

UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
Departamento de Ingeniería Informática



**Búsqueda local iterada para optimizar el tiempo en órdenes de
construcción en Starcraft II: experiencia Protoss**

Andrés Antonio Alcaíno Castro

Profesor guía: Manuel Villalobos Cid

Tesis para optar al título de Ingeniero de
Ejecución en Computación e Informática

Santiago – Chile

2021

RESUMEN

Los videojuegos de estrategia en tiempo real (RTS por sus siglas en inglés) son ampliamente conocidos y jugados en todo el mundo. La saga de Starcraft es uno de los principales exponentes, y sus juegos son de los que más campo de investigación ha generado desde su lanzamiento en 1998. En este Informe, se plantea un enfoque metaheurístico para intentar solucionar un problema conocido de este juego, los órdenes de construcción. En cada partida de este juego se empieza con una secuencia de construcción que generalmente usan jugadores profesionales, estas secuencias preconcebidas permiten entrenar unidades y adquirir tecnologías con mayor rapidez y frecuencia de lo normal. Encontrar estas secuencias y/u optimizar las más conocidas, permite obtener una ventaja en términos de tiempo y de cantidad de unidades o tecnologías con respecto al jugador oponente. Los resultados de este proyecto indican que es posible optimizar el tiempo en órdenes de construcción utilizando la metaheurística de búsqueda local iterada.

Palabras Claves: metaheurística, búsqueda local iterada, estrategia en tiempo real, videojuegos, orden de construcción

Si crees que algo es imposible, repítete a ti mismo que es posible, hasta que lo sea.

AGRADECIMIENTOS

Empezar agradeciendo a mi familia, mis padres, mi hermano, mi primo, a mis tíos y tíos, a mis abuelos y sobre todo a mi prometida. Gracias por estar a mi lado en todo este proceso. De no haber sido por todo el apoyo y paciencia que tuvieron conmigo probablemente hoy no estaría dando uno de los pasos mas importantes de mi vida. Muchas gracias por darme tantos consejos y por alentarme a seguir adelante. Jamás lo olvidaré.

Agradecer también a mi profesor guía, Manuel Villalobos, por enseñarme tantas cosas en tan poco tiempo, por otorgarme tantas horas fuera de su jornada de trabajo para ver el estado del proyecto, por estar siempre dispuesto a ayudar. Trabajar a su lado es una experiencia realmente enriquecedora.

Además, agradecer a mis compañeros Edson Reyes y Brandon Núñez con quienes tuve el privilegio de trabajar codo a codo en nuestros proyectos de titulación y por estar siempre dispuestos a resolver mis dudas.

Finalmente, quiero agradecer a mis mas preciados amigos de la universidad, con quienes viví aventuras realmente increíbles, compartimos momentos sumamente divertidos y sufrimos juntos las dificultades que nos presentó la vida universitaria. Gracias por apoyarme y por ayudarme cuando más lo necesitaba.

TABLA DE CONTENIDO

1	Introducción	1
1.1	Antecedentes y motivación	1
1.2	Descripción del problema	3
1.3	Solución Propuesta	3
1.4	Objetivos y alcance del proyecto	4
1.4.1	Objetivo general	4
1.4.2	Objetivos específicos	5
1.4.3	Alcances	5
1.5	Metodología y herramientas	5
1.5.1	Herramientas de desarrollo	6
1.6	Estructura del Documento	7
2	Marco Teórico	9
2.1	Starcraft II	9
2.1.1	Árbol de tecnologías	9
2.1.2	Orden de construcción	9
2.1.3	Velocidad de partida	10
2.1.4	Economía del juego	11
2.1.5	Habilidades	12
2.1.6	Unidades especiales	12
2.2	Starcraft II como problema de optimización	13
2.2.1	Resource-Constrained Project Scheduling Problem	13
2.3	Metaheurísticas	14
2.4	Irace	15
3	Estado del arte	17
4	Algoritmo propuesto y diseño experimental	20
4.1	Algoritmo propuesto	20
4.1.1	Búsqueda local iterada	20
4.1.2	Representación de datos	22
4.1.3	Operadores	25
4.1.4	Criterio de optimización	28
4.2	Diseño experimental	29
4.2.1	Parametrización	29
4.2.2	Evaluación	29
5	Resultados experimentales	31
5.1	Parametrización	31
5.2	Evaluación	31
5.2.1	Resultados de las pruebas	31
5.2.2	Convergencia del algoritmo	35
5.2.3	Comparación con órdenes de construcción conocidos	39
5.2.4	Aplicación en juego	41
6	Análisis de Resultados	42
6.1	Rendimiento en pruebas	42
6.1.1	Prueba n° 1: Obtener 5 Zealots	42
6.1.2	Prueba n° 2: Obtener 5 Dark Templars	43
6.1.3	Prueba n° 3: Obtener 5 Phoenix	43
6.2	Rendimiento contra órdenes de construcción conocidos	44

6.2.1	Prueba n° 1: Obtener 5 Zealots	44
6.2.2	Prueba n° 2: Obtener 5 Dark Templars	44
6.2.3	Prueba n° 3: Obtener 5 Phoenix	44
6.3	Aplicación en el juego	45
6.3.1	Dificultades	45
7	Conclusiones	47
7.1	Trabajo a futuro	48
Glosario		50
Referencias bibliográficas		52

ÍNDICE DE TABLAS

Tabla 1.1	Especificaciones de Hardware para ambiente Ubuntu	7
Tabla 1.2	Especificaciones de Hardware para ambiente MacOS	7
Tabla 2.1	Velocidad de juego	11
Tabla 3.1	Resumen estado del arte	19
Tabla 4.1	Orden de construcción inicial	27
Tabla 4.2	Orden de construcción perturbado	27
Tabla 4.3	Parámetros a evaluar utilizando Irace y el rango entregado	29
Tabla 5.1	Parametrización entregada por Irace	31
Tabla 5.2	Resultados entregados por la metaheurística para obtener 5 Zealots	32
Tabla 5.3	Resultados entregados por el algoritmo goloso para obtener 5 Zealots	33
Tabla 5.4	Resultados entregados por la metaheurística para obtener 5 Dark Templars . .	33
Tabla 5.5	Resultados obtenidos por el algoritmo goloso para obtener 5 Dark Templars .	34
Tabla 5.6	Resultados entregados por la metaheurística para obtener 5 Phoenix	34
Tabla 5.7	Resultados obtenidos por el algoritmo goloso para obtener 5 Phoenix	35
Tabla 5.8	Resultados orden de construcción conocido para obtener Zealots	40
Tabla 5.9	Resultados orden de construcción sugerido para obtener Zealots	40
Tabla 5.10	Resultados orden de construcción conocido para obtener Dark Templars . .	40
Tabla 5.11	Resultados orden de construcción sugerido para obtener Dark Templars . .	40
Tabla 5.12	Resultados orden de construcción sugerido para obtener Phoenix	41
Tabla 5.13	Resultados orden de construcción sugerido para obtener Phoenix	41
Tabla 5.14	Resultados orden de construcción sugerido para obtener Zealots	41
Tabla 5.15	Resultados orden de construcción sugerido para obtener Dark Templars . .	41
Tabla 5.16	Resultados orden de construcción sugerido para obtener Phoenix	41

ÍNDICE DE FIGURAS

Figura 1.1	Dinero entregado cada año en concepto de premios.	2
Figura 1.2	Representación visual del algoritmo de búsqueda local iterada	4
Figura 2.1	Orden de construcción de la raza protoss extraído del juego.	10
Figura 2.2	Partida de Starcraft 2 con algunos de los elementos del juego	13
Figura 2.3	Esquema de los componentes requeridos por Irace	16
Figura 3.1	Documentos por año sobre el tema "Starcraft" en Scopus.	17
Figura 4.1	Implementación del árbol de tecnologías.	23
Figura 5.1	Convergencia algoritmo goloso en prueba n°1	36
Figura 5.2	Convergencia algoritmo goloso en prueba n°2	36
Figura 5.3	Convergencia algoritmo goloso en prueba n°3	37
Figura 5.4	Convergencia algoritmo de búsqueda local iterada en prueba n°1	38
Figura 5.5	Convergencia algoritmo de búsqueda local iterada en prueba n°2	38
Figura 5.6	Convergencia algoritmo de búsqueda local iterada en prueba n°3	39

CAPÍTULO 1. INTRODUCCIÓN

1.1 ANTECEDENTES Y MOTIVACIÓN

Los juegos de estrategia en tiempo real (RTS por sus cifras en inglés) se han popularizado bastante las últimas décadas, siendo la saga Starcraft una de las más conocidas y queridas por la comunidad. En Starcraft 2 (la secuela de Starcraft), el jugador puede construir distintas unidades y edificios siguiendo un árbol de tecnologías, de modo que para desbloquear una unidad es necesario construir un edificio que la desbloquea o cumplir con una precondición. De esta forma, el jugador crea un ejército para derrotar a sus rivales. También es necesario recalcar que para crear unidades o construir edificios es necesario tener los recursos para hacerlo (tiempo, minerales, gas vespeno, constructores).

Puede compararse con un tablero de ajedrez donde existen miles de millones de combinaciones de movimientos, pero a su vez más complejo y con múltiples factores a considerar, como por ejemplo el tiempo de construcción, la velocidad de movimiento de las unidades, el terreno del mapa, el ejercito enemigo y el hecho de que no es posible observar los movimientos que hace el rival ya que existe lo que se conoce como “niebla” y solo se puede ver lo que hay en el mapa cuando el jugador tiene una unidad o edificio en ese lugar.

Ahora bien, existen 3 razas distintas con las que se puede jugar: Terran, Zerg y Protoss. Es necesario hacer énfasis en las diferencias entre estas 3 razas, por ejemplo, los Terran pueden generar unidades que saltan colinas; los Zerg sacrifican a los constructores para construir edificios; los Protoss pueden generar unidades en cualquier lugar del mapa teniendo un edificio llamado “Pilón” cerca. Son razas increíblemente complejas y distintas una de la otra en términos de jugabilidad.

Por otro lado, los jugadores pueden seguir un orden de construcción para obtener una unidad determinada en menor tiempo o simplemente para molestar al rival. Este orden de construcción no es más que una secuencia de acciones que debe realizar el jugador y le indica que unidad entrenar, así como que edificio construir en un determinado momento. Nuevamente y, haciendo una analogía con el ajedrez, se puede comparar un orden de construcción con una apertura del ajedrez, como por ejemplo la apertura siciliana, la cuál es ampliamente conocida y utilizada por ajedrecistas a nivel mundial.

Al ser Starcraft un juego con una gran comunidad de jugadores y fanáticos es que se realizan torneos internacionales masivos permitiendo a jugadores profesionales ganar muchísimo dinero. De hecho, según Garro (2018) en el año 2018 se repartieron 4.53 millones de dólares en concepto de premios. También se puede apreciar un incremento desde el año 2005 en la cantidad

de dinero que se entrega a los ganadores de estos eventos como se puede apreciar en la Figura 1.1.

Chart 1: Major Event Prize Money

Split Per Region | Global | 2016

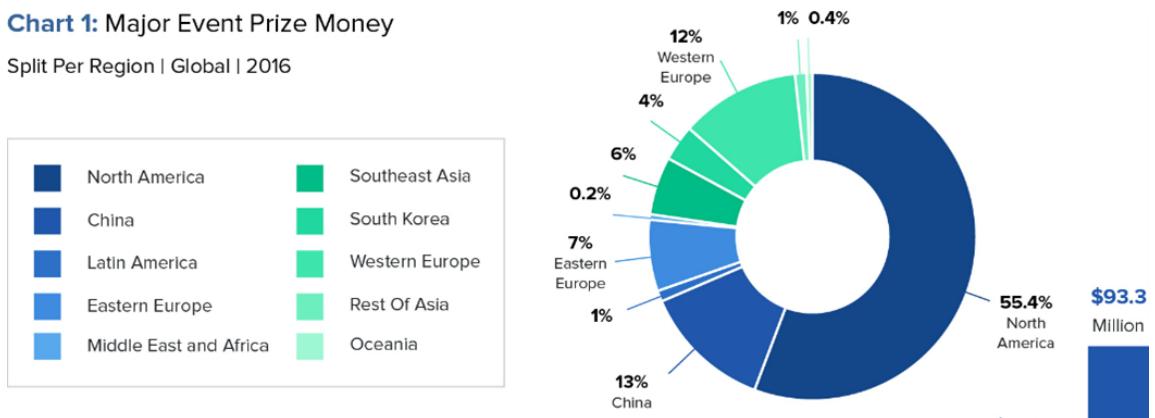
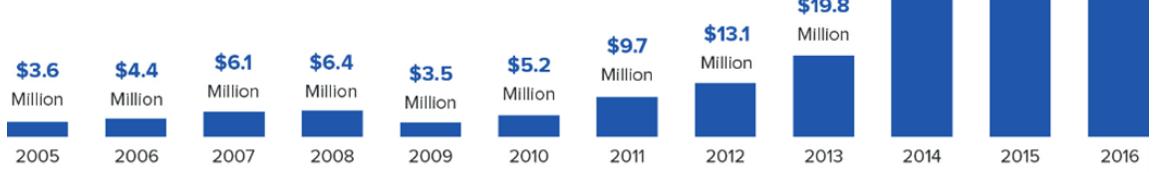


Chart 2: Prize Money Development

Global | 2005-2016



Source: Newzoo 2017 Report



Figura 1.1: Dinero entregado cada año en concepto de premios.

Fuente: Chapman (2017).

Y no solo es utilizado por jugadores, Kuo (2012) expresa que también es utilizado como herramienta de educación para enseñar matemáticas, economía y teoría del juego, e incluso en sistemas de entrenamiento militar como indica Manu Sharma (2007), debido a que el juego se trata justamente de estrategias militares y gestión de recursos, ya sea recursos económicos como minerales y gas vespeno, o recursos bélicos como unidades de infantería y de asedio.

Resolver los desafíos que presenta este juego influye no sólo en la comunidad de jugadores, sino que también en el campo del desarrollo de videojuegos, en la educación y, como se mencionó anteriormente, en el ámbito militar. Y justamente uno de los problemas más comunes es dar con órdenes de construcción eficientes que permitan aumentar las probabilidades de victoria de los jugadores al adquirir rápidamente sus tecnologías y unidades del juego.

1.2 DESCRIPCIÓN DEL PROBLEMA

Con respecto al juego, ¿cómo se pueden obtener buenos órdenes de construcción con la raza Protoss, que permitan hacer más eficiente el uso del tiempo y cantidad de unidades, y subsecuentemente obtener una ventaja con respecto a los jugadores oponentes? Siendo un orden de construcción los pasos a seguir que debe realizar el jugador para llegar a construir un cierto edificio o unidad. Pero, ¿por qué buenos órdenes de construcción y no óptimos? Para responder esta pregunta primero se debe entender lo siguiente: este problema se puede tratar como un problema RCPSP (Problema de programación de proyecto con recursos limitados, por sus cifras en inglés) y este a su vez según Viglietta (2012) tiene una complejidad de NP-Hard, esto es, no existen algoritmos de tiempo polinomial que puedan resolver estos problemas actualmente, pero si metaheurísticas que permiten encontrar buenos resultados, aunque no sean los óptimos, Suárez (2011).

Ahora bien, las tres razas del videojuego (Zerg, Terran y Protoss) son totalmente distintas una de la otra, con habilidades y factores operacionales de jugabilidad exclusivas. Es por esto que surge la necesidad de resolverlo para cada raza por separado.

Por lo tanto, y para formalizar: el problema es encontrar buenos órdenes de construcción en términos de tiempo y cantidad de unidades/tecnologías, o bien optimizar lo más posible los ya existentes, con la raza Protoss en el videojuego Starcraft 2.

