Grado en Ingeniería Aeroespacial
Vehículos Aeroespaciales

Algoritmos para la exploración óptima de árboles con drones de batería limitada

Autor: José Moya Carrasco
Tutor: José Miguel Díaz Báñez

Dep. Matemática Aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo de Fin de Grado
Grado en Ingeniería Aeroespacial
Vehículos Aeroespaciales

Algoritmos para la exploración óptima de árboles con drones de batería limitada

Autor:
José Moya Carrasco

Tutor:
José Miguel Díaz Báñez
Catedrático de Universidad

Dep. de Matemática Aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo Fin de Grado: Algoritmos para la exploración óptima de árboles con drones de batería limitada

Autor: José Moya Carrasco
Tutor: José Miguel Díaz Báñez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

El secretario del Tribunal

Fecha:

A mi madre

Que siempre cree en mí

Agradecimientos

A mi madre, por todo el empeño que ha tenido en que me esfuerze y sea capaz de dar lo máximo de mi, a mi novia por apoyarme en los momentos difíciles y a mi tuto, D. José Miguel Díaz por enseñarme las bases de la heurística y la optimización, así como dedicarme su tiempo y entendimiento para orientarme en este proyecto, sin él no lo habría logrado.

José Moya Carrasco

Sevilla, 2020

Resumen

En este proyecto se presenta un estudio sobre diferentes problemas que surgen para cubrir un grafo tipo árbol sobre el que se desplazaran un equipo de drones que han de visitar todos los nodos. El objetivo es hacerlo de la manera más eficiente posible teniendo en cuenta la limitación de la batería, lo que supone que cada dron puede ejecutar un número limitado de movimientos. Debido a la complejidad del problema, se proponen heurísticos con buen rendimiento para aproximar la solución óptima.

Índice

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICOS

Notación

- ***T*** Árbol, relacionado con árboles gráficos defindos como $T = (V, E)$
- ***r*** Raíz del árbol donde ***r*** ϵ V.
- ***h*** La profundidad o altura (*height*), la denotaremos $h(T)$. Se define como la máxima distancia entre cualquiera de los nodos y la raíz r
- ***p*** Tiempo de vida de la batería que posee cada drone. Relacionado con el número de pasos que pueden dar los drones.
- **Inmersion**. Secuencia de movimientos que debe seguir un dron desde que sale de la raíz del árbol hasta que vuelve a ella visitando una secuencia de hojas. La denotaremos como *I*.
- **Hojas** Denominamos hojas a aquellos nodos que están en las terminaciones del árbol, esto es, no tienen hijos
- **Cobertura** Conjunto de inmersiones que garantizan la visita a todas las hojas.
- $\subseteq$ Contenido o igual
- $>$ mayor que
- $<$ menor que
- **Min** mínimo
- **Max** máximo
- $\in$ que pertenece a

1 INTRODUCIÓN

"The amount of effort you put in is the amount of Resultados you end up with".

- Catherine Pulsifer -

En las últimas décadas, los avances tecnológicos, la disminución de los precios de la informática y el acceso a la información para todo el público ha cambiado la percepción de la humanidad y nos hemos trasladado a una nueva era de conocimiento científico. Estos signos de progreso, combinados con las diferentes redes de conocimiento y trasmisión de la información, como Internet o los medios de información, están permitiendo una difusión mucho más rápida de los desarrollos tecnólogicos logrados. Sin embargo, con cada nuevo avance, aparece un nuevo límite, cada vez más exigente. Dichos límites están en las áreas de seguridad, capacidad, velocidad, demoras, confiabilidad y precisión. Detrás de la cara visible de esta revolución tecnológica-informática, los investigadores de todo el mundo están utilizando las capacidades computacionales disponibles a su alcance para desarrollar nuevos métodos y algoritmos. La mayoría de ellos se basan en grandes cálculos numéricos para resolver los desafiantes problemas propuestos. Esos métodos pueden descomponerse en varios subcampos, cada uno de los cuales cubre áreas importantes de investigación, como diseño, la visualización, simulaciones y optimización. Todas estas áreas de estudio tienen un amplio ámbito de investigación, sin embargo, el último punto, la optimización, será precisamente el principal problema que se desarrollará y estudiará en este proyecto.

La tecnología aeroespacial está destacando en los últimos años por un destacado desarrollo de vehículos aéreos no tripulados. Esos vehículos se están muy valorados principalmente porque son altamente maniobrables, especialmente por su excelente desempeño de trabajos cooperativos y por tener, en general, costes más bajos que los presentados por los vehículos tripulados. Otro valor muy importante para motivar el uso de drones es que con este tipo de vehículos es posible abordar tareas con mayor riesgo para los pilotos involucrados en el control de dichos vehículos, lo que brinda la posibilidad de operar en entornos hostiles, condiciones de baja visibilidad o incluso inaccesibles para el ser humano.

Es necesario tener en cuenta que los algoritmos desarrollados para cubir espacios aéreos con drones pueden ser aplicados para drones que no sean vehículos aéreos. En el caso de este estudio, se investiga una situación en la que los vehículos son subterráneos, sin embargo, como ya se ha dicho, el algoritmo a desarrollar podría usarse en múltiples circunstancias en las que también pueden aplicarse a vehículos aéreos.

El uso de robots tiene ventajas en tiempo de ejecución de la tarea definida debido a la ejecución simultánea de las operaciones y la menor tolerancia dimensional con la que operan, entre otras. Estas ventajas hacen de esta tecnología un tema de estudio actual en la industria aeroespacial que busca nuevos avances a medida que evoluciona. De esta forma, estos vehículos se emplean actualmente en tareas como el montaje de estructuras, inspección (nuestro caso) y vigilancia aérea, entre otras muchas aplicaciones.

Un aspecto significativo cuando se trabaja con varios vehículos autónomos es la posible coordinación entre ellos; imaginemos un grupo de personas en un edificio grande y complejo o incluso en un bosque, y de repente se dan cuenta de que un miembro del grupo ha desaparecido. Se necesita coordinar el grupo para visitar todos los lugares del edificio / bosque / donde sea que encuentre a la persona perdida. Nos referiremos a este problema como *el problema de búsqueda garantizada*, donde los buscadores trabajan juntos para recorrer los alrededores y asegurarse de que visiten todos los puntos necesarios para encontrar un objetivo, en caso de existir. Es importante tener en cuenta que cuanto mayor sea el número de buscadores, menor será el tiempo para visitar todos los puntos o nodos.

