_1 = 3i$.

De ninguna de estas formas consigue llegar a $g_1 \geq o = INF$. El valor de ganancia $g_1 = i + 2$ es el máximo al que puede llegar con cualquier estrategia desde la acción $pasa(J_1)$ en el turno t.

4.3.2.3. Segundos terrenos intermedios normales

Si $(3 < t < k \wedge i \ mod \ 2 = 1 \wedge t \ mod \ 3 = 0)$ o $(3 < t < k \wedge i \ mod \ 2 = 0 \wedge t \ mod \ 3 = 0)$ (J_a tendría que comprar el segundo terreno normal del nivel i) y $\neg restante(J_a, n_i, m_i)$, J_a no puede comprar ningún terreno no esperado conforme a la estrategia definida, porque nuestra hipótesis inductiva nos asegura que aux_i y el primer terreno normal del primer nivel han sido comprados en los dos turnos anteriores, el invariante que $g_a = d_a = i$ y que no quedan terrenos en T de niveles inferiores, por lo que el único terreno que se puede permitir es el restante del nivel i.

Si $pasa(J_a)$, J_b puede hacer $restante(J_b, n_i, m_i)$ porque como el invariante nos asegura que cuando J_b compró su primer terreno del nivel i (n_i o m_i) tenía $d_b = g_b = i$, sabemos que en el turno $t+1$ $d_b = g_b = i + n_i.g = i + m_i.g = i + 1 > n_i.c = m_i.c = i$. De esta forma, como J_b ha comprado los dos terrenos normales del nivel i (en los turnos $t-1$ y $t+1$) y un terreno inicial identificativo (la hipótesis inductiva asegura que J_1 ha comprado ini_a y J_2 ini_b), y el terreno auxiliar del nivel i no corresponde a J_b (el terreno auxiliar de un nivel corresponde al jugador que, siguiendo la estrategia, tiene que comprar el segundo terreno normal del mismo nivel; en este caso, J_a), sabemos que $\exists y \in \mathbb{N}, 1 \leq y \leq 4 : R_{inter,y,i}$ donde todos los terrenos cuyos identificadores estén en $R_{inter,y,i}.ID$ han sido comprados por J_b (J_b ha completado una relación intermedia del nivel i). Por lo tanto, como $R_{inter,y,i}.g = INF$ J_b habrá ganado la partida ya que $g_b \geq INF = o$.

4.3.3. Terrenos finales

4.3.3.1. Último terreno auxiliar

Hay que considerar por separado el terreno auxiliar del último nivel (cuando $(t = k \wedge i \ mod \ 2 = 1 \wedge t \ mod \ 3 = 1)$ o $(t = k \wedge i \ mod \ 2 = 0 \wedge t \ mod \ 3 = 1)$) del resto de auxiliares. Ambos jugadores tienen ganancia f y empezarán sus turnos con $d_1 = d_2 = f$ (nos lo asegura el invariante). En lugar de comprar el terreno auxiliar, J_a tiene como opciones pasar y comprar un terreno final (no quedan más terrenos). Independientemente de la decisión de J_a, J_b podrá comprar el turno siguiente aux_f, ganando la partida por completar la relación auxiliar del último nivel.

Sabemos que J_b completa esta relación porque, como la hipótesis inductiva nos asegura que se ha seguido la estrategia, ha comprado aux_f (y no le corresponde porque la estrategia indicaba que debería haberlo comprado J_a en el turno k), todos los terrenos auxiliares que le corresponden, uno de los terrenos iniciales identificativos y un terreno normal del nivel anterior.

Además, sabemos que J_a no pudo ganar el turno anterior al comprar un terreno final porque $g_a = f + fin_a = f + fin_b = f + 1 \leq INF = o$ y no puede completar ninguna relación final. Estas relaciones son las únicas que contienen en sus ID el id de un terreno final, y contienen también el id del último terreno auxiliar que le corresponde a un jugador, junto a su terreno inicial identificativo. J_a no ha comprado el último terreno auxiliar y sabemos que le correspondía porque la estrategia indicaba que tenía que comprarlo.

4.3.3.2. Primer terreno final

Si $(t = k + 1 \ \wedge \ nvar \ mod \ 2 = 1)$ o $(t = k + 1 \ \wedge \ nvar \ mod \ 2 = 0)$ (ya se ha comprado el último terreno auxiliar y un jugador tiene que comprar uno de los dos terrenos finales) y el jugador al que le toca comprar un terreno final (J_a) decide no comprarlo ($\neg elige(J_a, fin_a, fin_b)$), como el resto de terrenos han sido comprados únicamente podrá pasar. Si pasa, J_b podrá comprar uno de los terrenos finales en el turno $k+2$. El juego puede acabar si $J_b = J_1$ y cuando $elige(J_1, fin_a, fin_b)$ completa una relación final. De no ser así, en el turno $k + 3$, J_a tendrá dos opciones:

- Comprar el terreno final restante. Esto es equivalente a haber seguido la estrategia. J_b elegirá primero un terreno final, pero como ambos terrenos tienen la misma ganancia y el resto de terrenos presentes en todas las relaciones a las que pertenecen son los mismos, el orden de elección es irrelevante.

- Pasar de nuevo. J_b puede comprar el turno siguiente el terreno final restante. La única diferencia respecto a si hubiese seguido la estrategia es que J_b tiene ambos terrenos finales y J_a ninguno. Esto es trivialmente igual o peor para J_a porque no existen terrenos ni relaciones con ganancias negativas.

Por lo tanto, pasar en el turno $k+1$ es igual o peor que haber seguido la estrategia.

4.3.3.3. Segundo terreno final

Si $(t = k + 2 \wedge nvar \ mod \ 2 = 1)$ o $(t = k + 2 \wedge nvar \ mod \ 2 = 0)$ (queda únicamente un terreno final en T y J_a tiene que comprarlo) y J_a no lo compra ($\neg restante(J_a, fin_a, fin_b)$), J_b puede comprarlo el turno siguiente, acabando la partida porque $T = \emptyset$. La única diferencia respecto a si hubiese seguido la estrategia es que J_b tiene ambos terrenos finales y J_a ninguno. Por lo tanto, pasar en el turno $k + 2$ es igual o peor que haber seguido la estrategia.

4.3.4. Demostración de que la estrategia resuelve TQBF

Con esto, hemos finalizado la demostración de que la estrategia es óptima para ambos jugadores. Si J_1 no puede ganar siguiendo esta estrategia, no podrá ganar con ninguna. Similarmente, si usando esta estrategia J_2 no puede impedir que J_1 gane, no podrá impedírselo con ninguna. Por lo tanto, es trivial que existen ciertas decisiones que permiten a J_1 ganar en todos los casos si y solo si se cumple tal cosa considerando para ambos jugadores únicamente las acciones que siguen la estrategia definida. Ahora debemos demostrar que esto último es cierto exactamente cuando la instancia de TQBF de la que partimos en la reducción es positiva.

El invariante nos asegura que en el turno k se cumplía que $g_1 = g_2 = f$. Asumiendo que se sigue la estrategia, no es posible que JE acabe antes de que J_1 tenga la oportunidad de comprar un terreno final porque, incluso si le toca a J_2 comprar un terreno final antes que a J_1, solamente sumará 1 ($fin_a.g = fin_b.g = 1$) a su ganancia ya que, como no ha comprado el último terreno auxiliar correspondiente a J_1, no puede completar ninguna relación final. Como $g_1 = f \leq o$, $g_2 \leq f + 1 \leq o$ y $T \neq \emptyset$, el juego no acabará antes de que J_1 pueda comprar un terreno final.