1.3 SOLUCIÓN PROPUESTA

Ahora bien, como solo se requiere una única solución (un orden de construcción) se propone implementar lo que se conoce como una S-metaheurística, o metaheurística basada en una solución única (Talbi, 2009). Para efectos de este proyecto se implementará la metaheurística de Búsqueda Local Iterada (ILS, por sus cifras en inglés) ya que ha demostrado ser un buen método para encontrar soluciones satisfactorias en este tipo de problemas, así lo afirman Ballestín & Trautmann (2008). Este algoritmo obtiene una primera solución a través de un algoritmo de búsqueda local, a la cual se le añade una perturbación y luego vuelve a realizar la búsqueda local. El proceso se repite hasta encontrar un punto cercano al óptimo. Esto se entiende mejor observando la Figura 1.2. El algoritmo será implementado basándose en el libro *Metaheuristics: from design to implementation* (Talbi, 2009).

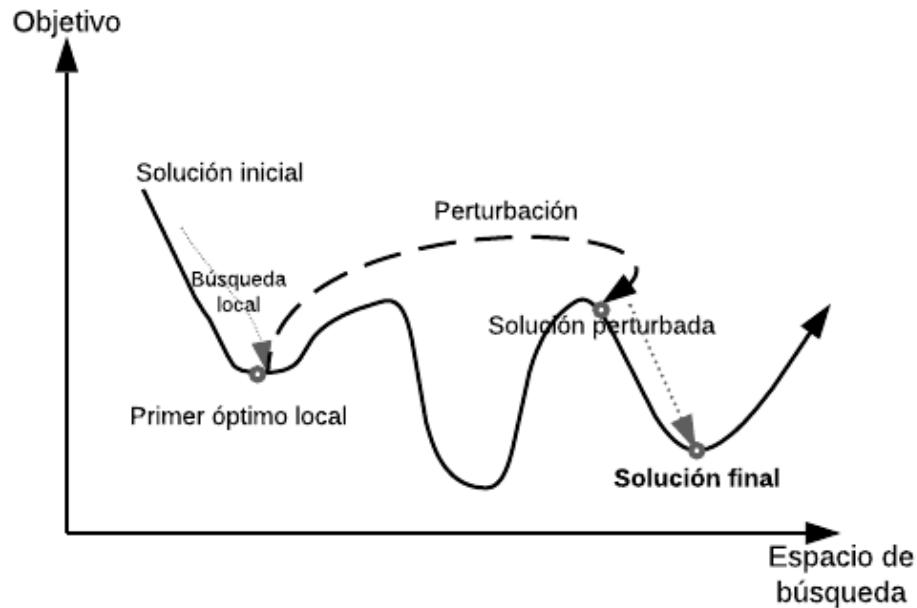


Figura 1.2: Representación visual del algoritmo de búsqueda local iterada

Fuente: Talbi (2009).

Este modelo tomará en consideración el árbol de tecnologías, el tiempo de construcción/entrenamiento/investigación de cada uno de los elementos del árbol de tecnologías, las habilidades especiales que influyen en el orden de construcción de la raza Protoss, que son únicas con respecto a las otras razas, y el consumo de recursos de cada elemento del árbol.

1.4 OBJETIVOS Y ALCANCE DEL PROYECTO

1.4.1 Objetivo general

Desarrollar un modelo basado en la metaheurística de búsqueda local iterada para generar órdenes de construcción optimizando el tiempo en el videojuego Starcraft 2 y utilizando la raza Protoss. Esto para obtener una ventaja frente a un jugador oponente, la cual se puede medir a partir de la cantidad y tipo de unidades, cantidad de tecnologías y, sobre todo, el tiempo en que se demoró el jugador en conseguirlas.

1.4.2 Objetivos específicos

- **OE1:** Definir la estructura de datos que se utilizará para los órdenes de construcción y para el árbol de tecnologías.
- **OE2:** Modelar el problema integrando búsqueda local iterada para ser utilizada con los parámetros de la raza Protoss (Árbol de tecnologías, unidades, tiempos de entrenamiento, costes de recursos y habilidades únicas de la raza).
- **OE3:** Desarrollar un sistema de software que ejecute el algoritmo.
- **OE4:** Obtener resultados y realizar comparaciones con órdenes de construcción conocidos por la comunidad para verificar que el algoritmo obtiene buenas soluciones.

1.4.3 Alcances

La solución tendrá como alcances la implementación de un algoritmo metaheurístico de búsqueda local iterada que pueda encontrar buenos órdenes de construcción y un programa o sistema de software que permita ejecutar el algoritmo de búsqueda local iterada.

Sin embargo, al ser Starcraft un juego de alta complejidad se tendrán que dejar de lado ciertos aspectos de la jugabilidad como por ejemplo la variabilidad de los mapas, la velocidad de movimiento de las unidades, los atributos de daño, defensa, escudos y vida de las unidades y edificios, las habilidades especiales de ciertas unidades que no influyen en el orden de construcción y los ataques de tropas enemigas. También se simplificará la recolección de recursos ya que estos dependen de la cantidad de trabajadores recolectando y no hay forma precisa de saber ese dato ya que los jugadores están constantemente moviendo sus trabajadores.

1.5 METODOLOGÍA Y HERRAMIENTAS

El proyecto contempla tanto desarrollo como investigación, por lo que será necesario utilizar una metodología por etapas, utilizando el método científico como base (C. León (2015)) para la parte de investigación. El problema ya se encuentra definido y se usará la siguiente pregunta de hipótesis: ¿Puede la metaheurística de solución única búsqueda local iterada encontrar buenas órdenes de construcción en términos de tiempo en el juego Starcraft II con la raza Protoss?

Por otro lado, se utilizará la metodología Kanban para el desarrollo del software, debido a que sólo una persona se dedicará a desarrollar el programa.

- **Etapa 1: Definición:** Esta etapa está relacionada con el objetivo específico 1, y consiste en definir la estructura de datos que se utilizará para los órdenes de construcción y para el árbol de tecnologías. Generalmente se utilizan matrices, para los órdenes de construcción, como es el caso de (Takino & Hoki (2015)), pero es algo que tendrá que ser evaluado en esta etapa. También se evaluará la implementación de grafos para modelar el árbol de tecnologías.
- **Etapa 2: Modelamiento:** Ligada al objetivo específico 2, esta etapa consiste en el modelamiento del problema aplicando las restricciones de jugabilidad de la raza Protoss, utilizando las estructuras de datos definidas en la etapa 1 e integrando el algoritmo de búsqueda local iterada. Se modelará el algoritmo utilizando el libro Metaheuristics: From Design to Implementation (Talbi, 2009) como referencia.
- **Etapa 3: Desarrollo de software:** Esta etapa está ligada al objetivo específico 3 y consiste en la integración de los modelos teóricos obtenidos de las etapas 1 y 2 en un sistema de software. Luego se implementará una interfaz gráfica simple para poder ingresar las entradas del programa de forma más sencilla. Y finalmente, se parametrizan los resultados obtenidos.
- **Evaluación de resultados:** La última etapa está alineada con el objetivo específico 4. En esta, se evaluará el desempeño del algoritmo en términos de convergencia y precisión. Luego, se evaluarán los resultados obtenidos a partir de la comparación entre las soluciones entregadas por el algoritmo y los órdenes de construcción conocidos. Se realizará un test estadístico para confirmar que la metaheurística obtiene mejores resultados que un algoritmo de búsqueda local.

1.5.1 Herramientas de desarrollo

Para el desarrollo de la solución se utilizarán las siguientes herramientas de trabajo:

- **Python** versión 3.8.8 o superior.
- **Visual Studio Code** versión 1.52.1 o superior.
- **Github y Git** versión 2.29.2 o superior.

- **Overleaf**, aplicación web para la escritura de documentos utilizando Latex.

En las Tablas 1.1 y 1.2 se indican las especificaciones de los ambientes de desarrollo en los cuáles se llevó a cabo el proyecto.

Tabla 1.1: Especificaciones de Hardware para ambiente Ubuntu

Sistema Operativo	Ubuntu 20.04 LTS
Procesador	AMD Ryzen 7 2700 (8 núcleos, 16 hilos) a 3.8GHz
Memoria Ram	2 x 8GB DDR4 Kingston Fury a 2666MHz
GPU Dedicada	MSI NVIDIA GTX 1650s de 4GB de VRAM

Fuente: Elaboración propia, 2021

Tabla 1.2: Especificaciones de Hardware para ambiente MacOS

Modelo	MacBook Pro (13", M1, 2020)
Sistema Operativo	MacOS Big sur 11.5.2
Procesador	Apple M1 8 núcleos
Memoria Ram	8 GB
GPU Integrada	Apple M1 8 núcleos

Fuente: Elaboración propia, 2021

1.6 ESTRUCTURA DEL DOCUMENTO

Con respecto al formato del presente documento, sin contar el capítulo actual que corresponde a la introducción, se compone principalmente de cinco capítulos que abarcan el desarrollo del proyecto. En el capítulo 2 se tiene el marco teórico en el cual descansa el proyecto, se definen los principales conceptos del juego Starcraft II como el árbol de tecnologías y el orden de construcción, así como también conceptos técnicos como las metaheurísticas. Luego en el capítulo 3 se tiene el estado del arte del problema, en donde se encuentran los principales proyectos e investigaciones relacionadas al problema en cuestión. En el capítulo 4 se tiene la presentación del algoritmo propuesto, en donde se detalla el diseño a implementar del algoritmo, las estructuras de datos que utilizará, los operadores y el criterio de optimización, y también se detalla la forma de evaluar los resultados del algoritmo. En el capítulo 5 se presentan los

resultados obtenidos mediante la experimentación. En el capítulo 6 se evalúan los resultados mediante el análisis de estos. Y finalmente en el capítulo 7 se presenta el trabajo pendiente y futuro, y las conclusiones de la memoria evaluando si se cumplieron los objetivos y haciendo un resumen del proyecto en general.

CAPÍTULO 2. MARCO TEÓRICO

Para tener una mejor comprensión del tema expuesto es que se procede a detallar los términos y conceptos más importantes referentes al juego en el que se basa el proyecto y a la metaheurística a implementar. Además, se incluye el estado del arte en la actualidad para que el lector conozca algunos de los avances e investigaciones en el tema.

2.1 STARCRAFT II

2.1.1 Árbol de tecnologías

El árbol de tecnologías es un diagrama que expone todas las entidades que se pueden construir o investigar con una raza determinada en el juego. En este caso, la raza *Protoss*, que es en la que se enfoca la presente memoria, cuenta con 60 entidades: 14 edificios, 19 unidades y 27 tecnologías. Cada entidad tiene una serie de requisitos para poder obtenerla, no se puede construir un edificio sin haber construidos aquellos que vienen antes del mismo y tampoco se puede construir si no se cuenta con los recursos necesarios. Un diagrama del árbol de tecnologías puede ser encontrado en la sección 4.1.2, figura 4.1.

2.1.2 Orden de construcción

Un orden de construcción es una recopilación de instrucciones que ejecuta el usuario en un determinado momento. Se compone principalmente por el tiempo en que se ejecuta la acción y la entidad que se ordena construir o investigar. Para efectos de este proyecto, un orden de construcción contempla el tiempo, la entidad, los recursos en ese momento, y si se utilizó o no la habilidad *Chrono boost* que permite acelerar el proceso de la construcción por 20 segundos. En la Figura 2.2 se puede apreciar un orden de construcción extraído del juego.

Tiempo	Acción	Suministros
1:04	 Sonda	17 / 23
1:14	 Sonda	18 / 23
1:14	 Asimilador	18 / 23
1:22	 Sonda	19 / 23
1:32	 Sonda	20 / 23
1:33	 Pilón	20 / 23
1:45	 Sonda	21 / 23
1:48	 Zelot	23 / 23
1:57	 Sonda	24 / 31
1:57	 Pilón	24 / 31

Figura 2.1: Orden de construcción de la raza protoss extraído del juego.

Fuente: Elaboración propia (2021).

2.1.3 Velocidad de partida

Los jugadores pueden definir la velocidad de la partida. Existen 5 modalidades: *slower*, *slow*, *normal*, *fast* y *faster*. Cada modalidad modifica la cantidad de segundos de juego que pasan en un minuto de tiempo de la modalidad normal. Por ejemplo, en la modalidad *faster* se necesitan 42 segundos para completar 1 minuto en tiempo normal. El proyecto considera la velocidad *faster* que es la que se utiliza en partidas normales Jugador contra Jugador y en partidas clasificatorias. En la tabla 2.1 se presentan las conversiones de segundos y los factores de velocidad por modalidad.

Tabla 2.1: Velocidad de juego

Velocidad de juego	Factor velocidad	Segundos en tiempo real en 1 minuto de tiempo normal
Slower	0.6	100
Slow	0.8	75
Normal	1	60
Fast	1.2	50
Faster	1.4	42.86

Fuente: Liquipedia (2021a)

2.1.4 Economía del juego

Un aspecto importante que se debe considerar es la economía presente en el juego. El jugador debe recolectar y administrar sus recursos en orden de poder construir mas unidades, edificios y tecnologías. El jugador que tenga una mejor economía tiene mayores probabilidades de ganarle a su rival, ya que tiene mayor potencial de generar ejércitos y/o defensas en sus ciudades. De hecho, una de las estrategias mas comunes es destruir a los trabajadores del rival y así disminuir sus ingresos de recursos.

Existen 3 tipos de recursos:

- **Minerales:** Los minerales son unos cristales que se encuentran en distintos puntos del mapa de juego. Los jugadores deberán construir un puesto de recolección (o *Nexus* en el caso de la raza *Protoss*) lo más cercano posible para que sus trabajadores no tengan que perder tiempo recorriendo grandes distancias con los recursos, exponiéndose así a los peligros y amenazas del rival. Los trabajadores pueden llevar hasta 5 minerales a la vez y deben llevárselos lo antes posible al puesto de recolección. La forma óptima de recolectar minerales es teniendo a máximo 3 trabajadores por campo de mineral, un cuarto trabajador no generaría ningún ingreso adicional. La tasa de recolección cuando el campo de minerales está saturado con 3 trabajadores es de 143 minerales por minuto.
- **Gas vespertino:** El gas vespertino es un recurso que se obtiene mediante la construcción de un edificio llamado *Assimilator* en uno de los géiseres de gas vespertino. Soporta hasta 3 trabajadores por edificio y por lo general se pueden construir dos por cada punto de recolección. La tasa de recolección cuando el edificio está saturado con 3 trabajadores es de 163 gas por minuto.
- **Suministros:** Cada entidad de tipo unidad utiliza una cierta cantidad de suministros para poder ser construida. Los *Probes* utilizan 1 de suministros, mientras que la unidad *Mothership* que es una de las mas fuertes del juego utiliza 8 de suministros. Para obtener

suministros es necesario construir unos edificios, en el caso de los *Protoss* se puede obtener mediante la construcción de un *Nexus* que otorga 15 de suministros o bien un *Pylon* que otorga 5 de suministros. Se puede aumentar el cupo de suministros hasta llegar a un total de 200.

- **Energía:** Ciertas entidades del juego poseen un atributo de energía para poder utilizar sus habilidades. Por ejemplo, la unidad *High Templar* tiene una habilidad llamada *Psionic Storm* que permite al usuario seleccionar un área del mapa para generar daño a los jugadores enemigos por 2.85 segundos. No es necesario utilizar trabajadores para recolectar energía ya que esta se regenera cada segundo, con una tasa de 0.7875 en la velocidad *Faster*.

2.1.5 Habilidades

Hay ciertas habilidades que permiten mejorar la tasa de construcción de un edificio. En el caso de los *Protoss* se puede encontrar la habilidad *Chrono boost* que se puede ejecutar en un *Nexus* gastando 50 puntos de energía de este. Esta habilidad aumenta la velocidad de construcción o investigación de un edificio seleccionado por el jugador en un 50%.

También existe una tecnología *Protoss* llamada *Warp gate* que, si se es investigada por el jugador, permite construir unidades en cualquier parte del mapa que cuente con el edificio *Pylon* cerca y en un tiempo mucho menor que construyéndolo de la forma tradicional. La desventaja de esto es que una vez construida la unidad se debe esperar un tiempo de enfriamiento antes de volver a utilizar el edificio.

2.1.6 Unidades especiales

Para cada raza hay ciertas unidades que no se construyen de manera convencional, esto es, en un edificio. Para el caso de los *Protoss* tenemos la unidad conocida como *Archon* que se construye a partir de 2 unidades de *Dark Templar* o 2 unidades de *High Templar*. Las unidades se sacrifican para generar una única unidad de *Archon* que es mucho mas fuerte y tiene otras habilidades. Esta unidad se demora 8,57 segundos en generarse en velocidad *faster*.