Este proyecto investiga el problema de una búsqueda garantizada de la manera más eficiente posible con múltiples robots autónomos. Pongámosnos en situación, los militares y el equipo de primera respuesta a menudo necesitan encontrar a los supervivientes perdidos en situaciones de desastre, o incluso las compañías petroleras importantes necesitan revisar sus tuberías desde el interior sin detener sus fábricas. Otro ejemplo relacionado con el problema que aquí planteamos es la búsqueda en equipo en cuevas submarinas, donde el tiempo (oxígeno disponible) está limitado. El uso creciente de drones autónomos para buscar en tales entornos crea la necesidad de desarrollar algoritmos para optimizar estas tareas. Una necesidad fundamental en robótica es tener algoritmos que conviertan las especificaciones de alto nivel de las tareas de los humanos en descripciones de bajo nivel de cómo actuar. Los términos planificación de movimiento y planificación de trayectoria a menudo se utilizan para este tipo de problemas.

Ilustración 1-1.1 Robot subterráneo usado en la exploración de una tubería.

El uso de heurísticos está estrictamente conectado entre el cálculo de aproximaciones a soluciones óptimas y dichas soluciones óptimas. En [1] se define heurístico como:

"Un método heurístico (también llamado algoritmo de aproximación, un procedimiento inexacto o simplemente heurístico) es un conjunto bien definido de pasos para identificar rápidamente una solución de alta calidad para un problema determinado"

De hecho, la resolución de problemas utilizando heurísticos siempre ha sido parte de la forma humana de trabajar. Desde finales de la década de 1960, se está desarrollando un conjunto de nuevas tendencias de optimización heurísticas. Esos métodos se basan en su mayoría en procesos estocásticos que delinean procedimientos generales de optimización, generalmente designados como metaheurísticos.

1.1 Trabajos relacionados

El problema estudiado en este trabajo forma parte del proyecto "Algorithms for autonomous navigation of underground systems" que se realizó en colaboración con la empresa SPT (Stockhom Precision Tools) y se encuentra en el marco de los proyectos dirigidos por el prof. Díaz-Báñez:

- Combinatorics of Networks and Computation. (CONNECT) (H2020-734922)
- GALGO: Diseño de Algoritmos Geométricos para Problemas de la Ingeniería (MTM2016-76272-R)

Los problemas de exploración y gestión del auto-piloto para robots en entornos desconocidos están siendo estudiados recientemente en numerosos artículos, como por ejemplo en [1]. Uno de los más recientes es la exploración que se describe en [2], donde desarrollan un algoritmo de exploración para k robots situados también en la raíz de un árbol de n nodos, demostrando que el tiempo de exploración es de orden $O\left(\frac{k}{\log(k)}\right)$ mayor que el tiempo de exporación óptimo en caso de que se conozca la distribución del árbol de manera previa. En estos estudios la geometría del terreno se modela como grafos, asumiendo que los robots se mueven a través de las ramas. En [1], [3], [4] y [5] se estudian la exploración de árboles no dirigidos en los que los robots recorren sus aristas de manera unidireccinal. En otros estudios, se imponen restricciones en el número de movimientos de los robots, como por ejemplo en [6] y [7], donde se supone que un robot debe recorrer un entorno volviendo cada cierto tiempo a la posición de inicio para recargar. El entorno es modelado como un gráfico no dirigido que inicialmente es no conocido por el robot y se presenta en [6] un algoritmo para recorrer todos los nodos y ejes del gráfico $G = (V, E)$ en el que se atraviesasn como mucho del orden de $O\left(E + V^{1+O(1)}\right)$ ejes. En otros trabajos, como [8], lo que se limita es la distancia máxima al origen, suponiendo por ejemplo una cuerda, demostrando que la exploración puede realizarse en un tiempo de orden $O\ (e\ )$, siendo e el número de ejes que tiene el gráfico. Otro caso distinto sería la exploración descrita en [7], [8], [10], [11], [12], que analizan y estudian las exploraciones de grafos no dirigidos en los que los robots pueden desplazarse en ambas direcciones de los ejes.

En todos los estudios anteriores, excepto en [2], se podría decir que la exploración está planteada desde la situación de un único robot. De hecho, el caso de exploración de gráficos por múltiples robtos de un entorno, suele estudiarse y plantearse para dicho entorno conocido. En [13], se plantean algoritmos de aproximación para exploraciones colectivas de gráficos aleatorios. Los autores de [14] desarrollan algoritmos en árboles con peso en los diferentes ejes y prueban que los problemas son NP-duros incluso para el caso de dos robots. Es en el trabajo [15] reciente donde se ha estudiado el problema de cobertura de un árbol conocido usando un equipo de robots. Prueban que lo sproblemas de optimización son de diferente complejidad para los casos de distancia total recorrida y tiempo máximo investido. La diferencia con nuestro problema es que no consideran batería sino la restricción es que los robots han de encontrarse cada cierto número de pasos. Los problemas y muchas ideas algorítmicas de este proyecto están contenidas en el trabajo:

Bereg, S., Caraballo L.E., Díaz-Báñez J.M. *Efficient Inspection of Underground Galleries Using k Robots with Limited Energy*. En: ROBOT 2017: Third Iberian Robotics Conference. ROBOT (2017) 706-717

1.2 Estructura del trabajo

El desarrollo del proyecto puede ser explicado en las siguientes partes:

- Definición de los problemas: En esta parte se introduce las definiciones de los prpblemas elegidos para su posterior análisis y optimización.
- Propiedades de las soluciones óptimas: En esta parte se explicaran algunas de las propiedades que tienen en común las soluciones óptimas para las definiciones de los problemas aportados. Estas propiedades serán indispensables para la programación de los algoritmos, pues en ellas estarán basados.

- Los algoritmos: Esta parte junto con la siguiente, es la parte principal del proyecto, donde se describen los algoritmos para cada uno de los problemas definidos anteiormente. Se presentan pseudocódigos para su entendimiento.
- Resultados: Est parte muestra ejemplos de resultados obtenidos tras la implementación de los algoritmos en Matlab, comparando los resultados computacionales con las souciones óptimas. Todas las pruebas engloban los resultados del desarrollo del proyecto, por lo que es una parte fundamental del mismo.
- Conclusiones y futuro trabajo: En esta parte se resumen todas las impresiones que ha generado la realización del proyecto, las conclusiones sacadas del desarrollo, las posibles mejoras, modificaciones o implementaciones que peuden realizarse en vistas a otro TFG o investigaciones futuras.