Esto nos asegura que, cuando acaba el juego, J_1 habrá comprado aux_f (si f es par) y uno de los terrenos finales. No completó ninguna relación al comprar aux_f porque le correspondía ese terreno auxiliar ($ini_b \notin E_1$) y, en el momento de la compra, no había comprado ningún terreno final. Recordemos que hay cuatro relaciones finales por cada conjunción en la matriz, una por cada combinación de un terreno normal del nivel $f - 1$ y un terreno final, e incluyen, además de esos dos terrenos, el último terreno auxiliar correspondiente a J_1 y todos los terrenos n_i o m_i asociados a las variables situadas en la posición i en el prefijo precedidas o no precedidas por una negación, respectivamente, presentes en la conjunción. Al comprar un terreno final, como ha comprado un terreno normal del nivel $f - 1$ y el último terreno auxiliar que le corresponde (porque ha seguido la estrategia), completará al menos una relación final, sumando INF a su ganancia, alcanzando $g_1 \geq o$ y ganando la partida, si J_1 ha comprado los terrenos que representan las decisiones del valor de las variables de la conjunción asociada a esa relación final que hagan que sea verdadera. Si no puede completar ninguna relación final, perderá la partida cuando se haya comprado el último terreno en T.

Debido a la construcción de la instancia de EJE y a nuestra estrategia, J_1 ha podido decidir con la compra de terrenos normales el valor de las variables de la fórmula QBF (instancia de TQBF) situadas en posiciones impares del prefijo ($3 < t \leq k \wedge i \ mod \ 2 = 0 \wedge t \ mod \ 3 = 1 \Rightarrow compra(J_1, aux_i)$), mientras que J_2 ha podido decidir, al elegir primero y por tanto obligar a J_1 a quedarse con el terreno no elegido, el valor de las variables situadas en posiciones pares del prefijo ($3 < t < k \wedge i \ mod \ 2 = 0 \wedge t \ mod \ 3 = 2 \Rightarrow elige(J_2, n_i, m_i)$). Asumiendo que todas las decisiones tienen que ser concordantes con la estrategia, las decisiones de compra de cada jugador en la instancia de EJE representan directamente las valoraciones de las variables de los pasos correspondientes en la instancia original de TQBF (las compras de J_1 representan las valoraciones de las variables cuantificadas con $\exists$ y, de forma intercalada, las compras de J_2 representan las valoraciones de las variables cuantificadas con $\forall$). Por tanto, existe una manera de escoger dinámicamente las

decisiones de J_1 en función de cualquier posible secuencia de decisiones intercaladas de J_2 que resulta siempre en la victoria de J_1 si y solo si existe una manera de valorar dinámicamente de las variables en posiciones impares del predicado de la instancia de TQBF de la que partimos (las que están cuantificadas existencialmente; $\exists$) en función de cualquier posible valoración intercalada de las variables en posiciones pares (las cuantificadas universalmente; $\forall$) de forma que al menos una cláusula conjuntiva de la fórmula QBF considerada sea verdadera.

Por lo tanto, alguna manera de jugar garantiza que al menos una cláusula conjuntiva en la matriz podrá ser cuantificada de forma que sea verdadera (y la QBF, como la matriz es una disyunción de conjunciones, también lo será) siguiendo la alternación de cuantificadores del prefijo si y solo si alguna forma de jugar garantiza la victoria de J_1. La respuesta del problema TQBF y la del problema EJE resultante de su transformación serán iguales.

Capítulo 5

Resolución práctica

El programa que hemos implementado es una herramienta capaz de simular el problema EJE y de resolver problemas TQBF mediante una transformación de sus entradas a entradas de EJE de la forma que hemos visto previamente [11].

Hemos implementado la herramienta con un algoritmo minimax. Vamos a empezar explicando los métodos heurísticos que hemos utilizado para resolver instancias del problema de forma subóptima y la poda alfa-beta. Esto será necesario para poder tratar instancias de EJE de gran tamaño debido a la complejidad temporal del problema. Con estos conocimientos podremos entender el funcionamiento interno del programa y cómo lo puede utilizar un usuario. Finalmente usaremos una funcionalidad del programa que permite simular muchos problemas en masa para comparar el efecto que tienen varios parámetros del algoritmo minimax en su precisión y en el tiempo que tarda en dar una solución.

5.1. Métodos heurísticos y poda alfa-beta del algoritmo minimax

El algoritmo minimax tiene un coste de tiempo elevado porque el tiempo requerido para recorrer cada nivel del árbol escala exponencialmente con la profundidad. Como no es factible recorrer todo el árbol hasta los nodos terminales para encontrar una solución, se puede explorar hasta una determinada profundidad y usar una heurística para asignar a los nodos una puntuación entre, en nuestro caso, 0 y 1 de forma proporcional a cómo de buena es la situación en la que está el primer jugador en la configuración que representan. Hay que encontrar un equilibrio entre lo rápido que tarda en calcular la puntuación la función heurística y su precisión.

El uso de heurísticas para reducir el recorrido puede dar lugar a soluciones subóptimas, pero es necesario para hacer factible la exploración de árboles densos de profundidad elevada. Por ejemplo, un algoritmo minimax que decida el mejor movimiento en una partida de ajedrez llegando hasta los nodos terminales es inviable, especialmente cuando la partida acaba de empezar, pero uno que llegue hasta una determinada profundidad y use una heurística para valorar la situación en función

de las piezas restantes y sus posiciones en el tablero puede dar buenos resultados.

Vamos a usar otra optimización que no compromete la optimalidad de la solución; la poda alfa-beta. Esta poda elimina nodos que sabemos que no pueden afectar a la solución. Vamos a añadir las variables alfa y beta a cada nodo que definen el rango de puntuaciones que un nodo puede recibir de sus nodos hijos. En el nodo raíz, alfa y beta empezarán siendo el límite inferior y superior de la función heurística, respectivamente. Cuando se recorre el árbol hacia abajo, los nodos pasan los valores de alfa y de beta a sus hijos. Cuando se propagan las puntuaciones hacia arriba, los nodos padre actualizan sus valores de alfa y beta en función del valor heurístico recibido de sus hijos. Cuando un nodo MAX padre recibe la puntuación de uno de sus hijos, ésta reemplazará el valor de alfa del padre si es mayor. Similarmente, cuando un nodo MIN padre recibe la puntuación de uno de sus hijos, ésta reemplazará el valor de beta del padre si es menor. En ambos casos, después de que se actualicen los valores de alfa y beta, el nodo dejará de explorar el resto de sus hijos si $alfa \geq beta$ porque esto significa que, aunque pueda encontrar mejores valores para él (mayores que alfa si es el turno del primer jugador y menores que beta si es el turno del segundo), estos nunca van a ser elegidos por su nodo padre porque ya tiene un valor mejor. Vamos a ver un ejemplo para ilustrar la poda alfa-beta.