En la Figura 2.2 se presentan algunos de los elementos mencionados en esta sección.



Figura 2.2: Partida de Starcraft 2 con algunos de los elementos del juego

Fuente: Elaboración Propia (2021)

2.2 STARCRAFT II COMO PROBLEMA DE OPTIMIZACIÓN

Como quedó expuesto en la sección anterior, Starcraft II puede ser modelado mediante grafos y listas. Por un lado se tiene al árbol de tecnologías, el cuál contiene nodos con requisitos de precedencia, y por otro lado se tienen los órdenes de construcción, los cuales son listas que indican acciones en un tiempo determinado y que se pueden realizar siempre y cuando cumplan con los requisitos de recursos de las entidades. A continuación se presenta una forma de ver este problema.

2.2.1 Resource-Constrained Project Scheduling Problem

Según Artigues et al. (2007) el problema de programación de proyectos con recursos limitados (*RCPSP* por sus siglas en inglés) es un problema en el cual se busca encontrar la programación mas cercana a la programación óptima de un proyecto. Este considera recursos de disponibilidad limitada, relaciones de precedencia y también peticiones de acciones en tiempos determinados .

RCPSP está definido como un problema optimización combinatorial, en donde existe un espacio solución X el cuál es discreto o puede ser reducido a un espacio discreto y un subset de soluciones factibles $Y \subseteq X$ asociado a una función objetivo $f : Y \rightarrow R$. En esta clase de problemas se busca encontrar una solución factible $y \in Y$ tal que $f(y)$ está minimizada o maximizada.

El proyecto está representado por el conjunto de actividades $V = [A_0, \dots, A_{n+1}]$, en donde A representa una actividad, por lo tanto A_0 es el inicio del proyecto y A_{n+1} es el final del proyecto. También se considera la precedencia de las actividades, a este conjunto de precedencias se le denomina E . Cada precedencia viene dada por un par de nodos tal que $(A_i, A_j) \in E$. En donde A_i precede al nodo A_j . El conjunto de actividades, las cuales poseen precedencia, pueden ser colocadas en el grafo $G(V, E)$. Finalmente, los recursos limitados están expresados por $R = [r_0, \dots, r_n]$ donde r_i es el recurso.

El problema *RCPSP* está catalogado como *NP-Hard*, razón por la cual se decidió trabajar con metaheurísticas en este proyecto. En las secciones posteriores se detallará la metaheurística utilizada, así como la forma en que se modeló el árbol de tecnologías y el orden de construcción, las cuales corresponden al grafo y al conjunto de actividades en el problema *RCPSP*.

2.3 METAHEURÍSTICAS

Las metaheurísticas son algoritmos que permiten abordar problemas de alta complejidad mediante la obtención de soluciones satisfactorias en un tiempo razonable. Sin embargo, no existe la certeza de que dichas soluciones sean óptimos globales o incluso locales. Son bastante utilizadas para resolver problemas de optimización, por ejemplo en el área de la dinámica de fluidos, telecomunicaciones, robótica, topología, modelamiento de sistemas, problemas de programación de proyectos y más (Talbi, 2007).

Existen distintos tipos de metaheurísticas, entre las cuales se encuentran:

- **Metaheurísticas de solución única:** También conocidas como *S-metaheurísticas* aplican iterativamente una perturbación a la solución única actual. En la fase de generación, un set de soluciones candidatas son generadas desde la solución inicial. Luego se selecciona un candidato del set de soluciones y se utiliza para reemplazar la solución actual. Este proceso continua hasta que un criterio de detención determine lo contrario.
- **Metaheurísticas basadas en población:** Conocidas como *P-metaheurísticas* empiezan desde una población inicial de soluciones. Luego se aplica de forma iterativa la creación

de una nueva población y el reemplazo de la población actual. En la fase de generación una nueva población de soluciones es creada. En la fase de reemplazo se selecciona la población actual entre la población nueva y la actual. Esto sucede de forma iterativa hasta que un criterio de detención indique lo contrario.

- **Metaheurísticas para optimización multiobjetivo:** Existen problemas de optimización en donde se quieren maximizar algunos objetivos y minimizar otros, por ejemplo en el diseño de un producto, en donde se puede minimizar el costo y maximizar la calidad. Para este caso la solución óptima no es única como para problemas mono-objetivo, sino que se trata de un set de soluciones definidas como las soluciones óptimas de Pareto. Una solución de Pareto óptima es aquella que no puede mejorar un objetivo dado sin deteriorar al menos otro objetivo. Describir el comportamiento de estas metaheurísticas requeriría de una sección completa, pero como este proyecto no se enfocará en este tipo de algoritmos es que se decidió dar una idea general de lo que realizan.

2.4 IRACE

Los problemas de optimización requieren, por lo general, configurar un gran número de parámetros para optimizar su rendimiento. Es el caso del problema que se presenta en este trabajo, pues se necesita encontrar la configuración óptima de ciertos parámetros que se detallarán en la sección 4.2.1, a modo de que el rendimiento del software sea el más eficiente posible.

López-Ibáñez et al. (2016) desarrollaron Irace, el cuál es un paquete de R y de Python que implementa un algoritmo llamado *Iterated racing procedure* para optimizar los parámetros de un algoritmo dado de forma automática, o dicho de otro modo encontrar la mejor configuración de un algoritmo. Para ello, el usuario le entrega un rango a verificar de los parámetros que desea optimizar, junto con las instancias de entrenamiento y la configuración del escenario de prueba. Luego, mediante un archivo llamado *targetRunner* se indica el código del algoritmo que Irace debe ejecutar. Finalmente, busca en el espacio de búsqueda de los parámetros buenas configuraciones, ejecutando el algoritmo en distintas instancias y con distintas configuraciones de parámetros.

En la Figura 2.3 se presentan algunos de los elementos mencionados en esta sección.

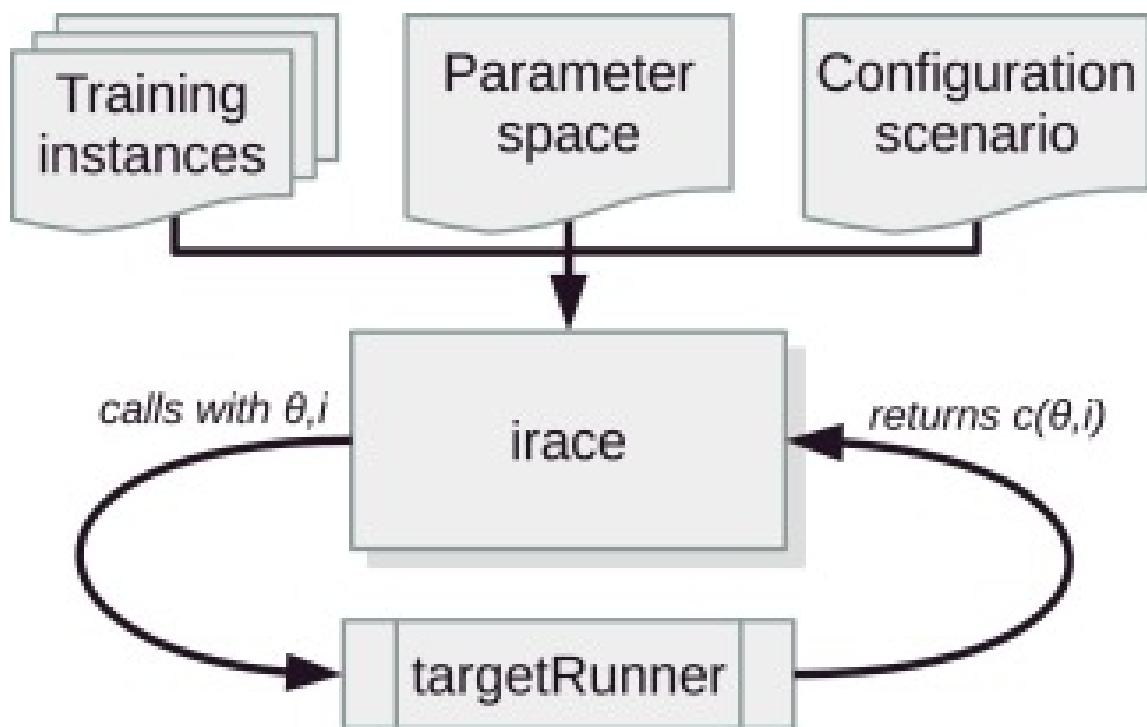


Figura 2.3: Esquema de los componentes requeridos por Irace

Fuente: López-Ibáñez et al. (2016)

CAPÍTULO 3. ESTADO DEL ARTE

La saga de Starcraft ha sido objeto de estudio desde su salida en 1998, pero no fue hasta el año 2010, con el lanzamiento de Starcraft 2, que los estudios con la palabra clave "Starcraft" se dispararon a más de 30 y hasta 50 por año como se puede apreciar en la Figura 3.1. Esto demuestra el interés que se tiene por parte de la comunidad científica de resolver los problemas que este juego introduce, pues influyen en la creación inteligencias artificiales cada vez más robustas y eficientes, así como en disminuir el tiempo de desarrollo de videojuegos del mismo género.

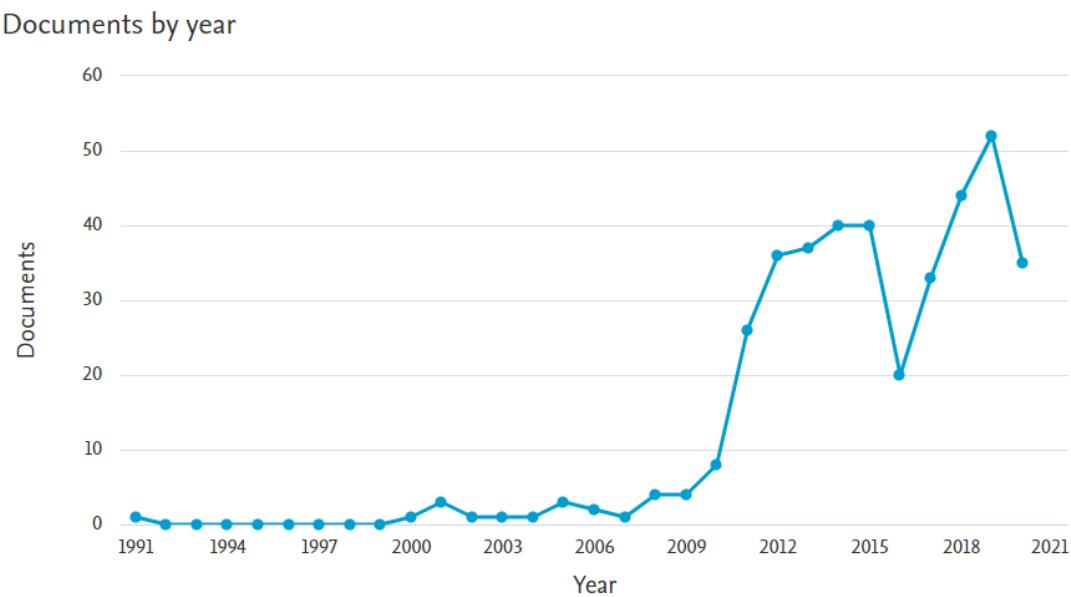


Figura 3.1: Documentos por año sobre el tema "Starcraft" en Scopus.

Fuente: Elaboración propia (2021).

A continuación, se mencionarán algunos de los estudios más relevantes para este proyecto que denotan el estado del arte actual del problema, partiendo por aquellos estudios que se dedicaron a las primeras ediciones del juego Starcraft y Starcraft: Brood War, siguiendo con el juego actual Starcraft 2.

Takino & Hoki (2015) desarrollaron un algoritmo que intenta imitar las acciones de un jugador humano en el videojuego Starcraft, mediante el análisis de los órdenes de construcción iniciales del oponente y utilizando órdenes de construcción prestablecidos. Los resultados demostraron que el algoritmo logra contrarrestar estrategias enemigas en la mayoría de los casos. Por otro lado, los autores afirman que la metodología de Machine Learning no es aconsejable ya que requiere extraer los datos de repeticiones de partidas anteriores y su transformación a una

estructura de datos matricial, que es la que se usó en el artículo, implica estimar muchos datos como, por ejemplo, los tiempos que involucran movimiento, cosa que no es aconsejable para este tipo de inteligencias artificiales debido a que requieren precisión.

Siguiendo la misma idea, Justesen & Risi (2017) desarrollaron el algoritmo COEP (Continual Online Evolutionary Planning) el cuál mejora el bot de código abierto UAlbertaBot (Diseñado para la primera entrega de Starcraft) agregando una capa de decisiones que le permite seleccionar distintos órdenes de construcción dependiendo de los datos que obtiene en tiempo real en la partida. Este bot ganó el 91,7% de las partidas contra el bot integrado del juego, el cual solo usa órdenes de construcción preestablecidos, y venciendo en varias ocasiones al mismo UAlbertaBot (Sin alterar). Este algoritmo sin embargo no era muy efectivo ante las estrategias que incluían un ataque temprano en la partida.

Churchill et al. (2019) crean una inteligencia artificial para Starcraft que divide en micro y macro estrategias los distintos órdenes de construcción, por micro estrategia se entiende todos aquellos órdenes de construcción que tienen como finalidad un objetivo a pocos movimientos de distancia, mientras que una macro estrategia es aquella que requiere una cantidad de órdenes de construcción considerablemente más grande, como por ejemplo construir un ejército de más de 50 unidades mejoradas (con tecnologías que mejoran ciertos atributos como ataque y defensa). Esta IA ganó el 100% de las partidas realizadas contra las IA integradas en el juego.

Wang et al. (2019) plantearon un problema de órdenes de construcción con restricciones, las cuales incluyen unidades, tecnologías, suministros, tiempo y edificios; problema se buscó resolver mediante un enfoque orientado a datos de repeticiones de juegos anteriores. Se demuestra en el artículo que el enfoque es mejor que aquellos que requieren de la entrada de datos de expertos en el dominio.

Pasando ahora a Starcraft 2, Köstler & Gmeiner (2013) presentaron un algoritmo genético multiobjetivo que buscaba optimizar los órdenes de construcción iniciales, de modo que se obtenga un mejor comienzo en la partida. Para esto se basaron en la raza Protoss y obtuvieron buenos resultados optimizando órdenes de construcción ya existentes, pero los autores afirman que para la raza Zerg y Terran son necesarios ciertos ajustes debido a incompatibilidades con el algoritmo y ciertos aspectos de la jugabilidad de estas razas, como por ejemplo que la raza Zerg obliga al jugador a sacrificar trabajadores para construir edificios, lo cual afecta directamente en la velocidad en que se extraen recursos y por ende en el tiempo en que se obtienen las distintas unidades o tecnologías. Por otro lado, la raza Terran permite generar trabajadores llamados “Mule” (Mobile Utility Lunar Excavator, por sus cifras en inglés) desde un satélite, sin impactar en el tiempo o en la economía del juego, así como también pueden hacer que sus edificios vuelen e intercambien estructuras anexas llamadas “Tecnolab” y “Reactor”, lo cual permite desbloquear

distintas unidades o tecnologías.

Volz et al. (2019) desarrolla un modelo que predice al ganador mediante un algoritmo de redes neuronales artificiales y el análisis de los datos de las partidas de Starcraft 2.

Y finalmente Wang et al. (2020) desarrollan un algoritmo utilizando integrales difusas que permite encontrar un conjunto de tropas capaces de enfrentarse a las tropas del rival. Sin embargo, este algoritmo no mostró ser eficaz para todas las tropas, ya que algunas poseen habilidades especiales y los autores especifican que es una labor que tendrán que tomar en cuenta más adelante.

Para comodidad del lector, en la tabla 3.1 se presenta un resumen con el estado del arte.