2 DEFINICIÓN DE LOS PROBLEMAS

"Despite all of our technological advances, content creation still requires time, inspiration, and a certain amount of sweat. There aren't any shortcuts. You can't write an algorithm for it. You can't predict it. You can't code it"
Shawn Amos

Supongamos que planeamos una expedición a Marte con la idea de enviar un equipo de robots a este planeta lejano. Los robots está previsto que aterricen a la entrada de un cráter desconocido. Cada robot móvil está provisto de un dispositivo de comunicación inalámbrico y baterías para obtener energía solar. La misión consiste en examinar el cráter durante la noche y llegar a la entrada del cráter para recargar la batería durante el día. Con un sónar se obtiene un mapeo del cráter completo desde la superficie antes del comienzo de la exploración. El objetivo de tal misión es explorar el mapa de dicho cráter minimizando tanto el consumo de energía del equipo como el tiempo total de la misión.

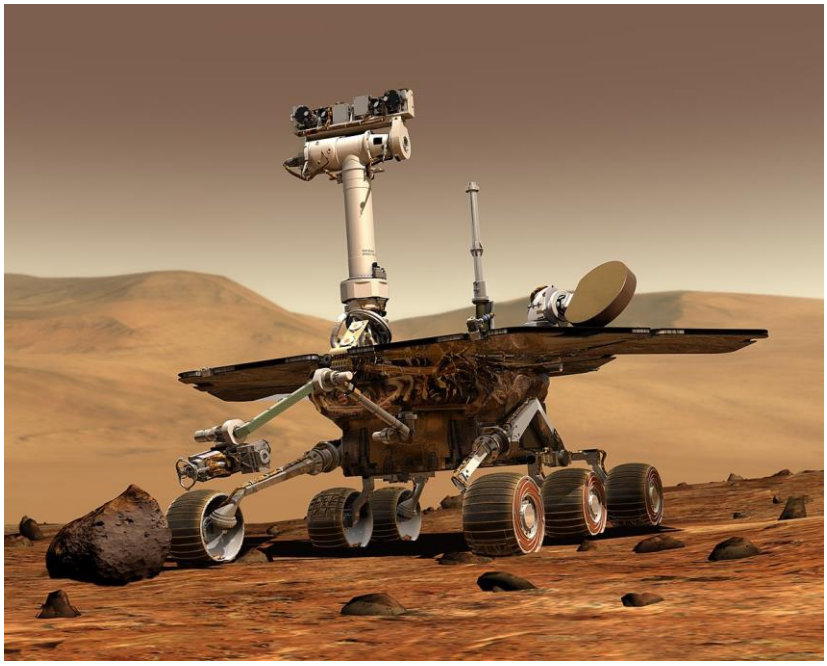

Ilustración 2-1. Robot solar en Marte

Debido a que la señal tarda muchos minutos en llegar a la Tierra la coordinación remota es imposible. En consecuencia, los robots han organizarse entre ellos como grupo y utilizar estrategias locales distribuidas para completar las tareas y regresar a la boca del cráter para transferir sus resultados a la Tierra y recargar las baterías para continuar la misión.

Vamos a estudiar y tratar de optimizar el problema de explorar un entorno por un grupo *k* de robots en el que todas las decisiones, asignación de tareas, rutas seleccionadas y planificación se toman antes del comienzo de la misión y todo ello con el conocimiento previo del mapa del terreno. El problema trata de planificar las inmersiones que se deben realizar estos robots para recorrer la totalidad del entorno.

Las entradas del problema serán entonces: un árbol *T= (V,E)* no ponderado y un equipo de *k* robots para explorar *T* partiendo siempre desde la raíz del árbol que llamaremos *r*. Cada robot se mueve a velocidad constante entre nodos vecinos con un coste unitario y supondremos la raíz del árbol como estación de carga en la que los drones recargan su batería de manera instantánea.

Teniendo esto en cuenta, podríamos decir que la principal restricción es que los robots deben volver a la raíz *r* dando no más de *p* pasos, siendo un paso el equivalente a un movimiento entre nodos vecinos. Esto, nos restringe entonces que para que se pueda llevar a cabo una cobertura total de *T*, la profundidad o altura del árbol debe cumplir: $p \geq 2\ h(T)$

Sea $I = \{I_1, \dots, I_m\}$ un conjunto de m inmersiones que cubren la totalidad de T y siendo $S_i \subseteq I_i$ la asignación de inmersiones para cada robot *i.* El coste en distancia y tiempo para cada S_i se denotará como:

$$d(S_i) = \sum_{I \in S_i} d(I)\ ,\ t(S_i) = \sum_{I \in S_i} t(I)\ .$$

El coste de cada inmersión I será igual al número de pasos que se den para recorrerla.

$$C(S_i) = \sum_{I \in S_i} C(I)\ .$$

2.1 Minimizar la distancia recorrida para *k* robots

TREE − INSPECTION k − MIN − DISTANCE

Entrada:

$un\ árbol\ T = (V, E)\ con\ un\ nodo\ raíz\ r \in V$

$k\ robots$

$p \in \mathbb{N}\ tal\ que\ p \geq 2h(T)$

Solución:

$Conjunto\ de\ inmersiones\ I_i\ que\ cubran\ T\ (\forall I_i \in I\colon d(I_i) \leq p)\ y\ su\ asignacion\ a\ los\ robots\ S_i$

Objetivo:

$minimizar\ \sum_{i=1}^{k} d(S_i)$

Nótese que en el problema k-MIN-DISTANCE, la solución optenida no dependería de las inmersiones asignadas a cada robot, si no de las distancias de dichas inmersiones exclusivamente. Esto significa que para un árbol *T* dado, con un nodo raíz *r,* la solución en distancia óptima de dicho problema permanerá invariante para para cualquier número de robots $k \geq 1$. En consecuencia, debe notarse que minimizar la distancia para 1 drone o para *k* drones es exactamente el mismo problema pues no depende del tiempo y su optimización no requiere trabajar en paralelo.

2.2 Minimizar el tiempo de cobertura para *k* robots

$\boldsymbol{TREE - INSPECTION\ k - MIN - TIME}$

Entrada:

$un\ árbol\ T\ =\ (V, E)\ \ con\ un\ nodo\ raíz\ r \in V$

$k\ robots$

$p\ \in\ \mathbb{N}\ tal\ que\ p \geq 2h(T)$

Solución:

$Conjunto\ de\ inmersiones\ I_i\ \ que\ cubren\ T\ (\forall I_i \in I\colon d(I_i) \leq p)\ y\ su\ asignación\ a\ cada\ robot\ i\colon S_i$

Objetivo:

$minimizar\ \ max\ _{i=1}^{k}\ t(S_i) = \sum_{I \in S_i} t(I)$

Una conclusión a la que se puede llegar tras observar la descripción del prolema es que para $k = 1$ el problema K-MIN-TIME se puede reducir al problema 1-MIN-DISTANCE, mientras que para $k > 1$ se trata de un problema totalmente diferente y a la vez más complejo.