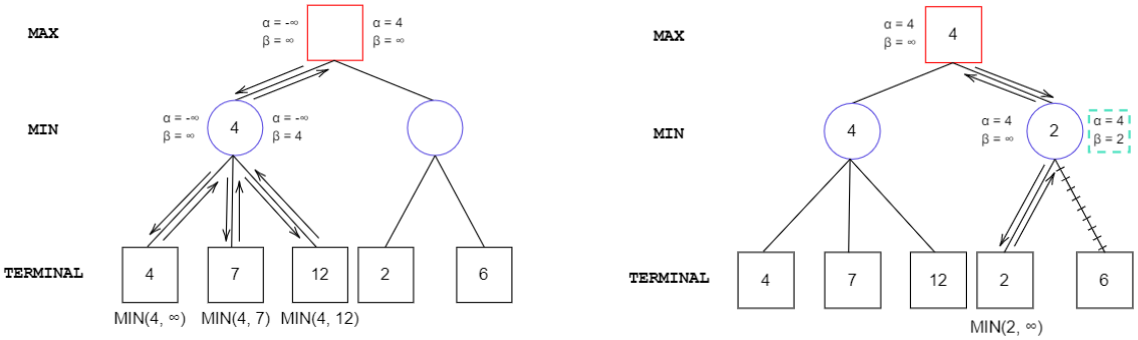


Figura 5.1: Ejemplo del algoritmo minimax con poda alfa-beta [7]

Los valores de alfa y beta se inicializan a -∞ e ∞ en el nodo raíz. Estos valores son pasados a los hijos. Cuando se propaga la puntuación del primer nodo terminal al primer nodo MIN, se actualiza beta a 4 porque es menor que su valor anterior (∞). Los dos otros hijos del nodo MIN no actualizan sus valores de alfa y beta porque son mayores que el que ya tiene. Cuando se propaga la puntuación del primer nodo MIN al nodo raíz (un nodo MAX), se actualiza alfa a 4 porque es mayor que su valor de alfa anterior (-∞). Los valores nuevos de alfa (4) y beta (∞) son pasados al segundo nodo MIN. Cuando se propaga la puntuación del cuarto nodo terminal, se actualiza beta a 2 porque es menor que su valor anterior (∞). Finalmente, como $alfa \geq beta$, no explora su hijo restante y propaga directamente su puntuación al nodo MAX, que como es menor que 4 la ignora. Este ejemplo muestra la utilidad de la poda alfa-beta; no es necesario que se explore el último nodo porque, incluso si pudiese mejorar a 2 (siendo menor que 2), no sería elegido por el nodo MAX porque no mejora a 4 (para mejorarlo, tendría que ser mayor que 4). Si no lo puede mejorar,

como en este caso, no será elegido por el segundo nodo MIN y no podrá llegar al nodo MAX.

5.2. Implementación de EJE

Hemos realizado la implementación en C++. Las clases *Terreno*, *Relacion*, *Accion* y *Configuracion* están definidas en *tipos.cpp*. Recordemos que una configuración de EJE es $(T, E_1, E_2, M, d_1, d_2, g_1, g_2, t, p_1, p_2)$. Los nodos del algoritmo minimax representan configuraciones y, por consiguiente, contienen todas las variables presentes en la tupla anterior. Las acciones que hemos definido anteriormente (compras de terrenos) serán los movimientos del algoritmo minimax.

Se genera la configuración inicial (el nodo raíz del árbol) con *generarRaiz()*. Esta función usa *entradaEJE()* y *transformarEntradaTQBF()* para generar la configuración a partir de cadenas que describen los dos tipos de entradas posibles; instancias de EJE y de TQBF. El procesamiento de *entradaEJE()* es trivial. El de *transformarEntradaTQBF()* se hace siguiendo la transformación de una entrada del problema TQBF a una de EJE que hemos visto anteriormente de forma teórica.

Para generar los hijos de un nodo, necesitamos saber qué acciones puede hacer el jugador al que le toque jugar en el turno asociado al nodo. Obtendremos esta lista de acciones con la función *accionesDesde()*, que puede generar una lista de todas las acciones posibles desde la configuración o únicamente tener en cuenta las acciones indicadas por la estrategia que hemos definido anteriormente. Esta segunda opción puede ser utilizada para que el algoritmo sea más eficiente únicamente cuando tratemos con configuraciones de EJE que proceden de instancias TQBF; de otra forma la estrategia que hemos definido previamente no tendrá sentido.

La mayor parte del procesamiento que hace *accionesDesde()* con la primera opción lo hace la función recursiva *accionesDesde()*. Esta función recorre todos los terrenos restantes en T y, por cada terreno, estudia qué opciones tiene si decide comprarlo o no. Para hacer esto, divide la exploración en dos. La primera exploración estudiará qué opciones tiene si decide no comprar el terreno que está siendo considerado. La segunda es realizada en una llamada recursiva a *accionesDesde()* y explora las opciones que tiene si decide comprarlo. Las posibles acciones encontradas se añaden al parámetro pasado por referencia *lista*. El número d es utilizado para asegurarnos de que el jugador actual se puede permitir los terrenos seleccionados y el mapa *visto* es utilizado para marcar los terrenos procesados de forma que no hayan acciones en las que un terreno sea comprado varias veces ni acciones repetidas. Aunque con tiempo y espacio ilimitados siempre daría una respuesta correcta, esta opción es estrictamente peor que la segunda cuando se ha transformado una entrada de TQBF.

El procesamiento que hace *accionesDesde()* con la segunda opción lo hace la función *estrategia()*, que simplemente devuelve una lista de todas las acciones que siguen la estrategia utilizando la lógica que hemos visto. Esto reduce drásticamente el número de hijos de cada nodo, haciendo el árbol de minimax mucho menos denso y posibilitando llegar a una mayor profundidad del árbol en un tiempo razonable, pero sólo es aplicable cuando se ha transformado la entrada de TQBF a la de EJE. Como

la estrategia es óptima, sabemos que los nodos omitidos no aportan información relevante.

Una vez hemos conseguido las acciones posibles desde una configuración, para obtener las siguientes configuraciones vamos a usar un constructor de Configuracion que recibe como parámetros una configuración y una acción. Este proceso es bastante simple y lo más notable es cómo se comprueba qué relaciones ha completado con una acción. Por cada terreno comprado, busca las relaciones que lo contienen y no tienen terrenos que el jugador actual no haya comprado, y suma sus ganancias a la del jugador.

La función *terminal()* calcula si la configuración actual es final y, si es así, qué jugador ha ganado la partida.

Hemos utilizado cinco heurísticas, que devuelven un valor entre 0 y 1 para cada configuración. La función *heuristica()* calcula estos valores.

Heurística 1: Tiene en cuenta únicamente la ganancia del primer jugador. Calcula el valor dividiendo la ganancia entre o. Sabemos que no puede ser superior a 1 porque si $g_1 \geq o$ J_1 habría ganado ya. Tiene una complejidad de tiempo constante ($O(1)$).

Heurística 2: Tiene en cuenta la ganancia de ambos jugadores usando dos valoraciones. La primera valoración es igual a la usada en la heurística 1, y es directamente proporcional al valor de g_1. La segunda valoración usa el método empleado en la heurística 1 y resta a 1 el resultado obtenido, y es inversamente proporcional al valor de g_2. Para obtener el valor heurístico se calcula la media de las dos valoraciones. Tiene una complejidad de tiempo constante.