Tabla 3.1: Resumen estado del arte

Fuente	Descripción
Takino & Hoki (2015)	Algoritmo que imita acciones de un jugador humano. Utiliza órdenes de construcción prestablecidos.
Justesen & Risi (2017)	Algoritmo que mejora al bot UAlbertaBot agregando una capa de decisiones que permite seleccionar órdenes de construcción conocidos.
Churchill et al. (2019)	Crean inteligencia artificial que divide en micro y macro estrategias los distintos órdenes de construcción.
Wang et al. (2019)	Plantean solucionar el problema de generar órdenes de construcción mediante un enfoque de análisis de repeticiones de juegos anteriores.
Köstler & Gmeiner (2013)	Algoritmo genético multiobjetivo que busca optimizar órdenes de construcción iniciales para obtener un mejor comienzo en la partida.
Volz et al. (2019)	Desarrollan modelo que predice un ganador mediante un algoritmo de redes neuronales y el análisis de datos de las partidas de Starcraft 2.
Wang et al. (2020)	Algoritmo que implementa integrales difusas. Permite encontrar un conjunto de tropas capaz de enfrentarse a las tropas del rival.

Fuente: Elaboración propia, 2021

CAPÍTULO 4. ALGORITMO PROPUESTO Y DISEÑO EXPERIMENTAL

En este capítulo se procederá a explicar el diseño y la implementación del sistema de software comprometido para satisfacer el objetivo principal del proyecto. También se presentarán la forma de recolección de los datos obtenidos a partir de la ejecución del software.

4.1 ALGORITMO PROPUESTO

4.1.1 Búsqueda local iterada

Se propone la utilización del algoritmo de búsqueda local iterada debido a que es una metaheurística de solución única, lo cuál es lo que necesita este proyecto, pues se intenta encontrar una solución eficiente y no una población de soluciones. Según Talbi (2009) este algoritmo plantea buscar una solución mediante la utilización de un algoritmo de búsqueda local, para luego perturbar dicha solución y aplicar otra vez una búsqueda local hasta satisfacer las condiciones dadas por el usuario.

En este caso, el algoritmo de búsqueda local que se implementará es un algoritmo goloso el cuál a partir de una primera solución genera n perturbaciones, se calcula un puntaje para cada solución y se elige el mejor, para luego ser perturbado nuevamente. Este proceso se repite por m generaciones almacenando siempre la solución con el mejor puntaje, el cuál se calcula con la ecuación 4.4 descrita en la sección 4.1.4. En el algoritmo 4.1 se puede apreciar lo descrito anteriormente.

Una vez implementado el algoritmo de búsqueda local se procede a implementar el algoritmo de búsqueda local iterada, el cuál como su nombre indica es realizar el proceso de búsqueda local reiteradas veces a modo de encontrar una solución eficiente global. El concepto es prácticamente el mismo que el del algoritmo goloso, solo que en esta ocasión, cada vez que se perturba una función se procede a realizar una búsqueda local y si la solución resultante es mejor que la anterior, en términos de tiempo y de cantidad de entidades, entonces esa solución se considera la mejor. Esto quedará mejor explicado con el algoritmo 4.2.

Algoritmo 4.1: Algoritmo goloso. Fuente: Elaboración propia (2021)

```
1 bestSolution ← Orden de construcción aleatorio;
2 score ← Puntuación de bestSolution (Ec. 4.4);
3 I ← 1;
4 N ← Iterations;
5 M ← Perturbations;
6 while I ≤ N do
7   | P ← 1;
8   | while P ≤ M do
9     |   | perturbedSolution ← Se perturba bestSolution;
10    |   | perturbedSolutionScore ← Puntuación de perturbedSolution (Ec. 4.4);
11    |   | if perturbedSolutionScore < score then
12      |     | score ← perturbedSolutionScore (Ec. 4.4);
13      |     | bestSolution ← perturbedSolution
14    |   | end
15    |   | P ← P + 1;
16   | end
17   | I ← I + 1
18 end
```

Algoritmo 4.2: Algoritmo de búsqueda local iterada. Fuente: Elaboración propia (2021)

```
1 bestSolution ← Orden de construcción obtenido de una búsqueda local;
2 bestSolutionScore ← puntuación de bestSolution (Ec. 4.4);
3 I ← 1;
4 N ← Iterations;
5 while I ≤ N do
6   | perturbedSolution ← Se perturba bestSolution;
7   | localSolution ← Búsqueda local a partir de perturbedSolution;
8   | localSolutionScore ← Puntuación de localSolution (Ec. 4.4);
9   | if localSolutionScore < bestSolutionScore then
10  |   | bestSolution ← localSolution;
11  | end
12  | I ← I + 1
13 end
```

4.1.2 Representación de datos

4.1.2.1 Árbol de tecnologías

Para generar órdenes de construcción es necesario modelar e implementar en el código el árbol de tecnologías de la raza. Para esto, se decidió utilizar un enfoque de grafos de modo que, para verificar si una entidad cumple con sus requisitos, simplemente se buscaría el camino más corto desde el nodo inicial. Cada nodo tendrá los atributos de la entidad, los cuales se implementan con la siguiente notación:

- **name**: nombre de la entidad.
- **code**: código identificador de la entidad.
- **minerals**: minerales requeridos para su construcción.
- **vespene**: gas vespene requerido para su construcción.
- **gameSpeed**: tiempo que demora en ser construida la entidad.
- **initialEnergy**: energía inicial de la entidad.
- **maxEnergy**: energía máxima que puede contener la entidad.
- **supply**: suministros necesarios para construir la unidad.
- **type**: tipo de entidad. Puede ser de tipo edificio (*Building*), de tipo unidad (*Unit*) o de tipo tecnología (*Tech*).
- **buildOn**: código del nodo en el que se construye la entidad. Si bien pareciera que simplemente encontrando el camino más corto al nodo objetivo se puede obtener el nodo donde se construye, esto no es así, ya que por ejemplo la unidad *Dark Templar* se construye en el edificio *Gateway*, pero se desbloquea al construir el edificio *Dark Shrine*. El camino más corto al nodo no diferencia si los nodos son un requisito o si son el edificio constructor.

Todos los datos de los atributos son extraídos de Liquipedia (2021b).

En la Figura 4.1 se puede observar el grafo resultante. Se debe hacer notar que el grafo no es idéntico al presentado por el juego, ya que si bien el edificio *Pylon* aparentemente no es requisito de nada, en realidad es requisito de casi todos los edificios, pues para construir los edificios debe existir previamente un *Pylon* en el área. Para efectos del proyecto

se considera que el *Pylon* es prerequisito de los demás edificios, sin considerar al *Nexus* ni al *Assimilator* ya que son los únicos edificios que no necesitan un *Pylon* para ser construidos.

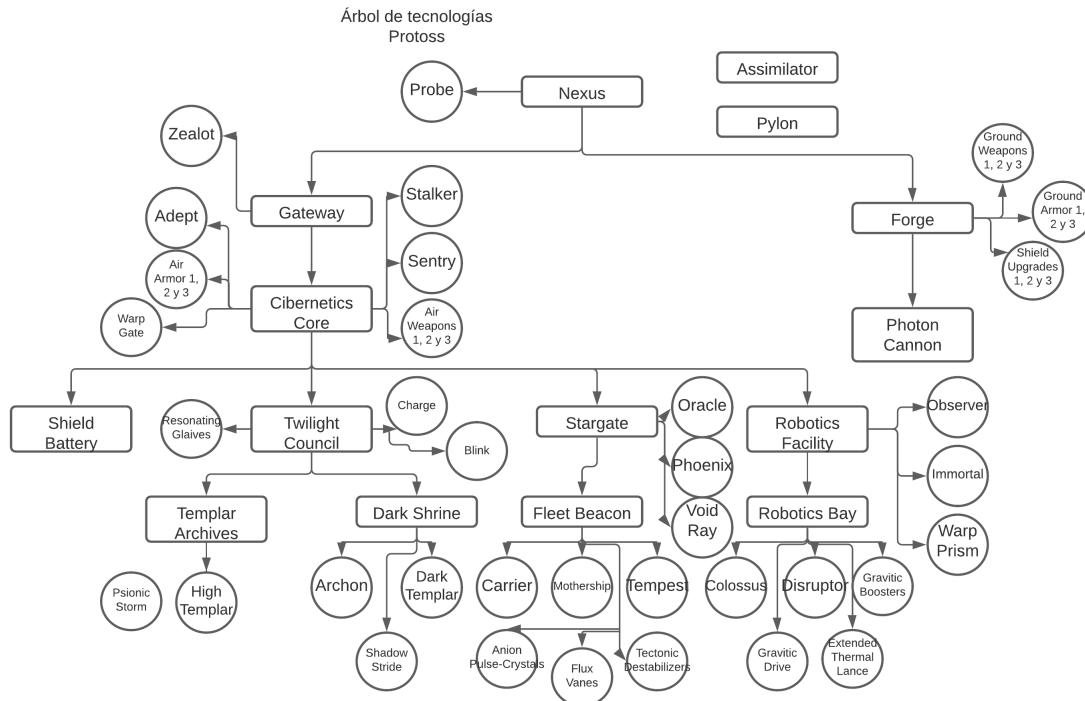


Figura 4.1: Implementación del árbol de tecnologías.

Fuente: Elaboración propia (2021).

Para la implementación del grafo se utiliza la librería *igraph-python* que permite construir de manera sencilla un grafo y que ya posee funciones útiles como la de obtener el camino más corto. También permite acceder fácilmente a los atributos de los nodos en caso de que sea necesario.

4.1.2.2 Orden de construcción

Como se mencionó en la sección 2.1.2 el orden de construcción se compone por los siguientes datos:

- **time (T)**: es el tiempo en el que se genera la orden de construir una entidad.
- **entityName (N)**: es el nombre de la entidad a construir.
- **supplyOccupied (S_o)**: son los suministros ocupados que hay al momento de ejecutar una orden de construir una entidad.

- **supplyTotal** (St): son los suministros disponibles al momento de ejecutar la orden de construir una entidad.
- **minerals** (M): minerales disponibles al momento de ejecutar la orden de construir una entidad.
- **vespeneGas** (V): gas vespeno disponible al momento de ejecutar la orden de construir una entidad.
- **chronoBoost** (Cb): si es posible utilizar la habilidad *Chrono boost* entonces se mostrará en el orden de construcción.

Además es necesario incluir datos que serán utilizados de manera interna en el programa para llevar a cabo procesos que se estudiarán en los capítulos posteriores. Estos son:

- **constructionQueue** (Cq): cola de construcción de edificios. Como se pueden construir múltiples edificios a la vez, se tienen que guardar en una lista de datos junto con su tiempo de construcción respectivo.
- **nodeQty** (Nq): lista que contiene la cantidad de entidades que hay en un momento dado.
- **unitQueue** (Uq): cola de construcción de unidades y tecnologías. En esta lista se agregan los edificios que pueden construir unidades o investigar tecnologías y cuando se ordena construir una de estas entidades se agrega a un edificio disponible junto con el tiempo de construcción de la entidad. También, deja lugar para el tiempo de enfriamiento por haber utilizado *Warp Gate* en los *Gateway* y para el tiempo de la habilidad *Chrono Boost* que puede durar hasta 20 segundos.
- **archonQueue** (Aq): cola de construcción de la unidad especial *Archon*. Como se mencionó en la sección 2.1.6 estas unidades no necesitan un edificio para ser construidas, en su lugar utilizan a dos unidades de *Dark Templar* o de *High Templar* y se construye en el mismo lugar donde se junten las unidades.

De esta manera, un orden de construcción queda representado como:

$$Oc = [A_0, \dots, A_{n-1}] \quad (4.1)$$

Oc = Orden de construcción.

$$A_i = [T_i, N_i, So_i, St_i, M_i, V_i, Cb_i, Cq_i, Nq_i, Uq_i, Aq_i]$$

n = Cantidad de acciones realizadas.

4.1.3 Operadores

4.1.3.1 Obtener un orden de construcción aleatorio

Una vez definidos las estructuras de datos, así como los algoritmos a implementar, es que se puede comenzar a modelar la generación de un orden de construcción a partir de los parámetros indicados por el usuario, los cuáles son: nombre de la entidad, cantidad deseada y tiempo máximo del orden de construcción. Así mismo, se le ofrece la posibilidad al usuario de indicar la cantidad de iteraciones de los algoritmos goloso y de búsqueda local iterada, así como el número de perturbaciones por generación. También estará la opción de utilizar los parámetros por defecto, los cuales estarán parametrizados. El proceso de parametrización se encuentra detallado en secciones posteriores del presente documento.

Para obtener un orden de construcción aleatorio, se realizó un ciclo desde el segundo 0 hasta el segundo indicado por el usuario. Una partida comienza con 13 *Probes* (trabajadores), los cuales comienzan a recolectar minerales de inmediato, y un *Nexus* para entrenar más *Probes*. Por cada iteración se elige un nodo al azar del árbol de tecnologías y se verifica si cumple con los requisitos para ser construido, esto es, verificar si existen los minerales, gas vespeno, suministros y prerequisitos necesarios para la construcción de la entidad solicitada. Si cumple con todos los requisitos, entonces se debe decidir a qué cola de construcción se agregará la entidad. Recordar que existen 3 colas de construcción: **constructionQueue**, **unitQueue** y **archonQueue**.

- **Si la entidad es un edificio:** Se agrega a *constructionQueue* con el tiempo de construcción que se encuentra en el nodo de la entidad.
- **Si la entidad no es un edificio:**
 - **Si la entidad no es un Archon:** Se agrega la entidad a *unitQueue* en el edificio que indica el atributo *buildOn* del nodo junto con el tiempo de construcción. Cabe destacar que tanto unidades como tecnologías son agregadas a *unitQueue* ya que se construyen de la misma manera.
 - **Si es un Archon:** Se eliminan dos unidades de *Dark Templar* o de *High Templar*. Luego se agrega a la cola *archonQueue* junto con su tiempo de construcción.

También se verifica si existe algún *Nexus* con suficiente energía como para aplicar su habilidad *Chrono boost*. Si existe, entonces se agrega al orden de construcción y aparecerá reflejado en la salida del programa.

Al inicio de cada ciclo se verifica si las colas de construcción están siendo ocupadas. Si es así, entonces se procede a descontar segundos de construcción a las entidades que están

primeras en las colas. Cuando ese tiempo llega a 0 es cuando se agrega la entidad a **nodeQty**, una lista que contiene todas las entidades construidas. Para el caso de *unitQueue* se verifica además si se está usando la tecnología *Warp Gate*, la cual permite construir unidades del edificio *Gateway* más rápido. Si está activo, entonces se cambian los atributos de *gameSpeed* en el nodo de las entidades que son construidas en los *Gateway*. También se verifica si se está usando *Chrono boost* en el edificio, si es así, entonces se descuenta 1.5 segundos en vez de 1 en el tiempo restante para construir la entidad. Mientras tanto, en la cola *constructionQueue*, si llega el momento de construir el edificio, se verifica si es un *Pylon* o un *Nexus* ya que estos edificios otorgan 5 y 15 suministros respectivamente y se deben sumar a *totalSupply*. Si el edificio que construyó es capaz de construir unidades o de investigar tecnologías, entonces se agregará a la cola *unitQueue* para que pueda construir unidades o tecnologías.

Mientras tanto, los trabajadores siguen recolectando recursos, aunque para efectos de este proyecto la recolección se simplificó, ya que como se mencionó en la sección 1.4.3 hubo algunos aspectos del juego que, por su complejidad, tuvieron que ser dejados de lado. Para entender como se recolectan recursos se presenta el algoritmo 4.3.

Algoritmo 4.3: Recolección de recursos. Fuente: Elaboración propia (2021)

```

1 probesTotal ← Cantidad de probes en total;
2 assimilatorsTotal ← Cantidad de asimiladores en total;
3 maxProbesInVespene ← 3 * assimilatorsTotal;
4 probesMiningVesp ← 0;
5 probesMiningMinerals ← 0;
6 probe ← 1;
7 while probe ≤ probesTotal do
8   if maxProbesInVespene > 0 probesMiningVesp ≤ maxProbesInVespene then
9     probesMiningVesp ← probesMiningVesp + 1;
10    probe ← probe + 1;
11  end
12  probesMiningMine ← probesMiningMine + 1;
13 end
14 minerals ← minerals + probesMiningMinerals * 0.916;
15 vespene ← vespene + probesMiningVesp;

```

Con este algoritmo, se asignan trabajadores a recolectar vespene sólo si existen *Assimilators* y no más allá de 3 trabajadores por edificio. También destacar que según Liquipedia (2021c) cuando un campo de mineral está completamente saturado, esto es, con 3 trabajadores minándolo, entonces se obtiene un total de 0.916 minerales por segundo. Mientras que con un *Assimilator* completamente saturado, se obtienen 1 de vespene por segundo. En este proyecto se asume que siempre están saturados los campos de minerales y los *Assimilators*.