2.3 Minimizar el tiempo de cobertura para *k* robots con inmersiones dadas

Para encontrar una aproximación al problema K-MIN-TIME, para el caso de $k > 1$, plantearemos un problema auxiliar e intermedio. En dicho problema intermedio tendremos de inputs un número de robots y un conjunto de inmersiones que cubran *T*; el objetivo será asignar las inmersiones dadas a los *k* robots para minimizar el tiempo de cobertura.

Este problema existe en la bibliografía y se denomina Problema de Assignación *(Assignment Problem)* y existen múltiples procedimientos heurísticos para asignar las tareas (*inmersiones*) a los recursos (*drones*) de la manera más óptima posible. La asignación de recursos consiste en encontrar una distribución óptima de las tareas a realizar entre un conjunto de sujetos. Para el problema definido, cada robot ejecutará de forma independiente las inmersiones que le sean asignadas de una en una.Varios drones pueden trabajar en sus respectivas inmersiones de manera simultánea y sin interferencias en sus desplazamientos entre ramas. Para obtener el conjunto de inmersiones que constituyen la entrada del problema, se utilizarán las soluciones del problema 1-MIN-DISTANCE, que asegura la cobertura total de *T*.

Vamos a ilustrar con un ejemplo el problema de la asignación de tareas: se consideran tres drones, y se tiene el siguiente conjunto de inmersiones. El problema es asignar esas inmersiones a los tres drones. ¿Cuál sería la forma adecuada de distribuir el trabajo entre los robots? Si cada inmersión tiene la misma longitud, por ejemplo, hay 9 inmersiones, cada una con 100 unidades de distancia, el trabajo es bastante sencillo y solo queda dividir las inmersiones en grupos del mismo tamaño.

100 100 100 ||100 100 100 ||100 100 100

Con esta partición, cada robot recorre 300 unidades, por lo que el tiempo total para cubrir el árbol será el máximo de entre los tres drones, o sea 300.

¿Qué sucede si las inmersiones no son del mismo tamaño? No se puede usar la misma partición, por ejemplo:

100 200 300 ||400 500 600 ||700 800 900

En esta partición, mientras el primer robot ha recorrido 600 unidades, el último ha viajado en 2400. En este caso, la distribución más adecuada para estas inmersiones sería:

100 200 300 400 500 ||600 700 || 800 900

Donde el robot que viaja más es el último, pero solo 1700, y el que viaja menos es el segundo, con 1300. Se puede definir entonces una variable para comprobar si dicha carga ha sido balanceada:

Carga de trabajo balanceada,

$$CTB = t_{max}/t_{min}$$

Viene a significar cómo de bien está distribuida la carga de trabajo entre los drones. En el primero de los ejemplos, el ideal, CTB = 1; mientras que en el segundo CTB = 1700/1300 = 1,30. El objetivo siempre será que CTB tienda a 1.

En general, el problema puede enunciarse así:

TREE − INSPECTION FIXED INMERSIONS − k − MIN − TIME

Entrada:

$un\ árbol\ T = (V, E)\ con\ un\ nodo\ raíz\ r \in V$

$k\ robots$

$p \in \mathbb{N}\ tal\ que\ p \geq 2h(T)$

$Conjunto\ de\ inmersiones\ I_i\ que\ cubren\ T\ (\forall I_i \in I: d(I_i) \leq p)$

Solución:

$La\ asignación\ de\ inmersiones\ a\ cada\ robot\ i: \{S_1, \dots, S_k\}$

Objetivo:

$minimizar\ \ max\ \sum_{i=1}^{k} t(S_i)$

Este problema se usará como auxiliar, para el cual se implementará un heurístico de tal forma que, utilizando la solución de éste, encontraremos una aproximación para el problema de minimizar el tiempo para *k* robots.

3 PROPIEDADES DE LAS SOLUCIONES ÓPTIMAS

"Despite all of our technological advances, content creation still requires time, inspiration, and a certain amount of sweat. There aren't any shortcuts. You can't write an algorithm for it. You can't predict it. You can't code it"
Shawn Amos

Vamos a exponer algunos apuntes y propiedades que serán útiles para el posterior diseño de los algoritmos. Estas propiedades están demostradas en el trabajo [16]. Como ya comentamos, los problemas de las secciones [2.1] y [2.2] son totalmente diferentes. Considerando el árbol de la figura 3.1 y suponiendo la presencia de dos robots con batería 6 o mayor, la solución para el problema [2.1] se muestra en la figura (b) con una única inmersión en un tiempo 6, mientras que para el problema [2.2], la solución óptima sería utilizar los dos robots y que cada uno de ellos realizara una inmersión de 4 unidades de tiempo, siendo el tiempo total de la cobertura igual a 4.

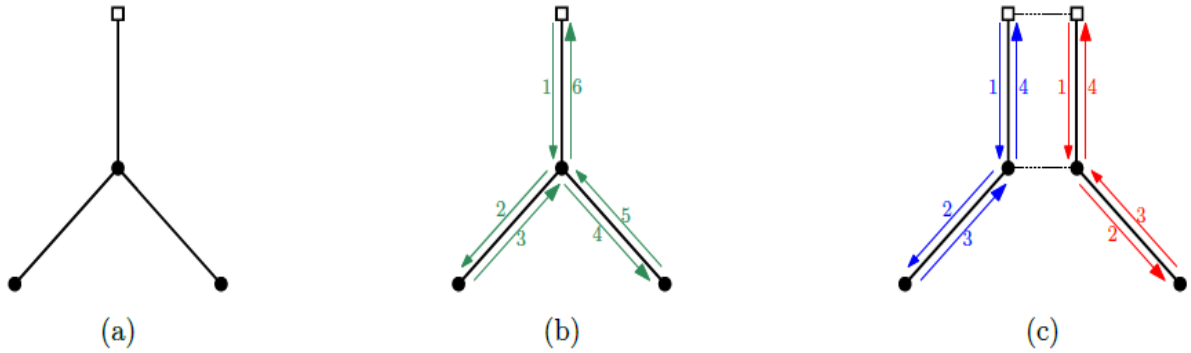

Ilustración 3.1.