Heurística 3: Tiene en cuenta la *ganancia esperada* de J_1. La ganancia esperada es igual a la ganancia de un terreno más una estimación de la ganancia que puede generar el terreno en el futuro indirectamente por formar parte de una relación. Es necesario realizar un precálculo cuando se genera la configuración inicial del problema después de que el programa haya leído la entrada. Para ello, en *generarRaiz()* se llama a *prepararHeuristica()* tras la creación de la configuración inicial. *prepararHeuristica()* recorre los terrenos y guarda en el mapa *contador* sus ganancias. Después recorre las relaciones y los terrenos que contienen, y por cada terreno suma en *contador* la ganancia de la relación entre el número de terrenos que contiene. Por ejemplo, un terreno con ganancia 1 contenido, junto a otros tres terrenos, en una relación con ganancia 2, será valorado como si su ganancia fuese $1 + 2/4 = 1{,}5$. Para evitar que ganancias extremadamente altas (presentes en configuraciones de EJE procedentes de haber transformado instancias de TQBF) reciban una importancia desmesurada, se reducen las ganancias superiores a o. Una vez ha sido realizado el precálculo, *heuristica()* recorre los terrenos comprados por J_1, sumando sus ganancias esperadas y finalmente normaliza el valor obtenido. Tiene una complejidad de tiempo lineal respecto al número de terrenos.

Heurística 4: Tiene en cuenta la ganancia esperada de ambos jugadores. Como en la heurística 2, la valoración de la ganancia esperada del segundo jugador es inversamente proporcional a su valor y se calcula el valor heurístico con la media de las dos valoraciones. Tiene una complejidad de tiempo lineal respecto al número de terrenos.

Heurística 5: El problema de las dos heurísticas anteriores es que cuando los dos

jugadores tienen terrenos pertenecientes a una misma relación, ésta no va a poder ser completada, pero las heurísticas siguen dando más valor a estos terrenos por estar en una relación. Esta heurística calcula la media entre las heurísticas 2 y 4 para dar un menor peso a la ganancia esperada. Tiene una complejidad de tiempo lineal respecto al número de terrenos (la suma de las complejidades de las heurísticas 2 y 4).

El programa tiene tres modos. Con el modo *manual*, el usuario especifica por la consola las acciones de ambos jugadores. El modo *simulación* se ejecuta sin requerir intervención del usuario y escribe en el fichero *out.txt* el resultado (y las acciones que han llevado a él) del problema suponiendo que ambos jugadores siguen una estrategia óptima, o, si esto fuera demasiado costoso, una estrategia subóptima aceptable utilizando una heurística. Este modo se puede usar para encontrar, si existen, valoraciones que satisfacen un problema TQBF. El modo *alterno* se utiliza para que un usuario juegue a través de la consola contra la máquina.

Además, como hemos mencionado al principio de este capítulo, el programa puede ejecutar muchas instancias de EJE y TQBF en masa para comparar la efectividad de las heurísticas y el efecto de la profundidad a la que se llegue en el árbol. Esto lo hace la función *testing()*, que utiliza entradas de EJE y TQBF generadas por *entradas.cpp*. El tamaño de las instancias generadas está determinado por varias constantes globales definidas en *entradas.cpp*.

La función *testing()* tiene dos modos. El primero es *testing por enfrentamientos* y compara todas las combinaciones posibles de los valores de profundidad máxima en un rango determinado y las distintas heurísticas. Con esta información se podrá deducir qué heurísticas y profundidades dan lugar a mejores resultados en la competición entre dos jugadores.

El segundo modo es *testing por comparación con la solución óptima* y clasifica los problemas por tipo (EJE o TQBF) y resultado ($\top$ o $\bot$) y mide la precisión y el tiempo de ejecución total y por categoría de problemas cuando se resuelven de forma óptima (profundidad ilimitada) y cuando se usan distintas heurísticas y se avanza hasta distintas profundidades, para poder razonar qué heurística y qué profundidad máxima elegir en cada situación. En este modo, en cada ejecución ambos jugadores usarán los mismos parámetros. El objetivo es encontrar los mejores parámetros para resolver problemas EJE o TQBF usando las soluciones óptimas como referencia, no encontrar los parámetros que den mejores resultados contra otras heurísticas subóptimas.

En la mayoría de los casos los parámetros obtenidos de ambas formas coincidirán, pero es posible que una heurística construida específicamente para obtener mejores resultados en el *testing por enfrentamientos* ganando a otras heurísticas aprovechando sus "puntos débiles" no obtenga los mejores resultados en el *testing por comparación con la solución óptima*.

5.3. Instrucciones de uso de la herramienta

El usuario puede especificar los valores de los parámetros de la herramienta escribiendo como argumentos el nombre o un prefijo del nombre del parámetro seguido de ":" y del valor del parámetro, sin espacios. Si el usuario ejecuta el programa sin ningún argumento, podrá especificar los valores de los parámetros por consola. Los parámetros no especificados tendrán un valor por defecto. Los valores disponibles para los siguientes parámetros están descritos a continuación:

estrategia: y o n (n por defecto). Con y se analizarán únicamente las acciones que siguen la estrategia, y es necesario que la entrada de EJE haya sido transformada a partir de una de TQBF. Con n se tienen en cuenta todas las opciones.

modo: s, m o a (s por defecto) para los modos simulación, manual o alterno, respectivamente.

heuristica: $h \in \mathbb{N}, 1 \leq h \leq 5$ (5 por defecto). Número de la heurística que utilizar. Se pueden especificar dos valores, uno para cada jugador (por ejemplo, heuristica:3:5). Si solo se especifica uno, será el mismo para ambos jugadores (por ejemplo, heuristica:4).

profundidad: $d \in \mathbb{N}, 0 \leq d < 10$ (0 por defecto). Profundidad máxima desde el nodo actual a la que llegar antes de aplicar una heurística. 0 indica que la profundidad será ilimitada. Como con el parámetro anterior, se puede especificar un único valor para ambos jugadores o uno para cada jugador.

pases: $p \in \mathbb{N}, 0 \leq p$ (2 por defecto). Constante del problema que indica el número de turnos consecutivos que puede pasar un jugador sin perder la partida automáticamente.

Para que la herramienta genere la configuración inicial de EJE a partir de una entrada de EJE, hay que pasar la entrada como un argumento del programa primero escribiendo "eje" o un prefijo de "eje". A continuación, se especifica el conjunto de terrenos T. Cada terreno se representa con su id, su coste y su ganancia, separados por comas y entre paréntesis. Después, se especifica el conjunto de relaciones M. Cada relación se representa, también entre paréntesis, con su ganancia seguida de los id de todos los terrenos que contiene, donde la ganancia y cada terreno están separados entre sí por comas. Después se especifica el dinero inicial del primer jugador ($d_{ini,1}$), del segundo ($d_{ini,2}$) y el valor de ganancia objetivo (o). Toda la entrada tiene que estar entrecomillada para que se lea como un solo argumento y "eje", T, M, $d_{ini,1}$, $d_{ini,2}$ y o tienen que estar separados entre sí por ":". Por ejemplo,"eje:(a, 1, 2) (b, 1.5, 2)(c,2,0.5):(0.5, a, c) (0.5, b, c):1:1.5:3".

Para que genere la instancia de EJE inicial transformando una entrada de TQBF, hay que pasar la entrada como un argumento escribiendo "tqbf" o un prefijo de "tqbf", seguido de una especificación del prefijo y de la matriz. Como con la entrada de EJE, toda la entrada tiene que estar entrecomillada y "tqbf", el prefijo y la matriz tienen que estar separados entre sí por ":". La especificación del prefijo es una secuencia de nombres de variables separados por espacios. Las variables cuyos nombres están en una posición impar y par están cuantificadas de forma existencial y universal, respectivamente. La matriz es una disyunción de conjunciones donde $\vee$ es "v", $\wedge$ es "^" y $\neg$ es "!" que contiene únicamente variables presentes en la matriz. Un ejemplo

de una entrada de TQBF en este formato es "tqbf:x1 x2:(x1 ^ x2) v (!x2)".