4.1.3.2 Operador de perturbación

Para generar una perturbación en el orden de construcción, simplemente se selecciona un número al azar entre 0 y el tiempo máximo del orden de construcción inicial y se eliminan las acciones posteriores a ese segundo en el orden de construcción. Luego se procede a generar el orden de construcción de la misma manera en que se explicó en esta sección.

Un ejemplo con un orden de construcción simplificado se muestra en la Tabla 4.1.

Tabla 4.1: Orden de construcción inicial

Tiempo	Nombre
00:00:30	Assimilator
00:00:46	Nexus
00:00:50	Pylon
00:01:04	Probe
00:01:34	Probe
00:01:39	Pylon
00:01:48	Forge
00:01:56	Forge
00:02:04	Probe

Fuente: Elaboración propia, 2021

Luego de aplicar la función de perturbación, en donde el tiempo escogido de manera aleatoria fue el minuto 00:01:04, resulta en el orden de construcción presentado en la Tabla 4.2.

Tabla 4.2: Orden de construcción perturbado

Tiempo	Nombre
00:00:30	Assimilator
00:00:46	Nexus
00:00:50	Pylon
00:01:04	Probe
00:01:14	Assimilator
00:02:01	Probe
00:02:10	Pylon

Fuente: Elaboración propia, 2021

Se puede apreciar que desde el minuto 00:01:04 el orden de construcción cambia y que incluso se realizaron menos acciones. Esto es debido a que en cada iteración se escoge un nodo al azar y si no cumple con los requisitos entonces simplemente no se construye y se pasa al siguiente segundo.

4.1.4 Criterio de optimización

Para decidir cuál orden de construcción es mejor con respecto a los de su generación es que fue necesario implementar un sistema que otorgue puntaje a cada orden de construcción en los algoritmos de búsqueda local y de búsqueda local iterada. Este puntaje considera tanto las entidades construidas como el tiempo final del orden de construcción, a menor tiempo menor puntaje y a mayor cantidad de entidades construidas menor puntaje. Mientras menor sea el puntaje, mejor será la solución entregada por el algoritmo.

$$P_t = \frac{t_0}{t_{max}} \quad (4.2)$$

Siendo:

P_t = Puntaje de tiempo.

t_0 = Tiempo final de la solución actual.

t_{max} = Tiempo máximo entregado por el usuario.

Luego se debe calcular el puntaje de las entidades obtenidas en relación a la cantidad de unidades requerida por el usuario y las entidades que son prerequisitos para llegar a esa entidad. Por ejemplo, si el usuario pide construir cinco unidades de *Zealot*, se deben considerar también los requisitos de esa unidad, es decir, construir un *Pylon*, luego un *Gateway* y luego los cinco *Zealot*. Para el puntaje obtenido por las entidades se considera la ecuación 4.3.

$$P_e = \frac{E_o - E_c}{E_o} \quad (4.3)$$

Siendo:

P_e = Puntaje de entidades.

E_c = Entidades construidas.

E_o = Entidades totales que se deben construir.

Cabe destacar que el algoritmo podría encontrar más unidades de las requeridas, por ejemplo, al encontrar 6 unidades de *Dark Templar* en vez de 4 que fue lo ingresado por el usuario. Si eso llega a pasar entonces el puntaje de entidades sería negativo, lo cual no ocasiona ningún problema ya que sigue minimizando el tiempo y las unidades de la solución. Si bien el algoritmo está diseñado para detenerse cuando encuentra las entidades objetivo, existen ocasiones en que en un segundo se crean varias entidades al mismo tiempo, como cuando 2 *Gateway* terminan de crear un *Zealot*, por eso es posible que se encuentre una solución que exceda lo requerido.

Finalmente, una solución obtenida puede tener muy poco tiempo, pero quizás no llegue a cumplir el objetivo de entidades por construir, por lo tanto es importante darle prioridad a

la construcción de entidades. Es por esto que se decide multiplicar el puntaje de tiempo por una ponderación de 0,2 y al puntaje de entidades por una ponderación de 0,8. De esta manera, la ecuación final para otorgar el puntaje queda expresada en la ecuación 4.4.

$$P = 0.2 * P_t + 0.8 * P_e \quad (4.4)$$

4.2 DISEÑO EXPERIMENTAL

4.2.1 Parametrización

Para que los algoritmos de búsqueda local y búsqueda local iterada entreguen las mejores soluciones posibles, es necesario determinar los parámetros de entrada, los cuales son: generaciones del algoritmo goloso, perturbaciones por generación e iteraciones del algoritmo de búsqueda local iterada. Para esto se utilizó la herramienta IRace definida en el capítulo 2. El proceso para instalar y configurar IRace se define en el manual de usuario entregado por López-Ibáñez et al. (2020).

Para cada parámetro, se le entrega un rango de entre 10 y 50 para que el algoritmo de IRACE determine cuáles son los valores óptimos que maximicen el rendimiento del algoritmo. Esto queda expresado en la tabla 4.3.

Tabla 4.3: Parámetros a evaluar utilizando Irace y el rango entregado

Parámetro	Rango
Iteraciones	(10-50)
Generaciones Goloso	(10-50)
Perturbaciones	(10-50)

Fuente: Elaboración propia, 2021

4.2.2 Evaluación

Es necesario determinar la precisión y la convergencia del algoritmo en términos estadísticos. Para esto, se realizarán 3 pruebas que constarán de 11 ejecuciones del algoritmo de búsqueda local iterada y 3 pruebas de 11 ejecuciones del algoritmo goloso con 11 ejecuciones cada una. Cada prueba tendrá por objetivo encontrar un orden de construcción de alguna unidad del árbol de tecnologías para, mas adelante, verificar si la metaheurística entrega mejores

resultados que el algoritmo goloso. Los mejores resultados de las pruebas se utilizarán para compararlos con órdenes de construcción conocidos y publicados por la comunidad.

4.2.2.1 Convergencia del algoritmo

Una vez realizadas las pruebas se graficarán los resultados para estudiar la convergencia del algoritmo de búsqueda local iterada, así como la del algoritmo goloso.

4.2.2.2 Comparación con órdenes de construcción conocidos

Se compara el mejor orden de construcción de cada prueba con algún orden de construcción conocido y valorado positivamente por la comunidad, y se verifica si ofrecen mejores o peores resultados. De esta manera se puede saber si el resultado se acerca a la solución óptima.

4.2.2.3 Aplicación en juego

El presente autor aplicará el mejor orden de construcción obtenido de cada prueba en una partida privada contra el ordenador, es decir, sin otros participantes. De esta manera se podrá observar qué dificultades se podrían tener al momento de aplicar la solución y los tiempos que podría demorar un jugador al utilizarla. Cabe destacar que el autor no es un jugador profesional, por lo que esta sección debe ser considerada como datos adicionales que podrían aportar mayor contexto al estudio. Esto se hace debido a que el algoritmo entrega los mejores tiempos pero no considera el factor humano, es decir, los errores que se pueden cometer al jugar. Tampoco considera la velocidad de movimiento de las unidades, por lo que es probable que no se pueda construir un edificio en el segundo preciso que indica el orden de construcción. También hay que considerar que para obtener mejores resultados el usuario debe poseer un nivel avanzado de conocimiento del juego. Una persona que juega por primera vez rendirá mucho peor que una persona que lleva años jugando.

CAPÍTULO 5. RESULTADOS EXPERIMENTALES

5.1 PARAMETRIZACIÓN

Una vez finalizada la ejecución de Irace, se obtienen los mejores valores para los parámetros de generaciones del algoritmo goloso, de perturbaciones por generación del algoritmo goloso y de iteraciones del algoritmo de búsqueda local iterada. La Tabla 5.1 muestra los resultados obtenidos.

Tabla 5.1: Parametrización entregada por Irace

Generaciones	Perturbaciones	Iteraciones
20	21	11

Fuente: Elaboración propia, 2021

5.2 EVALUACIÓN

5.2.1 Resultados de las pruebas

A continuación se muestran los resultados obtenidos para la primera prueba, la cuál corresponde a obtener 5 unidades de Zealots. La prueba ejecutará 11 veces el algoritmo de búsqueda local iterada y también el algoritmo goloso para posterior análisis. La metaheurística utiliza los parámetros obtenidos en la sub sección anterior, por lo que en cada ejecución se evalúan 4620 órdenes de construcción. Para que el algoritmo goloso evalúe la misma cantidad de soluciones se configura para que realice 11 iteraciones con 420 perturbaciones.

- Prueba n° 1: Obtener 5 Zealots

En la tabla 5.2 se muestran los resultados de la metaheurística y en la tabla 5.3 los resultados del algoritmo goloso. También se especificarán los resultados al realizar el test de rangos de Wilcoxon entre las mejores soluciones obtenidas por los algoritmos, pero se concluirá con respecto a ello en la sección 6.

Tabla 5.2: Resultados entregados por la metaheurística para obtener 5 Zealots

	Ejecución										
	1	2	3	4	5	6	7	8	9	10	11
Generación	Puntaje										
1	0,103	0,142	0,117	0,113	0,313	0,109	0,106	0,094	0,142	0,097	0,129
2	0,103	0,142	-0,029	0,113	0,095	0,109	0,106	0,094	0,142	0,097	0,129
3	0,066	0,089	-0,029	0,092	0,095	0,109	0,095	0,094	0,125	0,097	0,096
4	0,066	0,089	-0,029	0,092	0,095	0,109	0,095	0,094	0,118	0,097	0,096
5	0,066	0,089	-0,029	0,092	0,095	0,109	0,095	0,086	0,118	0,097	0,086
6	0,066	0,089	-0,029	0,092	0,095	-0,029	0,095	0,086	0,118	0,097	0,086
7	0,066	0,089	-0,029	0,092	0,095	-0,029	0,079	0,086	0,107	0,097	0,086
8	0,066	0,089	-0,029	0,092	0,095	-0,029	0,079	0,086	0,107	0,086	0,086
9	0,066	0,089	-0,029	-0,143	0,095	-0,029	0,079	0,086	0,107	0,086	0,086
10	0,066	0,089	-0,029	-0,143	0,095	-0,029	0,079	0,086	0,107	0,086	0,086
11	0,066	0,089	-0,029	-0,143	0,095	-0,029	0,079	0,086	0,105	0,086	0,086

Fuente: Elaboración propia, 2021

Siendo,

- Generación: iteración del algoritmo de búsqueda local iterada en una ejecución.
- Ejecución: ejecución completa con 11 generaciones del algoritmo de búsqueda local iterada.
- Puntaje: puntaje calculado para el mejor orden de construcción obtenido en cada generación de la metaheurística. Si la solución tiene un puntaje mayor que la mejor solución calculada hasta el momento, entonces se descarta y se mantiene el mejor puntaje.

Se pueden apreciar valores negativos en los puntajes, esto es debido a que el algoritmo logró encontrar un orden de construcción con más unidades de las que el usuario solicitó y dentro del rango de tiempo especificado.

Tabla 5.3: Resultados entregados por el algoritmo goloso para obtener 5 Zealots

	Ejecución										
	1	2	3	4	5	6	7	8	9	10	11
Generación	Puntaje										
1	0,200	0,269	0,269	0,200	0,126	0,200	0,200	0,200	0,269	0,110	0,200
2	0,145	0,269	0,199	0,200	0,126	0,124	0,200	0,200	0,269	0,110	0,200
3	0,145	0,200	0,199	0,148	0,126	0,124	0,200	0,200	0,269	0,110	0,152
4	0,145	0,086	0,086	0,148	0,126	0,124	0,200	0,200	0,200	0,110	0,152
5	0,145	0,086	0,084	0,148	0,126	0,124	0,133	0,200	0,085	0,110	0,152
6	0,145	0,086	0,084	0,148	0,126	0,124	0,133	0,200	-0,029	0,110	0,138
7	0,145	0,086	-0,029	0,148	0,126	0,124	0,085	0,200	-0,029	0,110	0,138
8	0,145	0,086	-0,143	0,148	0,126	0,124	-0,029	0,200	-0,029	0,110	0,138
9	0,125	-0,029	-0,258	0,148	0,126	0,124	-0,143	0,200	-0,143	0,110	0,138
10	0,125	-0,143	-0,372	0,148	0,126	0,124	-0,144	0,152	-0,143	0,110	0,138
11	0,125	-0,143	-0,372	0,148	0,126	0,124	-0,257	0,152	-0,143	0,110	0,138

Fuente: Elaboración propia, 2021

A continuación, se muestran los resultados de realizar el test de rangos de Wilcoxon entre la ejecución n° 3 del algoritmo goloso y la ejecución n° 7 de la metaheurística. Siendo la hipótesis nula que la mediana entre las diferencias de los resultados es igual a 0, se obtiene un estadístico de prueba $T = 17$ y con un nivel de significancia $\alpha = 0,05$ se obtuvo un valor $p = 0,145$.

- Prueba n° 2: Obtener 5 Dark Templars

Para la segunda prueba se le solicitó al algoritmo de búsqueda local iterada y al goloso encontrar 5 unidades de Dark Templar. Las tablas 5.4 y 5.6 muestran los resultados respectivamente.

Tabla 5.4: Resultados entregados por la metaheurística para obtener 5 Dark Templars

	Ejecución										
	1	2	3	4	5	6	7	8	9	10	11
Generación	Puntaje										
1	0,407	0,466	0,435	0,433	0,265	0,427	0,271	0,424	0,276	0,249	0,556
2	0,407	0,328	0,428	0,433	0,265	0,427	0,198	0,264	0,185	0,249	0,556
3	0,407	0,328	0,428	0,433	0,265	0,267	0,198	0,226	0,185	0,249	0,504
4	0,407	0,328	0,350	0,433	0,265	0,267	0,198	0,226	0,185	0,249	0,351
5	0,256	0,328	0,350	0,433	0,265	0,267	0,120	0,164	0,185	0,249	0,351
6	0,256	0,328	0,350	0,433	0,265	0,267	0,120	0,164	0,185	0,249	0,351
7	0,256	0,328	0,350	0,433	0,265	0,267	0,120	0,164	0,185	0,249	0,323
8	0,256	0,328	0,350	0,380	0,265	0,171	0,120	0,164	0,185	0,249	0,244
9	0,256	0,328	0,350	0,345	0,265	0,171	0,120	0,164	0,185	0,249	0,244
10	0,256	0,328	0,350	0,345	0,265	0,171	0,120	0,164	0,185	0,249	0,244
11	0,256	0,328	0,350	0,326	0,265	0,171	0,120	0,164	0,185	0,249	0,244

Fuente: Elaboración propia, 2021

Tabla 5.5: Resultados obtenidos por el algoritmo goloso para obtener 5 Dark Templars

	Ejecución										
	1	2	3	4	5	6	7	8	9	10	11
Generación	Puntaje										
1	0,506	0,565	0,575	0,574	0,574	0,531	0,573	0,397	0,573	0,573	0,418
2	0,506	0,565	0,575	0,541	0,574	0,531	0,573	0,397	0,470	0,458	0,418
3	0,506	0,565	0,479	0,541	0,462	0,531	0,573	0,397	0,470	0,458	0,418
4	0,506	0,565	0,479	0,541	0,462	0,531	0,543	0,397	0,396	0,440	0,418
5	0,506	0,565	0,479	0,541	0,462	0,531	0,543	0,397	0,396	0,440	0,418
6	0,506	0,565	0,479	0,541	0,462	0,531	0,543	0,397	0,396	0,440	0,418
7	0,506	0,565	0,479	0,541	0,462	0,531	0,543	0,397	0,396	0,440	0,418
8	0,506	0,565	0,479	0,541	0,462	0,531	0,543	0,397	0,396	0,440	0,418
9	0,506	0,565	0,479	0,541	0,462	0,531	0,543	0,397	0,396	0,440	0,349
10	0,506	0,565	0,479	0,541	0,462	0,531	0,543	0,397	0,396	0,440	0,349
11	0,506	0,565	0,479	0,541	0,462	0,531	0,543	0,397	0,396	0,440	0,349

Fuente: Elaboración propia, 2021

En este caso, se realiza el test de Wilcoxon con la ejecución n° 7 de la metaheurística y la ejecución n° 11 del algoritmo goloso, obteniendo un estadístico de prueba $T = 0$, y un valor $p = 0,0054$ con un nivel de significancia $\alpha = 0,05$.