(a) Árbol no ponderado. (b) Viaje de un solo agente para cubrir el árbol. (c) Los viajes de dos agentes para cubrir el árbol.

Dado un árbol $T = (V, E)$ con raíz en r, para cada inmersión I_i se determina un nuevo árbol (V_i, E_i) con raíz en r donde $V_i \subseteq V$ es el conjunto de nodos visitados en dicha inmersión I_i y $E_i \subseteq E$ es el conjunto de aristas atravesadas en el árbol durante dicha inmersión.

Asumiendo que el coste en tiempo y distancia es el mismo para una inmersión, vamos a trabajar únicamente con la variable del coste

$$C(S_i) = \sum_{I \in S_i} C(I) .$$

De esta manera vemos que:

$$2|E_i| = C(I_i)$$

Para el desarrollo de las diferentes propiedades, vamos a denotar cada inmersión por los correspondientes subárboles, de la manera:

$$I_i = (V_i, E_i)$$

PROPIEDAD 1:

"Las hojas en cada inmersión I_i de la solución óptima para el problema K-MIN-DISTANCE son hojas del árbol original. Mientras que para el problema de K-MIN-TIME y MIN-INMERSIONS siempre habrá una solución óptima con esta propiedad."

PROPIEDAD 2:

"Cada hoja del árbol T está solamente en una inmersión de la solución ótima para el problema K-MIN-DISTANCE. Para los problemas K-MIN-TIME y MIN-INMERSIONS siempre habrá una solución óptima con esta propiedad."

PROPIEDAD 3:

"Las soluciones de K-MIN-DISTANCE para un mismo árbol pueden tener distinto número de inmersiones".

Podemos poner un ejemplo que ilustra la Propiedad 3 usando el árbol de la figura siguiente, batería $p = 26$.

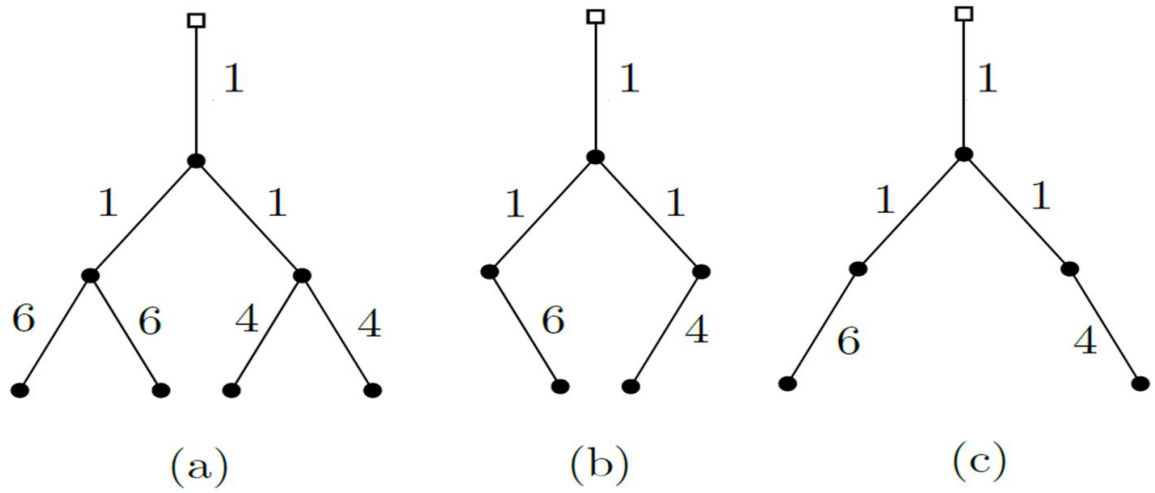

Ilustración 3-2. Árbol ejemplo para K-MIN-distance.

Podemos ver que la solución óptima vendría dada por las dos inmersiones (b) y (c) con un coste cada una de 26 unidades, lo que nos deja un coste total de inmersión de 52 unidades. No obstante, existe una solución para la cual el número de inmersiones es 3, y el coste total es idéntico, como podemos ver en la siguiente figura, para la que las inmersiones (b), (c) y (d).

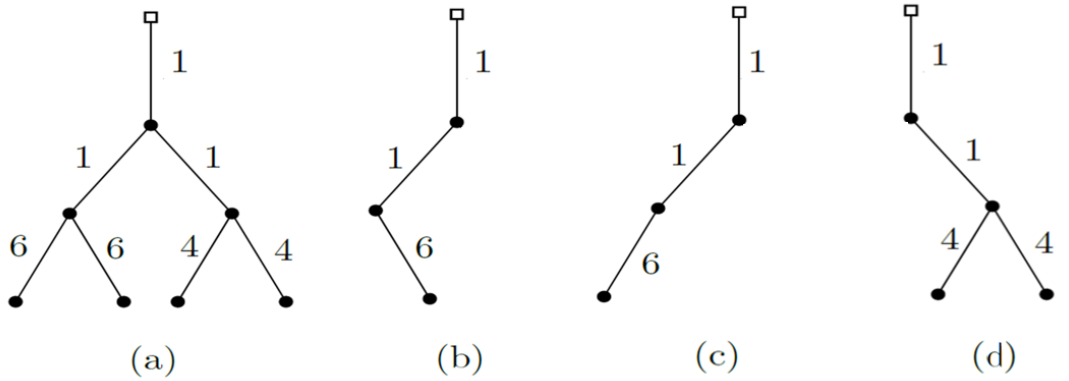

Ilustración 3-3 Soluciones óptimas con 3 inmersiones.

4 LOS ALGORITMOS

"Optimization is the procedure or procedures used to make a system or design as effective of funcional as posible, especially the mathematical techniques involved.

- American Heritage Dictionary-

Una vez definidos los problemas y apuntadas las propiedades encontradas para una solución óptima, se describirán los algorítmicos que proponemos para encontrar estas soluciones. Debido a la complejidad intrínseca de los problemas, trabajaremos con heurísticos y mostraremos su rendimiento. Con idea de realizar experimentos intensivos con muchos árboles aleatorios necesitaremos de un algoritmo para la generación aleatoria de árboles.

Hay que tener en cuenta que existen muchos métodos y algoritmos para generar árboles aleatorios. En nuestro trabajo se ha definieron en todo momento como entrada de los algoritmo el árbol *T*. Por ello se ha desarrollado un algoritmo generador de árboles en base al número de nodos y la restricción de que el nodo raíz fuera el nodo 1.