Finalmente, para probar muchas instancias de EJE y TQBF y obtener estadísticas (guardadas en *out.txt* y, en formato de tablas de látex, *latex.txt*) que nos permitan elegir razonadamente los valores de los parámetros que nos convengan, se puede pasar como argumento "testing" (o un prefijo de "testing") seguido de ":", el número de instancias de EJE que estudiar, ";" y el número de instancias de TQBF que estudiar. Por ejemplo, test:2:2.

Se pueden especificar los siguientes parámetros:

modo: e para el modo *testing por enfrentamientos*, o *testing por comparación con la solución óptima* por defecto.

escala: $e \in \mathbb{N}$ (10 por defecto). Se tiene que especificar para ambos jugadores (por ejemplo, escala:9:7). Determinará el tamaño de las instancias generadas, junto a unas constantes globales en *entradas.cpp*.

analizar: $h \in \mathbb{N}, 1 \leq h \leq 5$ (5 por defecto). Número de la heurística que queremos analizar con más detalle. Se usa exclusivamente en el modo *testing por enfrentamientos*.

Estas pruebas usan siempre el modo simulación, la estrategia que hemos definido para resolver instancias provenientes de TQBF y estudian los resultados obtenidos con todas las heurísticas. Los parámetros *pases* y *profundidad* se pueden especificar de la forma que hemos visto anteriormente.

Ejemplos:

estrategia:y modo:a "tqbf:x1 x2:(x1 ^ x2) v (!x2)"

estr:y "tqbf:x1 x2 x3 x4 x5 x6:(x1 ^ x2) v (x3 ^ x4) v (x5 ^ x6) v (x1 ^ !x2 ^ !x4 ^ !x6)"

"eje:(a, 1, 2) (b, 1.5, 2)(c,2,0.5):(0.5, a, c) (0.5, b, c):1:1.5:3"

estrategia:y modo:a "tqbf:x1 x2 x3 x4 x5 x6:(x1 ^ x2) v (x3 ^ x4) v (x5 ^ x6)"

estrategia:y "tqbf:x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10:(x6 ^ !x9 ^ x1) v (x1 ^ !x3 ^ !x5 ^ x2 ^ !x4 ^ !x6) v (x1 ^ x10 ^ x1 ^ !x9 ^ x8) v (!x6 ^ x5) v (x3 ^ x4)"

modo:m "eje:(a, 1, 2) (b, 1.5, 2.5)(c,2,0.5):(3, a, b) (1, b, c):1:1.5:3"

escala:8:6 test:1:2

prof:3 pas:1 esc:12:9 mod:e analizar:5 test:5:5

5.4. Rendimiento

Para medir el rendimiento de la herramienta que hemos desarrollado y comparar las distintas heurísticas y valores de profundidad máxima, primero hemos utilizado *testing por comparación con la solución óptima*. Hemos resuelto de forma óptima 50 instancias de EJE y de TQBF ejecutándolas con el modo simulación y profundidad ilimitada para ambos jugadores (habiendo transformado previamente las instancias de TQBF a instancias de EJE), y hemos comparado estas soluciones con las que se obtienen al ejecutar las instancias con el modo simulación y todas las combinaciones de profundidades entre 1 y 6 y las distintas heurísticas, obteniendo así la precisión

(porcentajes de aciertos) de cada combinación de parámetros.

Recordemos que, en este tipo de testing, en cada ejecución de una instancia ambos jugadores usan la misma profundidad y la misma heurística ya que el objetivo no es ver qué combinaciones subóptimas de parámetros ganan más contra otras, sino ver qué combinaciones obtienen resultados más similares a los óptimos. *Testing por comparación con la solución óptima* se usa para ver qué combinaciones de parámetros producen mejores resultados para resolver instancias de problemas EJE y TQBF, no para competir contra otras combinaciones.

Basándonos en los datos obtenidos (añadidos a continuación) podemos deducir que para resolver instancias de EJE (procedentes o no de instancias de TQBF) de estas características las heurísticas 4 y 5 obtienen mejores resultados, especialmente a niveles bajos de profundidad, sin repercutir negativamente en el tiempo de ejecución. La profundidad, como era previsible teóricamente, repercute negativamente en el tiempo de ejecución pero mejora la precisión, especialmente para instancias de EJE generadas de forma directa. La profundidad idónea para una situación dependerá de lo rápida que deba ser la ejecución y del valor de precisión que se requiera, pero en general no merece la pena aumentar la profundidad ya que esto hace que el tiempo de ejecución aumente de forma muy desproporcionada respecto a la precisión.

profundidad:6 escala:14:14 testing:50:50

100 instancias

50 de EJE (de 14 terrenos)

$\top$: 33

$\bot$: 17

50 de TQBF (de 14 variables, cuyas instancias de EJE asociadas tienen 48 terrenos)

$\top$: 23

$\bot$: 27

Total			
Profundidad	Heurística	Tiempo	Aciertos
1	1	3.667s	75 %
1	2	3.287s	80 %
1	3	3.608s	78 %
1	4	3.311s	96 %
1	5	3.497s	96 %
2	1	6.912s	81 %
2	2	7.515s	80 %
2	3	7.248s	96 %
2	4	6.73s	96 %
2	5	7.411s	96 %
3	1	12.883s	80 %
3	2	13.276s	81 %
3	3	13.155s	97 %
3	4	13.48s	97 %
3	5	12.896s	97 %
4	1	23.983s	81 %
4	2	24.505s	80 %
4	3	25.325s	97 %
4	4	24.596s	97 %
4	5	25s	96 %
5	1	35.029s	80 %
5	2	34.701s	81 %
5	3	36.054s	97 %
5	4	35.886s	97 %
5	5	35.586s	97 %
6	1	97.473s	81 %
6	2	96.671s	81 %
6	3	98.686s	97 %
6	4	98.941s	97 %
6	5	98.564s	97 %

EJE			
Profundidad	Heurística	Tiempo	Aciertos
1	1	0.182s	88 %
1	2	0.253s	98 %
1	3	0.111s	88 %
1	4	0.145s	98 %
1	5	0.228s	98 %
2	1	0.523s	100 %
2	2	0.613s	98 %
2	3	0.407s	98 %
2	4	0.425s	98 %
2	5	0.523s	98 %
3	1	2.068s	98 %
3	2	2.023s	100 %
3	3	2.083s	100 %
3	4	2.112s	100 %
3	5	2.101s	100 %
4	1	8.417s	100 %
4	2	8.792s	98 %
4	3	8.46s	100 %
4	4	8.441s	100 %
4	5	8.555s	98 %
5	1	13.483s	98 %
5	2	13.487s	100 %
5	3	13.273s	100 %
5	4	13.39s	100 %
5	5	13.344s	100 %
6	1	70.418s	100 %
6	2	69.802s	100 %
6	3	70.076s	100 %
6	4	70.002s	100 %
6	5	69.608s	100 %