- Prueba n° 3: Obtener 5 Phoenix

Para la tercera prueba se le solicitó al algoritmo de búsqueda local iterada y al goloso encontrar 5 unidades de Phoenix. Las tablas 5.6 y 5.7 muestran los resultados respectivamente.

Tabla 5.6: Resultados entregados por la metaheurística para obtener 5 Phoenix

	Ejecución										
	1	2	3	4	5	6	7	8	9	10	11
Generación	Puntaje										
1	0,419	0,168	0,284	0,482	0,610	0,165	0,547	0,371	0,431	0,176	0,399
2	0,338	0,168	0,284	0,159	0,419	0,165	0,391	0,280	0,431	0,176	0,309
3	0,338	0,168	0,283	0,159	0,419	0,165	0,391	0,259	0,431	0,176	0,276
4	0,338	0,168	0,283	0,159	0,327	0,165	0,391	0,259	0,378	0,176	0,276
5	0,249	0,168	0,283	0,159	0,154	0,165	0,351	0,259	0,378	0,176	0,276
6	0,249	0,168	0,276	0,159	0,154	0,165	0,351	0,224	0,378	0,176	0,276
7	0,249	0,168	0,276	0,159	0,154	0,165	0,351	0,224	0,378	0,153	0,276
8	0,249	0,168	0,276	0,159	0,154	0,165	0,351	0,224	0,378	0,153	0,111
9	0,249	0,168	0,276	0,159	0,154	0,165	0,320	0,224	0,378	0,153	0,111
10	0,249	0,168	0,258	0,159	0,154	0,165	0,287	0,224	0,378	0,153	0,111
11	0,249	0,168	0,258	0,159	0,154	0,165	0,175	0,224	0,378	0,153	0,111

Fuente: Elaboración propia, 2021

Tabla 5.7: Resultados obtenidos por el algoritmo goloso para obtener 5 Phoenix

	Ejecución										
	1	2	3	4	5	6	7	8	9	10	11
Generación	Puntaje										
1	0,396	0,390	0,159	0,406	0,522	0,523	0,368	0,522	0,522	0,523	0,525
2	0,396	0,390	0,159	0,406	0,522	0,409	0,368	0,522	0,522	0,493	0,525
3	0,396	0,390	0,159	0,406	0,500	0,378	0,368	0,522	0,341	0,493	0,525
4	0,396	0,390	0,159	0,406	0,436	0,378	0,368	0,522	0,341	0,493	0,525
5	0,396	0,390	0,159	0,406	0,436	0,378	0,368	0,494	0,341	0,493	0,525
6	0,396	0,390	0,159	0,406	0,436	0,378	0,368	0,494	0,341	0,493	0,525
7	0,396	0,390	0,159	0,406	0,436	0,378	0,368	0,494	0,341	0,493	0,349
8	0,396	0,390	0,159	0,406	0,436	0,377	0,368	0,494	0,341	0,493	0,349
9	0,396	0,390	0,159	0,406	0,436	0,289	0,368	0,494	0,341	0,493	0,349
10	0,396	0,390	0,159	0,406	0,436	0,200	0,368	0,494	0,341	0,493	0,349
11	0,396	0,390	0,159	0,406	0,436	0,200	0,368	0,433	0,341	0,438	0,327

Fuente: Elaboración propia, 2021

Finalmente, el test de Wilcoxon entre la ejecución n° 11 del algoritmo de búsqueda local iterada y la ejecución n° 3 del algoritmo goloso obtiene un estadístico de prueba $T = 10$ y un valor $p = 0,0492$ con un nivel de significancia de $\alpha = 0,05$.

5.2.2 Convergencia del algoritmo

En esta sección se grafican los resultados anteriormente expuestos para mostrar la convergencia del algoritmo.

- Algoritmo Goloso

En la Figura 5.1, 5.2 y 5.3 se grafican los resultados obtenidos al aplicar el algoritmo goloso en las pruebas n° 1, 2 y 3 respectivamente.

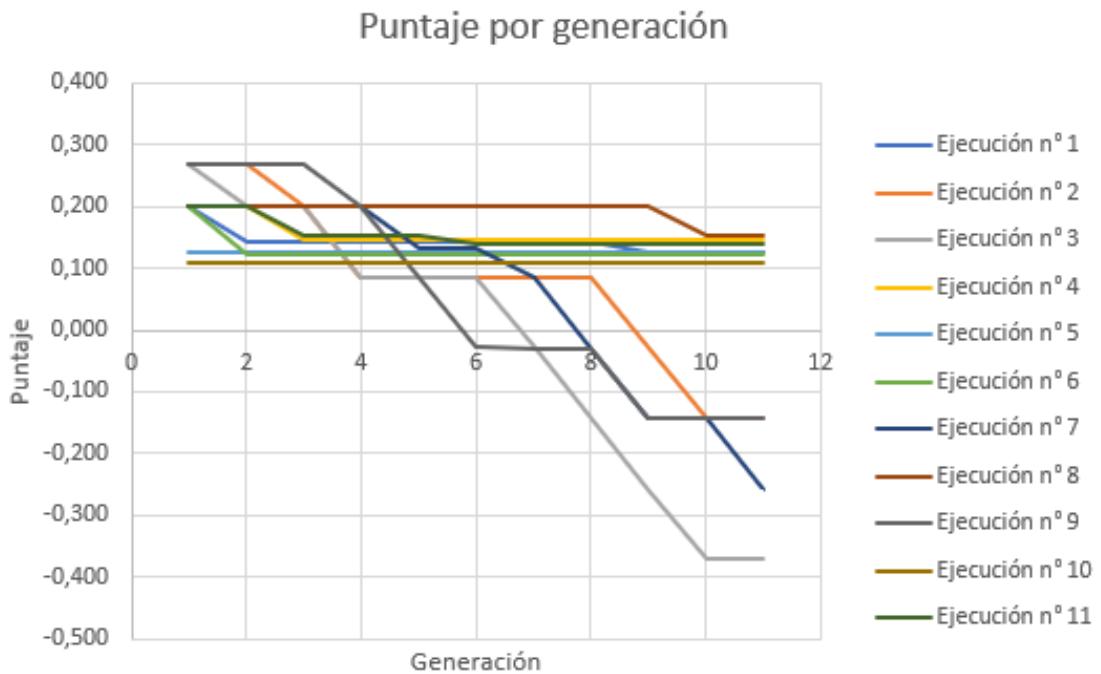


Figura 5.1: Convergencia algoritmo goloso en prueba n° 1

Fuente: Elaboración propia (2021).

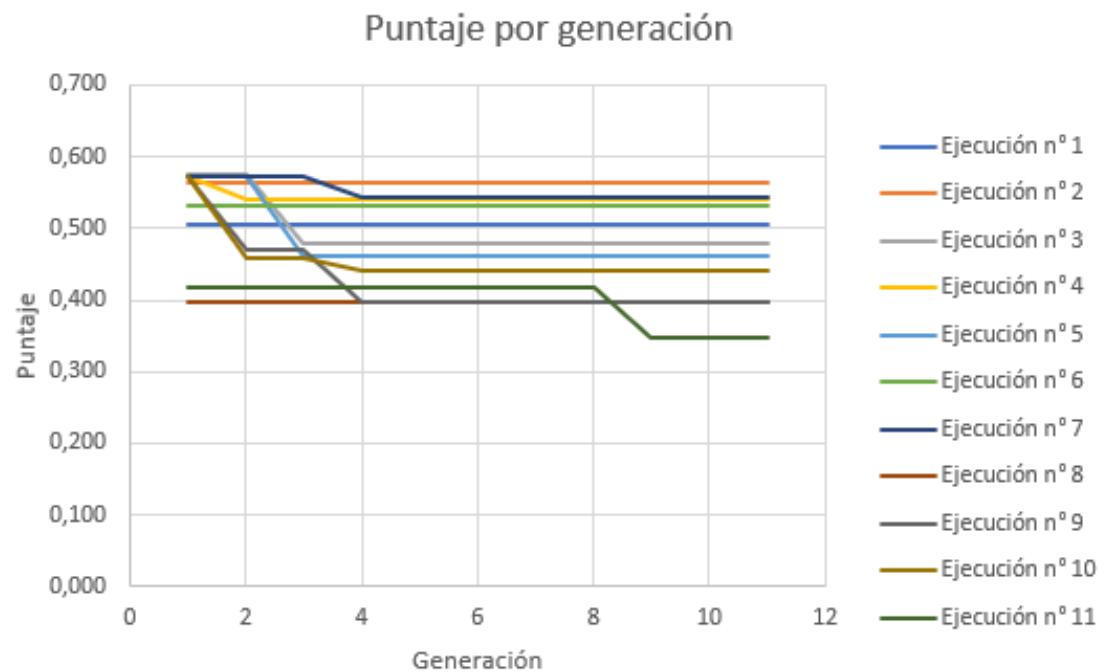


Figura 5.2: Convergencia algoritmo goloso en prueba n° 2

Fuente: Elaboración propia (2021).

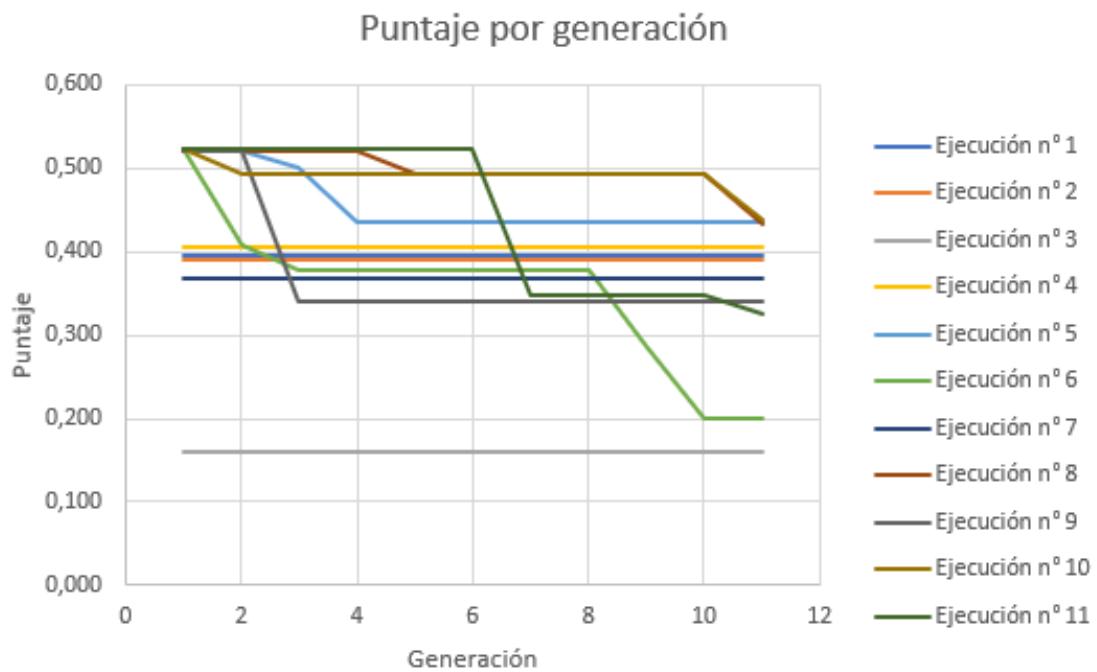


Figura 5.3: Convergencia algoritmo goloso en prueba n° 3

Fuente: Elaboración propia (2021).

- Algoritmo de búsqueda local iterada

En la Figura 5.4, 5.5 y 5.6 se grafican los resultados obtenidos al aplicar el algoritmo de búsqueda local iterada en las pruebas n° 1, 2 y 3 respectivamente.

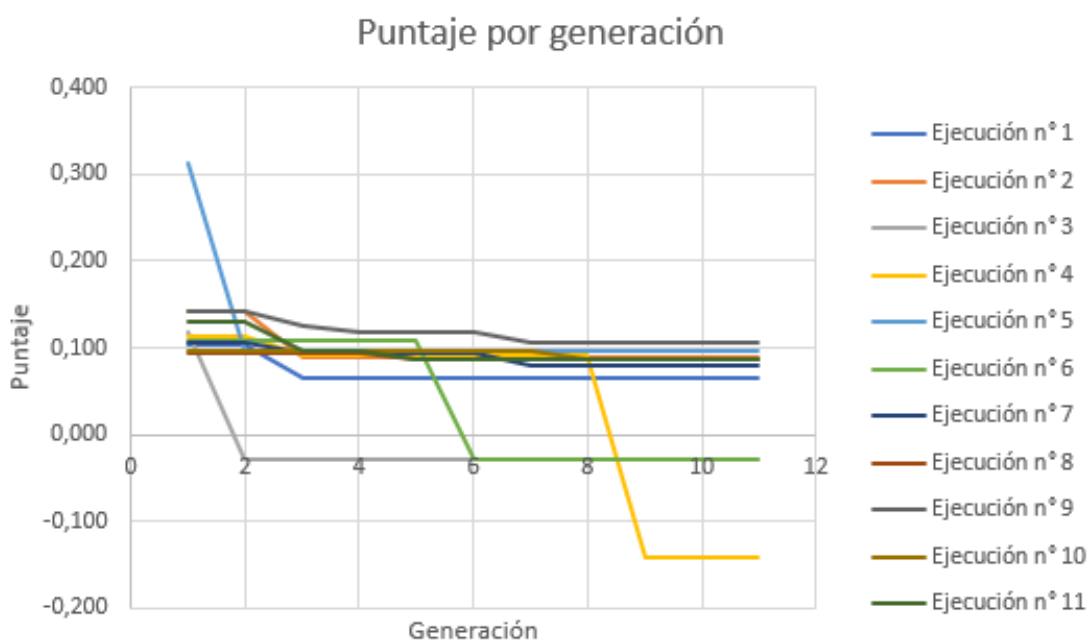


Figura 5.4: Convergencia algoritmo de búsqueda local iterada en prueba n° 1

Fuente: Elaboración propia (2021).

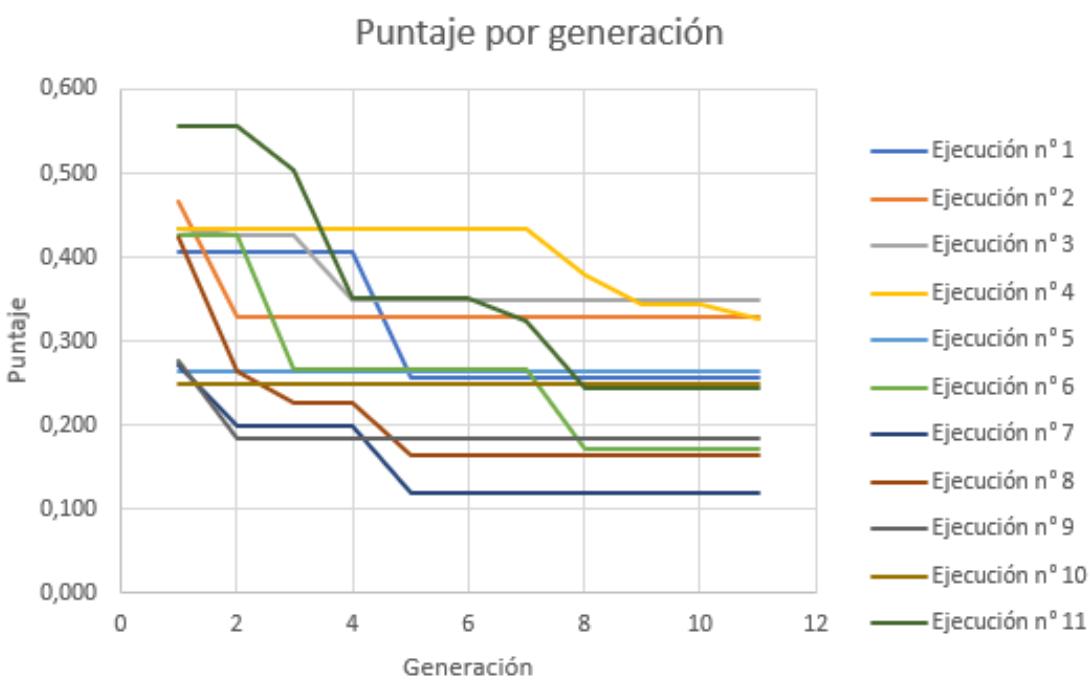


Figura 5.5: Convergencia algoritmo de búsqueda local iterada en prueba n° 2

Fuente: Elaboración propia (2021).