El desarrollo de dicho algoritmo está basado en que es necesario generar un árbol de n nodos con etiquetas $\{1,2,\ldots,n\}$. Comenzando con los conjuntos:

$$V = \{1\}$$

$$S = \{2, \ldots, n\}$$

$$E = \{\}(empty\ set)$$

Luego, se aplica la siguiente regla: seleccionando aleatoriamente un elemento $v \in V$ y otro $w \in S$, luego se agrega la arista $\{v, w\}$ a E, se elimina w de S y se agrega a V. Repetimos este proceso hasta que S esté vacío. El gráfico resultante es un árbol donde el nodo 1 es la raíz. Este sería el pseudocódigo del método que denotaremos `treeGenerator`.

```
Algorithm 1: Random generation of a tree
Input: integrer n
Output: tree T=(V,E) rooted at r=1

1: V={1};
2: E={};
3: S={2,…,n};
4:     while S is not empty do
5:           v = choose(V);
6:           w = choose(S);
7:           E = E ∪ {{v,w}};
8:           V = V ∪ {w};
9:           S = S \ {w};
10:    end while
11: r = 1;
12: return T = (V,E) rooted at r;
```

4.1 Algoritmo minimizar la distancia recorrida para *k* robots

En esta sección se describe el algoritmo para el problema K-MIN-DIST. Se trata de un algoritmo que busca en cada inmersión el nodo más profundo no visitado y decide la ruta más eficiente después alcanzarlo en su camino de vuelta a la raíz. La idea es que en su vuelta visite todos los nodos posibles con la batería restante tras consumir la necesaria para llegar al más profundo.

Denominaremos a este algoritmo como `DeepFirstThenClosest`

```
Algorithm
Require:
Positions of the n targets, target information (n, position, etc)

Ensure:
Every target is visited.

1: while oj_visited < n
2:     if number of farthest target > 1
3:           Robot goes to one of the one that has the closest nonvisited node
4:     end if
5:
6:     Robot goes to farthest target
7:
8:     while batery > distance to closest node + distance to root
9:           Robot goes to the closest target he can reach on his way to the root
10:    else
11:          Robot goes to the root following the shorter route
12:    end while
13:
14: end while
```

El algoritmo usa la técnica DFS (*Deep first search*). Los robots visitarán en orden de profundidad los nodos para cubrir *T*. La diferencia entre un algoritmo básico DFS y éste es que, tras visitar la hoja más profunda, las siguientes visitas a hojas en su camino a la raíz no tienen por qué ser las sucesivas más profundas, sino aquellas que permitan también volver a recargar baterías.

De este algoritmo se han desarrollado dos versiones durante el estudio del proyecto. En la primera versión del algoritmo, el código no se cuestionaba si tras haber realizado la visita a la hoja más profunda, el robot debería visitar la hoja cercana más profunda o la más profunda en general. Por el contrario, iba seleccionando por cercanía con respecto a la primera visita en la inmersion, de esta manera seleccionaba la siguiente hoja a la que ir. En caso de que hubiera dos hojas a una misma distancia, el algoritmo eligiría la siguiente al azar.

Para la segunda versión, se consideran diferentes aspectos tras la visita del nodo más profundo, como la profundidad a la que se encuentra el siguiente nodo o la distancia a la que está, y en cada una de las visitas siguientes, se elige según esos parámetros.

En la sección de resultados, compararemos la mejora computacional y la proximidad de ambos algoritmos con respecto a la solución óptima.

4.2 Algoritmo para el tiempo mínimo

Para dar una aproximación al problema de minimizar el tiempo de cobertura de árboles con k robots usaremos el problema auxiliar presentado en la Sección (2.3), esto es, calcular el tiempo mínimo con inmersiones dadas, en el que las inmersiones de la entrada hay que asignarlas a los drones.

Uno de los principales problemas en logística es la asignación de recursos. Estos recursos deben ser asignados a múltiples tareas, cada una de ellas valorada de manera diferente, de manera que la productividad sea óptima. Las heurísticas que estudian las diferentes maneras de asignar recursos van desde la asignación de tareas en función de la carga a una determinada decisión sobre el orden de realización de las tareas.

Desde que se descurbrió la potencia de cálculo de los computadores, uno de los problemas más demandados ha sido el de planificar una serie de tareas de forma eficiente. Existendiversas técnicas para resolver estos problemas: se puede plantear algoritmos greedy, de ramificación y poda, de programación dinámica, etc…

En el mundo empresarial, en el que es imprescindible planificar los recursos y distribuir la carga de trabajo, planificar la producción o ajustar los horarios de descanso del personal es una tarea compleja donde se usan algoritmos que, en muchos casos, son heurísiticos sencillos. El caso concreto de repartir una serie de tareas entre un número de individuos es conocido como "task allocation" y el objetivo claro es minimizar el tiempo conjunto de realizar una serie de tareas.

Dicho esto, y tratando de optimizar el tiempo total de cobertura para k-robots, el heurístico propuesto para el problema de minimizar el tiempo es una combinación de heurísticos de los problemas anteriores. La idea es que usemos los heurísticos del problema de minimizar la distancia total recorrida para generar una solución sub-óptima y obteniendo de este un conjunto de inmersiones, tomemos estas como fijas y asignemos de la manera más óptima posible para minimizar el tiempo.

Para la parte de la asignación de las inmersiones a los robots usaremos un heurístico que se basará en la idea de la creación recursiva tipo Greedy de una matriz en la que estén correlacionados las inmersiones con cada robot *k*.

Si consideramos cada una de las inmersiones como una tarea con tiempo de ejecución igual a la duración de la inmersión, los inputs del algoritmo serán el número de robots *k* y el vector I $(I_1, I_2 \dots, I_n)$ que contiene los tiempos de ejecución para cada una de las inmersiones *n* (obtenido previamente). El heurístico buscará en el vector I de mayor a menor tiempo de inmersión e irá asignando al robot que sea capaz de terminar todas las tareas que tenga ya asignadas más la próxima en el menor tiempo posible. El output del algoritmo es una matriz en el que cada columna *k* contiene las tareas asignadas al robot *k*. De esta matriz podríamos obtener el tiempo de ejecución total de cada uno de los robots, así como el tiempo de ejecución total para completar la cobertura completa por los robots.