TQBF			
Profundidad	Heurística	Tiempo	Aciertos
1	1	3.485s	62 %
1	2	3.034s	62 %
1	3	3.497s	68 %
1	4	3.166s	94 %
1	5	3.269s	94 %
2	1	6.389s	62 %
2	2	6.902s	62 %
2	3	6.841s	94 %
2	4	6.305s	94 %
2	5	6.888s	94 %
3	1	10.815s	62 %
3	2	11.253s	62 %
3	3	11.072s	94 %
3	4	11.368s	94 %
3	5	10.795s	94 %
4	1	15.566s	62 %
4	2	15.713s	62 %
4	3	16.865s	94 %
4	4	16.155s	94 %
4	5	16.445s	94 %
5	1	21.546s	62 %
5	2	21.214s	62 %
5	3	22.781s	94 %
5	4	22.496s	94 %
5	5	22.242s	94 %
6	1	27.055s	62 %
6	2	26.869s	62 %
6	3	28.61s	94 %
6	4	28.939s	94 %
6	5	28.956s	94 %

Para averiguar qué heurísticas y profundidades dan mejores resultados en competiciones contra otras heurísticas, hemos comparado el porcentaje de victorias y tiempo de ejecución de las heurísticas usando *testing por enfrentamientos*. Por cada una de las 50 instancias de EJE y de TQBF generadas, se han realizado 2 ejecuciones con el modo simulación por cada par de combinaciones de un nivel de profundidad (entre 1 y 3) y una heurística (una combinación de parámetros para cada jugador). En la primera ejecución de una instancia con cada par de combinaciones de parámetros la primera combinación es asignada al jugador 1, y en la segunda ejecución al jugador 2. El propósito de esto es conseguir una igualdad de oportunidades entre todas las combinaciones de parámetros, ya que en cada instancia hay un jugador que parte con ventaja: el jugador que ganaría si ambos jugadores usaran una estrategia óptima. Esto explica la baja varianza de los porcentajes de victorias; en muchas

instancias hay un jugador que gana prácticamente siempre independientemente de la calidad de la heurística utilizada debido a lo favorable que es su posición respecto al otro jugador.

Además de los porcentajes de victorias y el tiempo de ejecución total de cada combinación de parámetros para todas las instancias, las instancias de EJE y las de TQBF, como hemos añadido "analizar:5" en el comando, la herramienta va a mostrar de forma agregada (haciendo la media de los resultados de todas las profundidades contra todas las profundidades) los porcentajes de victorias de la heurística 5 contra todas las posibles heurísticas opuestas.

Usando este modo, como no requiere una resolución óptima de cada instancia, podemos analizar instancias de mayor tamaño en un tiempo razonable. Como era previsible, las heurísticas que obtienen mejores resultados son también la 4 y la 5. En instancias de EJE (cuyos árboles de resolución tienen menor profundidad pero mayor densidad), la heurística 5 tiene mejores resultados.

profundidad:3 escala:20:20 modo:e analizar:5 testing:50:50

100 instancias

50 de EJE (de 20 terrenos)

50 de TQBF (de 20 variables, cuyas instancias de EJE asociadas tienen 66 terrenos)

Total			
Profundidad	Heurística	Tiempo	Victorias
1	1	135.04s	39 %
1	2	141.90s	44 %
1	3	125.97s	45 %
1	4	128.16s	55 %
1	5	142.20s	55 %
2	1	388.95s	42 %
2	2	384.05s	44 %
2	3	368.48s	54 %
2	4	367.80s	54 %
2	5	383.85s	56 %
3	1	1746.99s	44 %
3	2	1836.96s	44 %
3	3	1706.71s	55 %
3	4	1741.10s	55 %
3	5	1825.78s	56 %

EJE			
Profundidad	Heurística	Tiempo	Victorias
1	1	32.95s	42 %
1	2	41.19s	51 %
1	3	26.78s	40 %
1	4	27.44s	50 %
1	5	41.13s	51 %
2	1	177.66s	49 %
2	2	172.96s	53 %
2	3	157.04s	48 %
2	4	156.86s	48 %
2	5	170.55s	53 %
3	1	1410.81s	52 %
3	2	1500.13s	53 %
3	3	1369.80s	50 %
3	4	1400.22s	50 %
3	5	1490.94s	53 %

TQBF			
Profundidad	Heurística	Tiempo	Victorias
1	1	102.08s	36 %
1	2	100.71s	36 %
1	3	99.19s	49 %
1	4	100.72s	60 %
1	5	101.07s	60 %
2	1	211.29s	36 %
2	2	211.09s	36 %
2	3	211.44s	60 %
2	4	210.94s	60 %
2	5	213.29s	60 %
3	1	336.18s	36 %
3	2	336.82s	36 %
3	3	336.91s	60 %
3	4	340.88s	60 %
3	5	334.83s	60 %

Heurística 5	
Heurística opuesta	Victorias
1	65.00 %
2	62.00 %
3	54.17 %
4	51.00 %
5	50.00 %

Capítulo 6

Caso de estudio

En este capítulo, usaremos la herramienta que hemos implementado para asistir en la toma de decisiones en una situación de expansión empresarial. Debido a la dificultad de obtener los datos necesarios para representar una situación de expansión real, hemos decidido usar datos ficticios pero, en la medida de lo posible, coherentes. Es necesario conocer algunos términos básicos de economía para entender el caso de estudio.

Earnings before interest, taxes, depreciation and amortization (EBITDA) es un indicador de la rentabilidad de una empresa o negocio y es calculado restando a los ingresos los costes de los bienes vendidos y los costes generales de administración [2].

Enterprise value (EV) es una medida del valor total de una empresa o negocio para sus acreedores financieros y para los accionistas [1]. EV puede ser considerado el precio teórico de una empresa [5].

El *múltiplo EBITDA* es una medida que compara el EV de una compañía con su EBITDA anual o una estimación futura del mismo [4]. Se calcula dividiendo el EV entre el EBITDA de una compañía. Este múltiplo suele ser común entre empresas pertenecientes a la misma industria, y por ello es común usar el valor del múltiplo EBITDA de una industria para calcular una aproximación del EV de una empresa perteneciente a dicha industria a partir de su EBITDA.

Vamos a estudiar una situación de expansión empresarial simplificada en la que una empresa A quiere expandirse por cinco municipios de la Comunidad de Madrid comprando gimnasios. Sabe que hay una empresa B que tiene los mismos planes, y conocemos el presupuesto de ambas. Queremos saber si la empresa A puede alcanzar una determinada cantidad de EBITDA anual antes que la empresa B.

Cada terreno será un gimnasio. Hemos utilizado precios reales de traspasos en la zona como referencias para los costes de los terrenos. El término "ganancias" que usamos de forma genérica en nuestro problema será, en esta situación, los EBITDA, y supondremos que estos valores se mantendrán constantes después del traspaso. Para obtener las ganancias de los terrenos, hemos usado el múltiplo EBITDA de la industria de gimnasios, centros de fitness y spas (12.27 [9]) y el precio de los traspasos como EV. La instancia del problema tendrá una relación por cada ciudad

que contiene todos los terrenos asociados a los restaurantes en la ciudad. El valor de ganancia de las relaciones será el 10 % de la suma de las ganancias de los terrenos que contienen. Estas relaciones simbolizan el beneficio de tener una situación de monopolio en una zona.