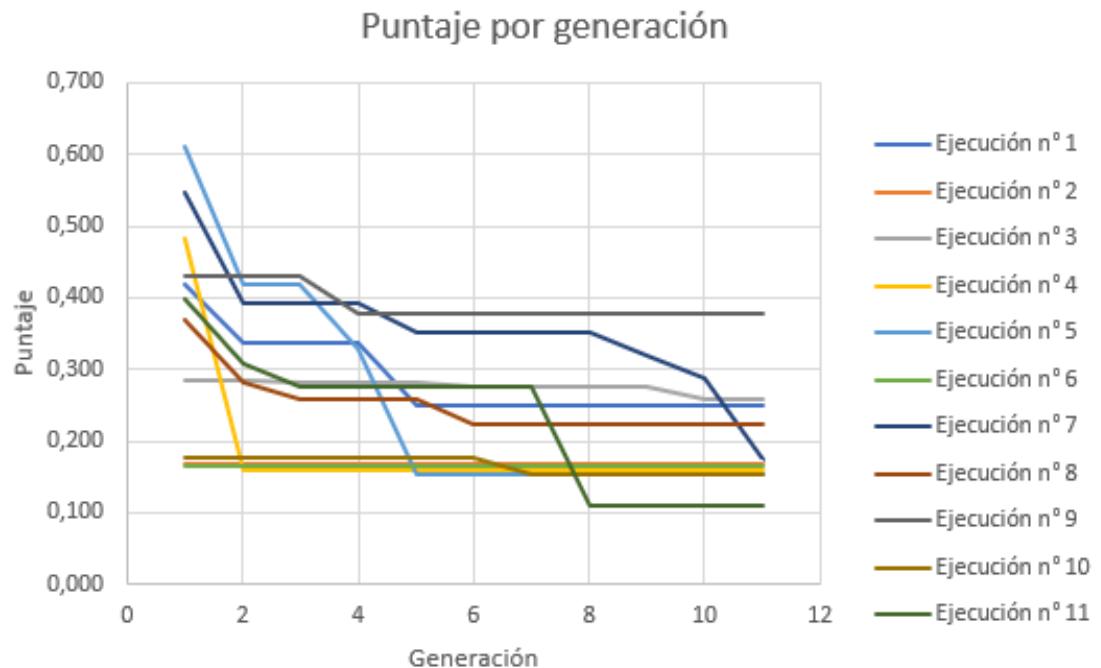


Figura 5.6: Convergencia algoritmo de búsqueda local iterada en prueba n° 3

Fuente: Elaboración propia (2021).

5.2.3 Comparación con órdenes de construcción conocidos

En esta sección se realizarán comparaciones entre los órdenes de construcción obtenidos por el algoritmo de búsqueda local iterada y órdenes de construcción publicados en la página web Spawning Tool, los cuales son elegidos por la comunidad mediante un sistema de votación.

5.2.3.1 Obtener 5 unidades de Zealot

Para esta comparación se consideró el orden de construcción llamado *PvP - DT Rush - Zealot / Archon all-in (VOD) (PvP All-In)* de Zuka (2018). Se requiere obtener 5 unidades de *Zealots* en el menor tiempo posible. Los resultados del orden de construcción se ven reflejados en la Tabla 5.8. Cabe mencionar que la unidad *Probe* no se consideran en el conteo de unidades ya que los órdenes de construcción públicos asumen que se están generando *Probes* de manera constante y eficiente. Además el orden de construcción publicado no especifica cuantos *Probes* se tienen en cada momento.

Tabla 5.8: Resultados orden de construcción conocido para obtener Zealots

Tiempo(mm:ss)	Zealots	Unidades	Edificios	Tecnologías	Entidades Totales
06:30	5	16	21	1	38

Fuente: Elaboración propia, 2021

El mejor orden de construcción obtenido en la prueba n° 1 para la obtención de 5 Zealot es el de la ejecución n° 4. Los resultados se ven reflejados en la tabla 5.9.

Tabla 5.9: Resultados orden de construcción sugerido para obtener Zealots

Tiempo(mm:ss)	Zealots	Unidades	Edificios	Tecnologías	Entidades Totales
05:53	8	8	26	3	37

Fuente: Elaboración propia, 2021

5.2.3.2 Obtener 5 unidades de Dark Templar

Para esta comparación se consideró el orden de construcción llamado *MaxPax's PvT Easy MMR Macro DT Opener (PvT Economic)* de ESChamp (2021). Se requiere obtener 5 unidades de *Dark Templar* en el menor tiempo posible. Los resultados del orden de construcción se ven reflejados en la tabla 5.10.

Tabla 5.10: Resultados orden de construcción conocido para obtener Dark Templars

Tiempo(mm:ss)	Dark Templars	Unidades	Edificios	Tecnologías	Entidades Totales
10:15	5	37	29	4	70

Fuente: Elaboración propia, 2021

El mejor orden de construcción obtenido en la prueba n° 2 para la obtención de 5 Dark Templar es el de la ejecución n° 7. Los resultados se ven reflejados en la tabla 5.11.

Tabla 5.11: Resultados orden de construcción sugerido para obtener Dark Templars

Tiempo(mm:ss)	Dark Templars	Unidades	Edificios	Tecnologías	Entidades Totales
10:03	6	28	55	8	91

Fuente: Elaboración propia, 2021

5.2.3.3 Obtener 5 unidades de Phoenix

Para este experimento se utilizó el orden de construcción llamado *Trap's Cyber First Phoenix/Colossus (PvT Economic)* de Gemini_19 (2021). Los resultados obtenidos utilizando el orden de construcción se encuentran en la tabla 5.12.

Tabla 5.12: Resultados orden de construcción sugerido para obtener Phoenix

Tiempo(mm:ss)	Phoenix	Unidades	Edificios	Tecnologías	Entidades Totales
05:10	5	7	15	1	23

Fuente: Elaboración propia, 2021

El mejor resultado entregado en por el algoritmo fue el de la prueba n° 3, ejecución n° 11. Los resultados se ven reflejados en la tabla 5.13.

Tabla 5.13: Resultados orden de construcción sugerido para obtener Phoenix

Tiempo(mm:ss)	Phoenix	Unidades	Edificios	Tecnologías	Entidades Totales
05:30	6	15	25	4	44

Fuente: Elaboración propia, 2021

5.2.4 Aplicación en juego

En la tabla 5.14 se pueden observar los resultados de aplicar el mejor orden de construcción de la primera prueba en donde se pedía obtener 5 unidades de Zealot, en la tabla 5.15 se aplica el orden de construcción para obtener 5 unidades de Dark Templar y en la tabla 5.16 se aplica el orden de construcción para obtener 5 unidades de Phoenix. Recordar que es el autor quien lleva a cabo estas ejecuciones y no es un jugador profesional, por lo que los tiempos podrían diferir mucho del tiempo de la solución entregada por el algoritmo. Esta sección debe ser considerada solo para dar contexto sobre el uso del algoritmo y las dificultades que un jugador podría encontrar al aplicar un orden de construcción.

Tabla 5.14: Resultados orden de construcción sugerido para obtener Zealots

Tiempo(mm:ss)	Zealots	Unidades	Edificios	Tecnologías	Entidades Totales
06:25	8	8	26	3	37

Fuente: Elaboración propia, 2021

Tabla 5.15: Resultados orden de construcción sugerido para obtener Dark Templars

Tiempo(mm:ss)	Dark Templars	Unidades	Edificios	Tecnologías	Entidades Totales
12:24	6	28	55	8	91

Fuente: Elaboración propia, 2021

Tabla 5.16: Resultados orden de construcción sugerido para obtener Phoenix

Tiempo(mm:ss)	Phoenix	Unidades	Edificios	Tecnologías	Entidades Totales
07:10	6	15	25	4	44

Fuente: Elaboración propia, 2021

CAPÍTULO 6. ANÁLISIS DE RESULTADOS

En esta sección se analizarán los datos obtenidos, verificando si la metaheurística tuvo mejor o peor desempeño que el algoritmo goloso por si solo. También se analizará el rendimiento de las soluciones en comparación con los órdenes de construcción conocidos en términos de tiempo y de cantidad de entidades construidas.

6.1 RENDIMIENTO EN PRUEBAS

6.1.1 Prueba n° 1: Obtener 5 Zealots

Se comenzará analizando los resultados del test de Wilcoxon. Para la primera prueba se obtuvo un valor $p = 0,1450$ con un nivel de significancia de $\alpha = 0,05$, lo cuál indica que no existe evidencia suficiente para rechazar la hipótesis nula, por lo tanto no existen diferencias significativas entre los resultados del algoritmo goloso y de la metaheurística. Esto, como se verá mas adelante, puede ser debido a la baja dificultad de la solicitud, ya que obtener Zealots es relativamente sencillo dentro del juego y el algoritmo goloso no tuvo problemas para encontrar buenas soluciones.

Con respecto a la precisión de la metaheurística, se obtuvo para esta prueba una media de puntaje de 0,0448, una mediana de $0,0857 \pm 0,0461$ y una desviación estándar de 0,078. El algoritmo goloso tuvo una media de puntaje de 0,0008, una mediana de $0,1242 \pm 0,1135$ y una desviación estándar de 0,1920. Esto indica que, si bien el algoritmo goloso tuvo en promedio un puntaje más bajo, el algoritmo de búsqueda local iterada fue más preciso, entregando resultados con menos dispersión.

En conclusión, en la prueba n° 1, la metaheurística no fue capaz de vencer al algoritmo goloso en términos de puntaje. Ahora bien, como se mencionó anteriormente, obtener Zealots tiene una dificultad baja. Todavía falta averiguar si la metaheurística tiene mejor desempeño con tareas más complicadas.

6.1.2 Prueba n° 2: Obtener 5 Dark Templars

En la segunda prueba, se obtuvo un valor $p = 0,0054$ con un nivel de significancia de $\alpha = 0,05$, lo cual permite concluir que existe evidencia suficiente para rechazar la hipótesis nula y que, en esta prueba, existen diferencias significativas entre los resultados del algoritmo goloso y el algoritmo de búsqueda local iterada. De esta manera, se puede concluir que la metaheurística obtuvo significativamente mejores resultados.

El algoritmo de búsqueda local iterada obtuvo una media de 0,2416, con una mediana de $0,2494 \pm 0,04429$ y una desviación estándar de 0,0750. El algoritmo goloso, por su parte, obtuvo una media de 0,4735, con una mediana de $0,4794 \pm 0,0422$ y una desviación estándar de 0,0714. En este caso, ambos algoritmos tuvieron una dispersión de resultados similar, sin embargo la búsqueda local iterada logró encontrar resultados significativamente más bajos.

En conclusión, para la segunda prueba, la metaheruística fue significativamente mejor que el algoritmo goloso, como lo demuestra el test de Wilcoxon y la media de puntajes. Esta prueba era la más difícil de las tres, debido a que la unidad Dark Templar tiene muchos prerequisitos en el árbol de tecnologías.

6.1.3 Prueba n° 3: Obtener 5 Phoenix

Para la última prueba, en el test de Wilcoxon se obtuvo un valor de $p = 0,0492$ con un nivel de significancia de $\alpha = 0,05$. Por lo tanto se concluye que existe evidencia para rechazar la hipótesis nula y que, por ende, existen diferencias significativas entre los mejores resultados del algoritmo goloso y del de búsqueda local iterada. Así, la metaheurística permitió encontrar una mejor solución que el algoritmo goloso.

El algoritmo de búsqueda local iterada obtuvo una media de 0,1995, con una mediana de $0,1678 \pm 0,0437$ y una desviación estándar de 0,0740. El algoritmo goloso obtuvo una media de 0,3541, con una mediana de $0,3904 \pm 0,0560$ y una desviación estándar de 0,0943. En esta ocasión el algoritmo goloso no solo obtuvo una media de puntaje mayor, sino que también tuvo mayor dispersión en los resultados de cada ejecución.

En conclusión, en la prueba n° 3, se tiene que la metaheurística tuvo mejor rendimiento en términos de precisión y encontrando buenos resultados con puntajes más bajos. La dificultad de esta prueba se encontraba entre la primera y la segunda prueba, por lo que se podría concluir que el algoritmo goloso tiene mejores resultados en solicitudes de menor dificultad como por ejemplo Zealots, Probes o edificios como Gateway o Forge. En cambio, la

metaheurística permite encontrar buenos resultados en cualquier nivel de dificultad.

6.2 RENDIMIENTO CONTRA ÓRDENES DE CONSTRUCCIÓN CONOCIDOS

6.2.1 Prueba n° 1: Obtener 5 Zealots

Con respecto a las comparaciones con los ordenes de construcción conocidos con los ordenes de construcción sugeridos por el algoritmo, se tiene que para la primera prueba la mejor solución entregada por la metaheurística mejoró en 37 segundos al orden de construcción conocido. Logró construir 8 Zealots, en vez de los 5 requeridos, y que son 3 más de los obtenidos en el orden de construcción conocido. También construyó 5 edificios más y 2 tecnologías extra. En general, el algoritmo entregó un resultado satisfactorio, cumpliendo el objetivo de superar al orden de construcción conocido.

6.2.2 Prueba n° 2: Obtener 5 Dark Templars

En la segunda prueba se puede apreciar que la solución entregada por el algoritmo entregó un tiempo menor al ya conocido por la comunidad demorando 12 segundos menos. Obtuvo 6 unidades de Dark Templar, en vez de las 5 obtenidas por el orden conocido. Entrenó 9 unidades menos, pero construyó 26 edificios más y 4 tecnologías extra. La solución entregada por el algoritmo logró vencer al orden de construcción en términos de tiempo y de cantidad de entidades.

6.2.3 Prueba n° 3: Obtener 5 Phoenix

Siguiendo con la tercera prueba, el orden de construcción obtenido tuvo peor desempeño en términos de tiempo, demorando 20 segundos más que el orden de construcción conocido. Sin embargo, la solución entregada por el algoritmo permitió construir 8 unidades, 15 edificios y 3 tecnologías extra. Dando un total de 44 entidades construidas versus las 23 obtenidas en el orden de construcción conocido. En este caso, la solución obtenida no superó al algoritmo en términos de tiempo, pero aún así se debe considerar como una buena solución, ya que se

está obteniendo una mayor cantidad de entidades creadas y eso le podría servir de referencia los jugadores.

6.3 APLICACIÓN EN EL JUEGO

Finalmente, en la aplicación en el juego, se pudo observar que seguir el orden de construcción depende de muchos factores como por ejemplo la habilidad y la experiencia del jugador, la velocidad de movimiento de las unidades y la capacidad de agrupar edificios de manera eficiente. Si alguno de estos aspectos falla, entonces aplicar el orden de construcción se podrá llevar a cabo, pero con retrasos. La mejor solución entregada por el algoritmo para construir 5 Zealots indica que se necesita un tiempo mínimo de 353 segundos, pero en la práctica eso se tradujo en 385 segundos; y lo mismo ocurre con los órdenes de construcción conocidos, ya que si no se es un jugador experimentado, entonces aplicar un orden de construcción conlleva retrasos. Hay que recordar que estos órdenes de construcción conocidos fueron creados por jugadores profesionales.