Incluimos aquí el pseudocódigo del algoritmo de asignación de inmersiones:

```
InmersionAllocation
Input:
k = number of robots
n = number of inmerions
I = vector with dimension n, each position of the vector is the distance of the I_i
inmersion

Require:
Assign n inmersions to k robots

Ensure:
Every inmersion is assigned.

Objetive: minimize → max(sum(I_a))

Output:
Inmersion assigned to k robots I_a{n,k};
```

```
Coverture_time = max(sum(I_a))

I_a = zeros(n,k)

1: while max(I) is not 0 do
2:     [i_time, pos]                    = max(I);
3:     [accumulative_time, robot]       = min(sum(I_a));
4:     I_a(pos,robot)                   = i_time
5:     I(pos) = 0
6: end
```

5 RESULTADOS

The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.

Claude Shannon, 1948

En esta sección se mostrarán los resultados que se obtuvieron con los algoritmos de la sección 4, implementados en Matlab. Además, se comparará el tiempo computacional con respecto a los algoritmos que obtienen soluciones óptimas. Tener en cuenta que todas las implementaciones han sido realizdas en Matlab R2015b (8.6.0.267246) 64-bits en un ordenador con 4Gb de RAM Intel Core i-5.

Como se indicó en secciones previas, los algoritmos propuestos no garantizan obtener unas soluciones óptimas al ser heurísiticos, debido a esto, el propósito de esta sección es estudiar qué tan buena son nuestros algoritmos en comparación con las soluciones óptimas.

5.1 Heurístico para distancia mínima

Ya dijimos que el *input* de treeGenerator era el número de nodos que se desea que tenga el árbol, y que la salida es T y las hojas que tiene dicho árbol. Con el código del generador aleatorio de árboles agregado al principio del código de DeepFirstThenClosest y limitando el número de nodos de T a 13 (para el que se obtiene un tiempo de computación aceptable), se comparará la distancia obtenida por la solución del heurístico contra distancia de la solución óptima.

Para encontrar la distancia óptima usaremos un método exhaustivo, esto quiere decir que calcularemos todas las combinaciones posibles de inmersiones en las que las distancias no excedan el límite de la batería y obtendremos la distancia total de la exploración. Después de eso elegiremos la que tenga la menor distancia de cobertura.

El tiempo computacional para hallar dicha distancia óptima es exponencial ya que todas las combinaciones posibles están en orden $O(2^n)$ donde n es el número de hojas de T y, para un número relativamente grande de hojas, el ordenador se bloquea. Evidentemente, con el heurístico (el tiempo computacional para hallar la solución es de orden lineal con el tamaño del árbol) el número de hojas con las que el ordenador puede calcular una solución aproximada es mucho mayor.

5.1.1 Comparación del coste con respecto a la solución óptima en distancia

Para tener un criterio sobre la eficacia del algoritmo se ejecutará el programa completo con 10.000 repeticiones de árboles aleatorios y solo mostrará en pantalla los resultados donde la solución heurística devuelve un resultado peor en comparación con el óptimo, de esta manera podremos ir estudiando mejoras y diferentes adaptaciones del heurístico para obtener mejores aproximaciones a la solución óptima.

Las salidas del código incluyen la cobertura de la solución heurística, la cobertura de la solución óptima, las distancias de ambas soluciones y la desviación entre ambas distancias. Hay que tener en cuenta que esta desviación siempre ha de estar por encima de 1, siendo 1 solo en los casos en que la heurística ha encontrado la cobertura óptima y por lo tanto su distancia de cobertura total coincide con la óptima.

Como ya se ha dicho en los párrafos anteriores, una de las maneras que se ha seguido para plantear mejoras al algoritmo hasta conseguir el heurístico final es estudiar aquellos casos en los que la solución heurística fuera peor que la óptima.

Ahora se va a proceder a describir un caso en el que la cobertura encontrada no es óptima. El siguiente árbol, con 15 nodos y siete hojas, dado por el algoritmo treeGenerator, es el primer ejemplo que mostramos donde nuestra HeuristicCover no es óptima:

```
Leaves = [4,7,8,9,13,14,15]
```

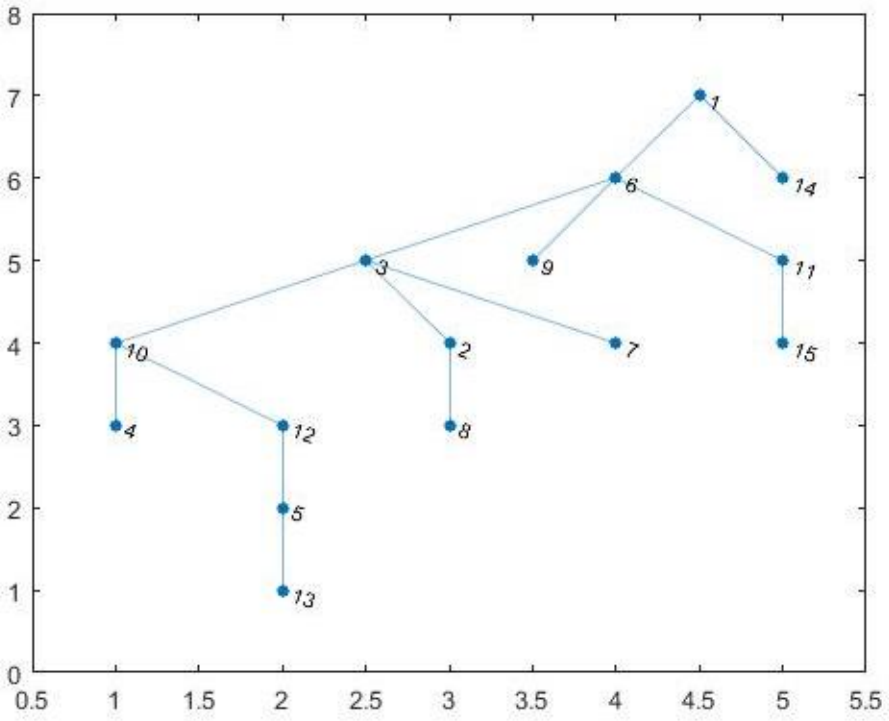

Ilustración 5-1 Árbol aleatorio. Ej 1

Introduciendo este árbol en el programa HeuristicCover, después de unos segundos, el programa muestra esto en pantalla.

```
Heuristic_cover = [1,13,1,4,7,14,1,8,9,1,15,1]
Optimum_cover   = [1,13,1,4,8,1,7,15,9,1,14,1]


Heuristic_distance = 40
Optimal_distance   = 38


Deviation = 1.0526
```

La solución óptima muestra que la forma más eficiente de cubrir este árbol es combinar los nodos 4 y 8 en la misma inmersión. Podemos ver que la heurística elige al azar al nodo número 4 en la segunda inmersión, y teniendo en cuenta que el nodo 7 es el más cercano a él, el robot irá al 7, lo que significa que en una futura inmersión el robot debe visitar el nodo número 8. Durante las siguientes ilustraciones se pueden ver cada uno de los pasos de la cobertura, inmersión a inmersión.