Situación de expansión empresarial

Madrid: 7 traspasos disponibles (400 000 €, 390 000 €, 200 000 €, 150 000 €, 145 000 €, 39 000 €, 37 000 €)

Móstoles: 5 traspasos disponibles (360 000 €, 190 000 €, 140 000 €, 100 000 €, 25 000 €)

Getafe: 5 traspasos disponibles (310 000 €, 200 000 €, 160 000 €, 110 000 €, 90 000 €)

Alcalá de Henares: 3 traspasos disponibles (220 000 €, 180 000 €, 120 000 €)

Leganés: 3 traspasos disponibles (250 000 €, 150 000 €, 110 000 €)

Presupuesto inicial de ambas empresas: 500 000 €

EBITDA anual objetivo: 250 000 €

Para ejecutar esta instancia del problema EJE en la herramienta, vamos a usar el siguiente comando:

pa:10 he:5 pr:5 “eje:(m1,400,32.599837) (m2,390,31.7848411) (m3,200,16.2999185) (m4,150,12.2249389) (m5,145,11.8174409) (m6,39,3.17848411) (m7,37,3.01548492) (mo1,360,29.3398533) (mo2,190,15.4849226) (mo3,140,11.409943) (mo4,100,8.14995925) (mo5,25,2.03748981) (g1,310,25.2648737) (g2,200,16.2999185) (g3,160,13.0399348) (g4,110,8.96495518) (g5,90,7.33496333) (a1,220,17.9299104) (a2,180,14.6699267) (a3,120,9.7799511) (l1,250,20.3748981) (l2,150,12.2249389) (l3,110,8.96495518) : (136.1,m1,m2,m3,m4,m5,m6,m7) (80.5,mo1,mo2,mo3,mo4,mo5) (86,g1,g2,g3,g4,g5) (52,a1,a2,a3)(51,l1,l2,l3) : 500 : 500 : 250”

Con este comando vamos a ejecutar en el modo simulación la instancia del problema, tratando de resolverlo usando para ambos jugadores la heurística 5 y recorriendo en cada decisión el árbol minimax hasta una profundidad de 5 nodos. El número máximo de turnos (en nuestro caso, años) que puede pasar un jugador sin perder la partida son 10. Hemos obtenido que, con estas especificaciones, J_1 ganará la partida.

Debido a las dimensiones del caso de estudio no es factible resolverlo de forma óptima. Aún así, en el capítulo anterior hemos analizado 50 instancias de EJE (de 14 terrenos, en lugar de los 23 de la instancia que estamos estudiando) y hemos visto que las resoluciones de esas instancias con un algoritmo minimax que usa la heurística 5 y una profundidad de 5 nodos para ambos jugadores dan lugar a los mismos resultados. En las 33 instancias de EJE cuyos resultados son $\top$ que hemos analizado J_1 gana cuando ambos jugadores realizan los movimientos que les indica un algoritmo minimax subóptimo con estos parámetros, y similarmente para las instancias cuyos resultados son $\bot$, y J_2.

Por lo tanto, pese a que no haya sido factible analizar el rendimiento de la herramienta con instancias de EJE de 23 terrenos (construidas de forma directa y no transformadas de instancias de TQBF), consideramos que es razonable suponer que la primera empresa considere que esta resolución subóptima es suficientemente precisa.

En la resolución heurística, la partida acaba en el siguiente estado de éxito 20 años después del primer traspaso:

t:42
d_1:400.972291
d_2:287.387941
g_1:250.755501
g_2:132.436838
T=$\emptyset$
E_1={g1, l3, a3, mo2, mo1, mo5, g3, m3, m6, m7, l2, g4, m2, l1}
E_2={g2, m1, m5, a2, m4, mo3, mo4, g5, a1}

La herramienta recomienda que, para poder llegar a esta situación de éxito, la primera empresa empiece por el tercer gimnasio de Leganés, el tercero de Alcalá de Henares, el quinto de Móstoles y el sexto y séptimo de Madrid.

Capítulo 7

Conclusión y Trabajo Futuro

Este Trabajo de Fin de Grado ha proporcionado una base sólida para el análisis de expansión empresarial desde una perspectiva teórica y práctica, y ha impulsado el desarrollo académico del autor en el área de la complejidad computacional.

En este estudio hemos demostrado que el problema de expansión empresarial es un problema PSPACE-completo, utilizando primero un algoritmo minimax para demostrar su pertenencia a PSPACE y después reduciendo polinómicamente el problema PSPACE-completo TQBF a nuestro problema.

Hemos implementado una herramienta práctica para poder abordar el análisis y la resolución de instancias del problema de expansión empresarial. Debido a la intratabilidad del problema estudiado, hemos utilizado métodos heurísticos (concretamente, un algoritmo minimax con poda alfa-beta) para poder resolver instancias de tamaños significativos en un tiempo razonable.

Con esta herramienta hemos ejecutado una gran cantidad de instancias de tanto el problema de expansión como, tras una transformación previa, del problema TQBF generadas de forma artificial. Estas ejecuciones nos han servido para identificar el impacto de la elección de los parámetros de entrada en el tiempo y en la precisión de la herramienta. Además, hemos utilizado esta herramienta para realizar nuestro caso de estudio, usando una versión simplificada de un problema de expansión empresarial.

Realizando el caso de estudio, nos hemos dado cuenta de la dificultad de trasladar un caso real de expansión empresarial a una instancia de nuestro problema. Por ejemplo, en un caso real el valor de ganancia de cada edificación es desconocido y realizar una proyección futura suele ser un proceso muy complejo. Las relaciones de nuestro problema son ganancias adicionales que surgen por haber realizado ciertas edificaciones sinérgicas debido a, por ejemplo, dar lugar a situaciones de monopolio y son incluso más complejas de medir. Además, en una situación real no se suele contar con información total del adversario. Debido a esto, hemos tenido que usar datos ficticios pero relativamente razonables.

Estas dificultades hacen que nuestro problema sea insuficiente para abordar la mayoría de casos reales de expansión empresarial. Pese a esto, esperamos que este caso de estudio haya servido para destacar la importancia de contar con buenas

herramientas computacionales capaces de abordar problemas intratables como el nuestro en la toma de decisiones empresariales, y que resulte útil e interesante para otros investigadores.

Originalmente, nuestro juego de expansión empresarial (JE) tenía unas condiciones de finalización distintas. En lugar de terminar cuando cualquier jugador alcanzase una determinada cantidad de dinero o no quedasen terrenos disponibles para comprar, JE terminaba cuando el *primer* jugador alcanzase una determinada cantidad de dinero o no quedasen terrenos disponibles para comprar. Esto es más fiel a lo que verdaderamente preocupa a una empresa acerca de una expansión; no es importante quién alcance el dinero objetivo primero, saber con certeza que es alcanzable es suficiente. Habíamos realizado la mayor parte del análisis de complejidad con esta versión alternativa del problema y parecía que de esta forma iba a ser también PSPACE-completo, pero este detalle aparentemente pequeño hacía que la demostración fuera considerablemente más extensa sin aportar prácticamente nada de interés a la misma. Por ello, decidimos usar la otra versión del problema.

Respecto al trabajo futuro, una posibilidad sería terminar de explorar esta versión alternativa del problema, o alguna generalización de nuestro problema que sea más fiel a la realidad. Trivialmente, cualquier generalización de EJE tendrá por lo menos la misma dificultad que el problema original (PSPACE), pero podría tener una dificultad superior (como EXPTIME o EXPSPACE), añadiendo un interés adicional a su estudio. Otra forma de continuar avanzando con este trabajo es estudiar el uso de métodos heurísticos alternativos a la poda alfa-beta en el algoritmo minimax implementado, como *SSS** [12, 15].