6.3.1 Dificultades

Algunas de las dificultades que existieron al momento de ejecutar los órdenes de construcción fueron:

- Dificultad para seguir el ritmo del orden de construcción. A veces el orden de construcción indicaba un segundo para construir un edificio, pero la unidad constructora se demoraba más en llegar. También, en determinadas ocasiones, se pedían realizar muchas acciones en pocos segundos, pero se carecen de las habilidades necesarias para llevarlas a cabo tan rápido.
- Falta de recursos en determinados momentos para construir una entidad. Esto pasaba porque se construían *Assimilators*, pero no se ponían a los trabajadores a minar gas vespeno ya que no lo indicaba el orden de construcción. El algoritmo implementado asume que cada vez que existe un *Assimilator* entonces se le asignaran Probes de manera inmediata. A esto también hay que sumar las limitaciones que tiene el algoritmo de recolección de recursos mencionadas en la sección 4.1.3.1.

- Olvidar aplicar la habilidad *Chrono Boost* que permite reducir el tiempo de construcción de unidades a la mitad, lo cuál aumentó considerablemente el tiempo final.
- Falta de espacio para construir. Los edificios pueden ser construidos en las áreas circundantes a un *Pylon*. A veces no se utilizaba bien el espacio para construir edificios lo cuál ocasionaba una demora para elegir una buena ubicación.

CAPÍTULO 7. CONCLUSIONES

En la presente memoria se exploró la posibilidad de optimizar el factor tiempo en órdenes de construcción para el juego Starcraft II, específicamente para la raza *Protoss*. Los resultados obtenidos indican que es posible obtener buenos órdenes de construcción en términos de tiempo mediante la implementación de búsqueda local iterada, aunque estos son imprecisos debido principalmente a las limitaciones de factores como el movimiento de las unidades, la recolección de recursos con tasa variable, la experiencia del jugador y su velocidad de respuesta. Todos estos factores implican un aumento en el tiempo y por lo tanto, las soluciones entregadas por el sistema de software desarrollado para este proyecto, no siempre pueden ser desarrolladas en el tiempo especificado de cada actividad. Aún así, los tiempos obtenidos al aplicar las soluciones se pueden acercar bastante a otros órdenes de construcción que utilizan jugadores profesionales e incluso mejorarlos.

Se modeló el problema basándose en el problema de optimización *Resource-constrained project scheduling problem* o *RCPSP*. Este problema catalogado como NP-Hard busca obtener la mejor combinación de actividades para finalizar un proyecto en el menor tiempo posible. Para eso considera restricciones de precedencia y restricciones de recursos. Starcraft II es justamente eso, un árbol de tecnologías que presenta las restricciones de precedencia y los órdenes de construcción que representan las actividades del proyecto, con costos asociados que utilizan recursos escasos.

Por otra parte, se logró definir la estructura de datos para el árbol de tecnologías y para el orden de construcción. Así mismo, se logró modelar el problema integrando la metaheurística de búsqueda local iterada para poder ser utilizada con los parámetros de la raza *Protoss*. También, se desarrolló un sistema de software que implementa el algoritmo y lo ejecuta con los parámetros ingresados por el usuario y se evaluaron los resultados mediante la comparación de los órdenes de construcción conocidos con los sugeridos por el algoritmo. En ese sentido, todos los objetivos específicos se cumplieron satisfactoriamente y servirán para sentar las bases de futuros proyectos que requieran modelar algún problema similar o incluso proyectos que busquen encontrar soluciones óptimas de la raza *Protoss*.

El objetivo principal de este proyecto era optimizar lo más posible el tiempo mediante la utilización de la metaheurística de solución única conocida como búsqueda local iterada, lo cuál se logró hacer como quedó demostrado en la sección 5.2.3 en donde se comparan órdenes de construcción conocidos con órdenes de construcción generados por el algoritmo, los cuales mostraron tener una capacidad similar o mejor en términos de tiempo de ejecución y de cantidad de entidades totales construidas, por lo que se puede decir que el algoritmo encuentra buenas soluciones. También se realizó la comparación entre la metaheurística y el algoritmo goloso

que utiliza en sus iteraciones, demostrando que una búsqueda local iterada obtiene resultados significativamente mejores que una búsqueda local por si sola al momento de obtener soluciones en solicitudes de alta dificultad.

Finalmente, respondiendo a la pregunta de hipótesis planteada. ¿Es posible encontrar buenos órdenes de construcción en términos de tiempo utilizando la metaheurística de búsqueda local iterada, en el juego Starcraft II con la raza Protoss? La respuesta es si. Pues, como quedó demostrado en la sección 5.2.3, el algoritmo superó en 2 ocasiones a los órdenes de construcción conocidos.

7.1 TRABAJO A FUTURO

Se deben realizar pruebas con jugadores de distintos rangos de experiencia. Un jugador profesional o de mayor nivel puede mejorar considerablemente los tiempos obtenidos en la sección 5.2.4 y podría dar a conocer que cambios podrían hacerse para mejorar el algoritmo. También se podría realizar una serie de pruebas entre jugadores que permita ver si el algoritmo mejora la tasa de victorias en las partidas. Ya que, como quedó expuesto en la sección de resultados, en las pruebas n° 2 y n° 3 se obtuvieron más entidades. Por lo que, a mayor cantidad de entidades, es posible que aumente la probabilidad de victoria del jugador.

Por otro lado, implementar un algoritmo que permita simular la recolección de recursos de mejor forma podría mejorar considerablemente el tiempo obtenido en las soluciones que se entregan ya que, como se mencionó en la sección 1.4.3, para efectos de este proyecto no hay forma de saber con precisión cuantos trabajadores se encuentran recolectando en un campo de minerales o de vespene. Esto es, determinar en qué momento existen 1, 2 o 3 trabajadores en un campo de minerales para poder determinar la tasa de extracción. Solucionando este problema se realizarían acciones en tiempos más realistas, ya que actualmente hay ocasiones en que el orden de construcción pide construir una entidad, pero el jugador no lo puede construir porque no hay recursos suficientes debido a que la tasa de extracción, al ser variable, puede ser ligeramente más lenta en el juego que en el algoritmo.

Finalmente, el algoritmo puede ser mejorado implementando una capa de decisión que permita escoger qué unidad es la siguiente en el orden de construcción. Actualmente se escoge de manera aleatoria el siguiente nodo del árbol de tecnologías a construir, por lo que si en vez de ser aleatorio, se decide por ejemplo construir una unidad que no exista, o bien un prerequisito de la entidad objetivo, entonces los tiempos de las soluciones se verían bastante reducidos e incluso el poder del ejército del jugador se vería incrementado, ya que esta capa de decisión le podría dar prioridad a unidades, en vez de edificios o tecnologías. Es más, el algoritmo

actual suele escoger varias veces el mismo nodo, cuya adición no implica una ventaja contra el rival. Por ejemplo, no tiene mucho sentido construir varias veces el edificio *Forge* cuando ya todas las tecnologías han sido investigadas.

El algoritmo también puede ser mejorado si en cada segundo de decisión se pudiera realizar más de una acción. Actualmente el software solo ejecuta una acción por segundo, pero en la realidad los jugadores realizan cientos de acciones por minuto, lo cuál se traduce en tiempos significativamente más cortos.

GLOSARIO

- Árbol de tecnologías: corresponde a las tecnologías, edificios y unidades que posee cada raza.
- Dark Templar: unidad terrestre de la raza Protoss en el juego Starcraft 2.
- Gas vespeno: recurso del juego que se obtiene mediante la construcción de una refinería en los géiseres de gas y mediante la asignación de trabajadores que recolectan dicho recurso.
- Inteligencia Artificial (IA): en el contexto de los juegos de estrategia en tiempo real, es un programa o algoritmo que realiza acciones asemejando un jugador real. En el caso de Starcraft suele ser un programa que puede jugar siguiendo una serie de órdenes de construcción pre establecidos.
- Metaheurísticas: es un método heurístico para resolver un tipo de problema de computación general. Se aplican a problemas que no tienen una solución satisfactoria conocida o cuando no se pueda implementar un algoritmo que dé con una solución óptima.
- Mineral: recurso obtenido por unidades trabajadoras de cada raza.
- Mule: unidad Terran que puede extraer recursos sin tener que entrenar un trabajador ni gastar minerales.
- NP-Hard: categoría de problemas de decisión para los cuales no existe un algoritmo de complejidad polinomial capaz de resolverlos y son a la vez tan o más difíciles que el problema más difícil de la categoría NP (Nondeterministic polynomial time).
- Orden de construcción: secuencia de construcción de edificios y/o unidades.
- Phoenix: unidad aérea de la raza Protoss en el juego Starcraft 2.
- Pilón: edificio básico que genera suministros de la raza Protoss.
- Portal: edificio que permite entrenar infantería en la raza Protoss.
- Protoss: raza de alienígenas inteligentes de la saga de videojuegos Starcraft.
- Reactor: estructura que se puede anexar a edificios Terran. Permite entrenar a 2 unidades al mismo tiempo.
- Real-Time Strategy (RTS): género de videojuegos de estrategia en tiempo real.
- Starcraft: videojuego de estrategia en tiempo real desarrollado por Blizzard Entertainment y puesto a la venta en marzo de 1998.
- Starcraft 2: secuela del videojuego Starcraft, puesto a la venta en julio de 2010 y desarrollado por Blizzard Entertainment.
- Sonda: unidad constructora y recolectora de la raza Protoss.
- Suministros: recurso del juego que se obtiene construyendo depósitos de suministros (Terran), Amos supremos (Zerg) o pilones (Protoss).
- Tecnolab: estructura que se puede anexar a edificios Terran. Permite investigar tecnologías.
- Tecnología: en el juego, las tecnologías son mejoras que se obtienen comprando con recursos. Estas pueden mejorar unidades y edificios.
- Terran: raza de humanos de la saga de videojuegos Starcraft.

- UAlbertaBot: inteligencia artificial diseñada para el juego Starcraft y Starcraft: Brood War. Escrita en C++ e integrado con BWAPI: The Brood War API.
- Unidad: nombre que se le da a la infantería, a los trabajadores o a los médicos del juego.
- Zealot: unidad de infantería de la raza Protoss.
- Zerg: raza de alienígenas tipo enjambre de la saga de videojuegos Starcraft.
- Zángano: unidad de infantería de la raza Zerg.

REFERENCIAS BIBLIOGRÁFICAS

- Artigues, C., Demassey, S., & Neron, E. (2007). *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*. ISTE.
- Ballestín, F., & Trautmann, N. (2008). An iterated-local-search heuristic for the resource-constrained weighted earliness-tardiness project scheduling problem. *International Journal of Production Research*, 46(22), 6231–6249.
URL <https://doi.org/10.1080/00207540701420560>
- C. León, C. R. E. S., G. Miranda (2015). El método científico en la era de los ordenadores. *Andorra la Vella*.
URL http://bioinfo.uib.es/~joemiro/aenui/procJenui/Jen2015/le_elme.pdf
- Chapman, J. (2017). Esports: A guide to competitive video gaming. Accessed: 2021-12-02.
URL <https://www.toptal.com/finance/market-research-analysts/esports>
- Churchill, D., Buro, M., & Kelly, R. (2019). Robust continuous build-order optimization in starcraft. vol. 2019-August. Cited By 3.
URL <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85073099050&doi=10.1109%2fCIG.2019.8848109&partnerID=40&md5=9fd9537e2096471a5aef39b5c1b8ea81>
- ESChamp (2021). Maxpax's pvt easy mmr macro dt opener (pvt economic). Accessed: 2021-08-20.
URL <https://lotv.spawningtool.com/build/156025/>
- Garro, J. (2018). Lluvia de millones en los esports, estos son los 10 juegos que más dinero han repartido en 2018.
URL <https://esports.xataka.com/ligas-y-competiciones-de-esports/lluvia-millones-esports-estos-10-juegos-que-dinero-han-repartido-2018>
- Gemini_19 (2021). Trap's cyber first phoenix/colossus (pvt economic). Accessed: 2021-08-20.
URL <https://lotv.spawningtool.com/build/149968/>
- Justesen, N., & Risi, S. (2017). Continual online evolutionary planning for in-game build order adaptation in starcraft. (pp. 187–194). Cited By 24.
URL <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85026384177&doi=10.114S%2f3071178.3071210&partnerID=40&md5=941fb0ff90250f600005a4aa01840c5c>
- Kuo, I. (2012). Schools are using starcraft 2 as serious education tools.
URL <https://www.gamification.co/2012/12/04/schools-using-starcraft-2-as-education-tools/>
- Köstler, H., & Gmeiner, B. (2013). A multi-objective genetic algorithm for build order optimization in starcraft ii. *KI - Künstliche Intelligenz*, 27(3), 221–233. Cited By 13.
URL <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84974838144&doi=10.1007%2fs13218-013-0263-2&partnerID=40&md5=4b7d91fa68bb55bd165873507e176f96>
- Liquipedia (2021a). Game speed. Accessed: 2021-07-20.
URL https://liquipedia.net/starcraft2/Game_Speed
- Liquipedia (2021b). Protoss tech tree (legacy of the void). Accessed: 2021-09-20.
URL [https://liquipedia.net/starcraft2/Protoss_Tech_Tree_\(Legacy_of_the_Void\)](https://liquipedia.net/starcraft2/Protoss_Tech_Tree_(Legacy_of_the_Void))
- Liquipedia (2021c). Resources. Accessed: 2021-08-20.
URL <https://liquipedia.net/starcraft2/Resources>

- López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3, 43–58.
URL <https://www.sciencedirect.com/science/article/pii/S2214716015300270>
- López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., & Stützle, T. (2020). The irace package: User guide.
URL <https://cran.r-project.org/web/packages/irace/vignettes/irace-package.pdf>
- Manu Sharma, J. S. A. I. C. I. A. R., Michael Holmes (2007). Transfer learning in real-time strategy games using hybrid cbr/rl. *Georgia Institute of Technology*.
URL <https://www.aaai.org/Papers/IJCAI/2007/IJCAI07-168.pdf>
- Suárez, O. (2011). Una aproximación a la heuristica y metaheuristicas. *Universidad Antonio Nariño*.
URL <https://core.ac.uk/download/pdf/236383515.pdf>
- Takino, H., & Hoki, K. (2015). Human-like build-order management in starcraft to win against specific opponent's strategies. (pp. 97–102). Cited By 0.
URL <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84962732307&doi=10.1109%2fACIT-CSI.2015.25&partnerID=40&md5=1c2e13eb1cc3f1ddd4a7cf776001d8>
- Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*. Wiley Publishing.
- Viglietta, G. (2012). Gaming is a hard job, but someone has to do it! *Carleton University*.
URL <https://arxiv.org/abs/1201.4995>
- Volz, V., Preuss, M., & Bonde, M. (2019). Towards embodied starcraft ii winner prediction. *Communications in Computer and Information Science*, 1017, 3–22. Cited By 0.
URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85069181899&doi=10.1007%2f978-3-030-24337-1_1&partnerID=40&md5=c7deaf43bc5741c94948bcb41ba8fa7c
- Wang, L., Zeng, Y., Chen, B., Pan, Y., & Cao, L. (2020). Team recommendation using order-based fuzzy integral and nsga-ii in starcraft. *IEEE Access*, 8, 59559–59570. Cited By 0.
URL <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85083448098&doi=10.1109%2fACCESS.2020.2982647&partnerID=40&md5=aac861859c66ebe191ec05e9b2b2ce89>
- Wang, P., Zeng, Y., Chen, B., & Cao, L. (2019). A data-driven approach to solve a production constrained build-order optimization problem. vol. 2019-July, (pp. 2692–2697). Cited By 0.
URL <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85074418687&doi=10.23919%2fChiCC.2019.8866045&partnerID=40&md5=2329fb6a7633fad311cc52f44e0794f2>
- Zuka (2018). Pvp - dt rush - zealot / archon all-in (vod) (pvp all-in). Accessed: 2021-11-20.
URL <https://lotv.spawningtool.com/build/77158/>