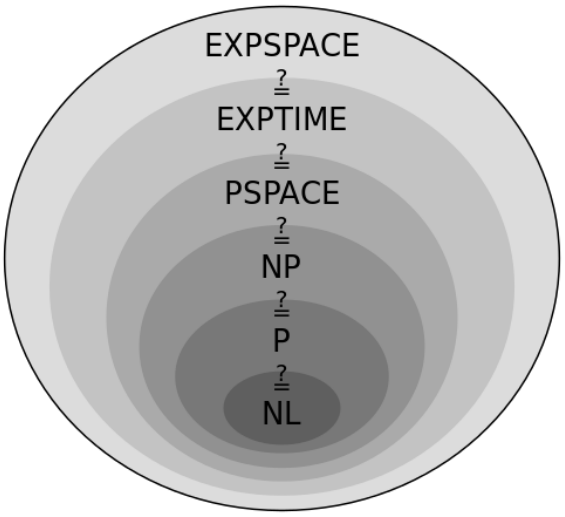


Figura 7.1: Representación de la relación entre clases de complejidad (NL, P, NP, PSPACE, EXPTIME, EXPSPACE) [13]

Chapter 8

Conclusions and Future Work

This Final Degree Project has provided a solid foundation for the analysis of business expansions from a theoretical and practical perspective and has fomented the author's academic development in the area of computational complexity.

In this study we have proven that the business expansion problem is PSPACE-complete by first using a minimax algorithm to prove its membership in PSPACE and then polynomially reducing the PSPACE-complete problem TQBF to it.

We have implemented a practical tool to analyze and solve instances of the business expansion problem. Due to the intractability of the studied problem, we have used heuristic methods (specifically, a minimax algorithm with alpha-beta pruning) to be able to solve instances of significant sizes in a reasonable time.

We have executed with this tool a large quantity of artificially generated instances of both the expansion problem and, after a transformation, TQBF. These executions have helped us identify the impact of the choice of input parameters on the time and precision of the tool. Additionally, we have used this tool to conduct our case study, using a simplified version of a business expansion problem.

While working on the case study, we have realized the difficulty of translating a real case of business expansion to an instance of our problem. For example, in a real case the gain value of each building is unknown and making a future projection is usually a very complex process. The relations in our problem are additional gains that arise from having made certain synergistic edifications due to, for instance, creating monopoly situations and measuring them is even more complex. Additionally, in a real situation it is very uncommon to have total information about the opponent. Because of this, we have had to use fictitious but relatively reasonable data.

Due to these difficulties, our problem is insufficient to address most real cases of business expansions. Despite this, we hope that this case study has served to highlight the importance of having good computational tools capable of handling intractable problems like ours in business decision making and that it will prove useful and interesting to other researchers.

Originally, our business expansion game had different termination conditions. Instead of ending when any player reached a certain amount of money or T was

empty, the game ended when the *first* player reached a certain amount of money or T was empty. This is more true to what truly concerns companies about an expansion; it is not important who reaches the target amount of money first, knowing with certainty that it is achievable is enough. We had done most of the complexity analysis with this alternative version of the problem and it seemed like it was also going to be PSPACE-complete, but this seemingly small detail made the demonstration considerably longer without adding virtually nothing of interest to it. Therefore, we decided to use the other version of the problem.

Regarding the possible future work, an option would be to finish exploring this alternative version of the problem, or some generalization of our problem that is closer to reality. Trivially, any generalization of the problem will have at least the same difficulty as the original problem (PSPACE), but it could have a higher difficulty (like EXPTIME or EXPSPACE), making its study more interesting. Another way to continue advancing with this work is to study the use of alternative heuristic methods to the alpha-beta pruning in the implemented minimax algorithm, such as *SSS** [12, 15].

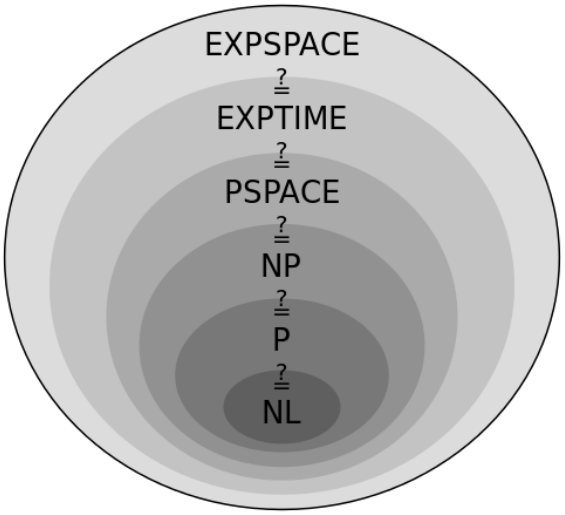


Figure 8.1: Representation of the relationship between complexity classes (NL, P, NP, PSPACE, EXPTIME, EXPSPACE) [13]

Bibliografía

[1] Alfonso. Valor de empresa (enterprise value). 2016. Disponible en `https://economipedia.com/definiciones/valor-empresa-enterprise-value.html` (último acceso, september, 2023).

[2] Arias, A. S. EBITDA. 2014. Disponible en `https://economipedia.com/definiciones/ebitda.html` (último acceso, september, 2023).

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

[4] CFI Team. EBITDA multiple. 2023. Disponible en `https://corporatefinanceinstitute.com/resources/valuation/ebitda-multiple/` (último acceso, september, 2023).

[5] CFI Team. Enterprise value (EV). 2023. Disponible en `https://corporatefinanceinstitute.com/resources/valuation/what-is-enterprise-value-ev/` (último acceso, september, 2023).

[6] Fernández González, V. Complejidad y resolución práctica de razonamientos espacio-temporales para criminología. 2022.

[7] Galiana Ruiz de la Hermosa, J. Cómo frenar una propagación: Análisis de complejidad y resolución. 2021.

[8] Horowitz, E., Sahni, S. y Rajasekaran, S. *Computer algorithms C++: C++ and pseudocode versions*. Macmillan, 1997.

[9] Mascarello, C. EBITDA multiples by industry. 2023. Disponible en `https://www.equidam.com/ebitda-multiples-trbc-industries/` (último acceso, september, 2023).

[10] Neapolitan, R. y Naimipour, K. *Foundations of Algorithms Using C++ Pseudocode*. Algorithms Series. Jones and Bartlett, 2004. ISBN 9780763723873.

[11] Pantoja, D. Resolución práctica de un problema de expansión empresarial. `https://github.com/davidpantojasanchez/TFG.git`, 2023.

[12] Plaat, A., Schaeffer, J., Pijls, W. y de Bruin, A. A minimax algorithm better than SSS*. *Artificial Intelligence*, vol. 87(1-2), páginas 255–293, 1996.

[13] Qef. Complexity subsets PSPACE. 2008. Disponible en `https://commons.wikimedia.org/wiki/File:Complexity_subsets_pspace.svg` (último acceso, september, 2023).

[14] Rodríguez, I., Rabanal, P. y Rubio, F. How to make a best-seller: Optimal product design problems. *Applied Soft Computing*, vol. 55, páginas 178–196, 2017.

[15] Stockman, G. C. A minimax algorithm better than alpha-beta? *Artificial Intelligence*, vol. 12(2), páginas 179–196, 1979.