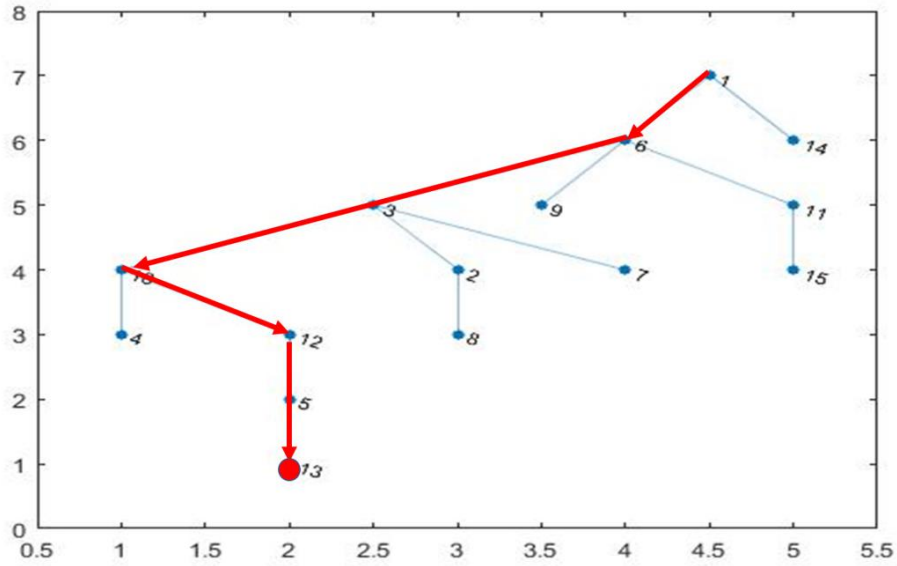

Ilustración 5-2 Cobertura heurística Inmersion 1.

El dron visitará el nodo número 13. Distancia de la inmersión: 12 unidades.

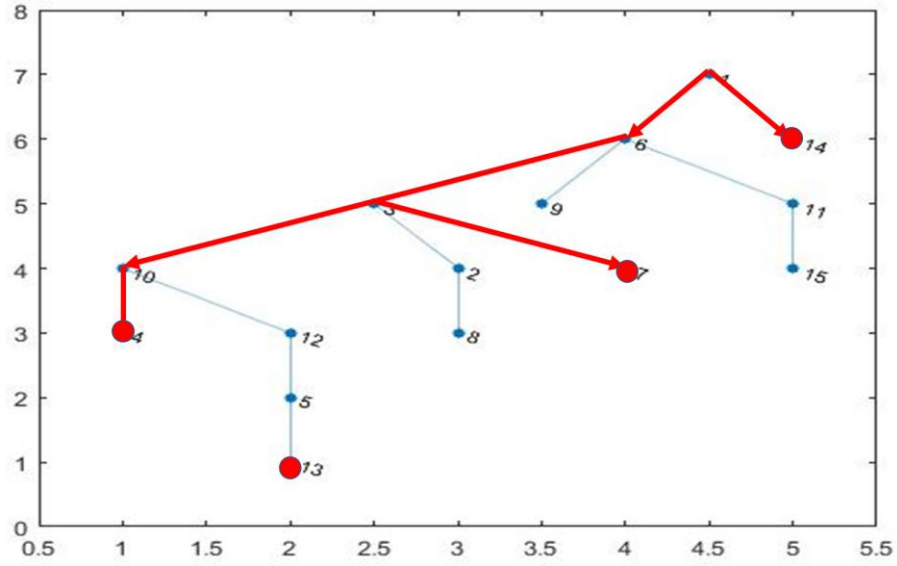

Ilustración 5-3 Cobertura heurística Inmersion 2.

En este viaje, el dron visita los nodos 4, 7 y el nodo 14. El nodo 13, que está resaltado en rojo, ya está visitado. Distancia de inmersión: 12 unidades.

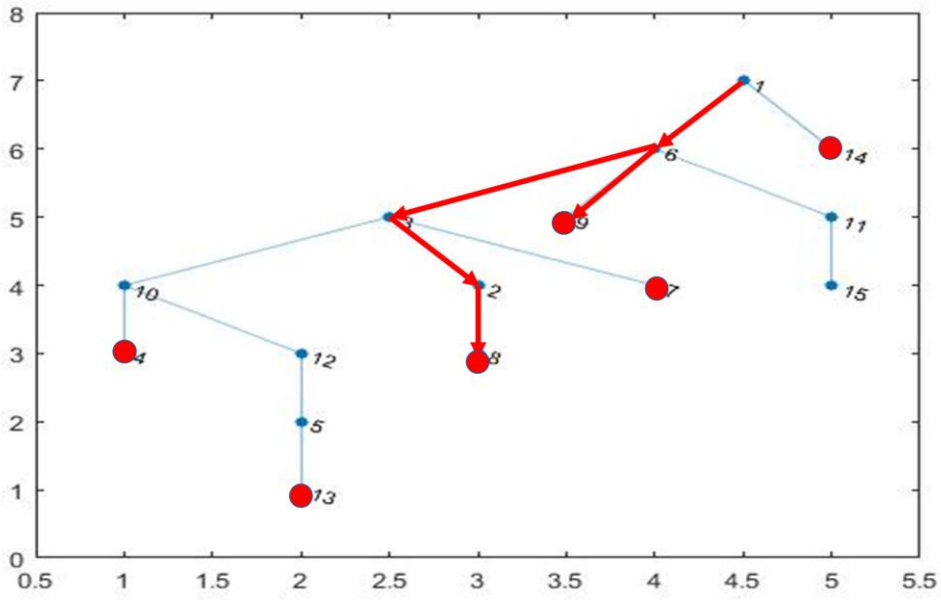

Ilustración 5-4 Cobertura heurística, Inmersion 3.

El dron visitará los nodos 8 y 9, comenzando por el más profundo. La heurística elige el nodo más profundo no visitado, el número 8. En esta inmersión, el robot no puede visitar el nodo número 15 ya que la batería no permite tanta distancia en una inmersión. Distancia de inmersión: 10 unidades.

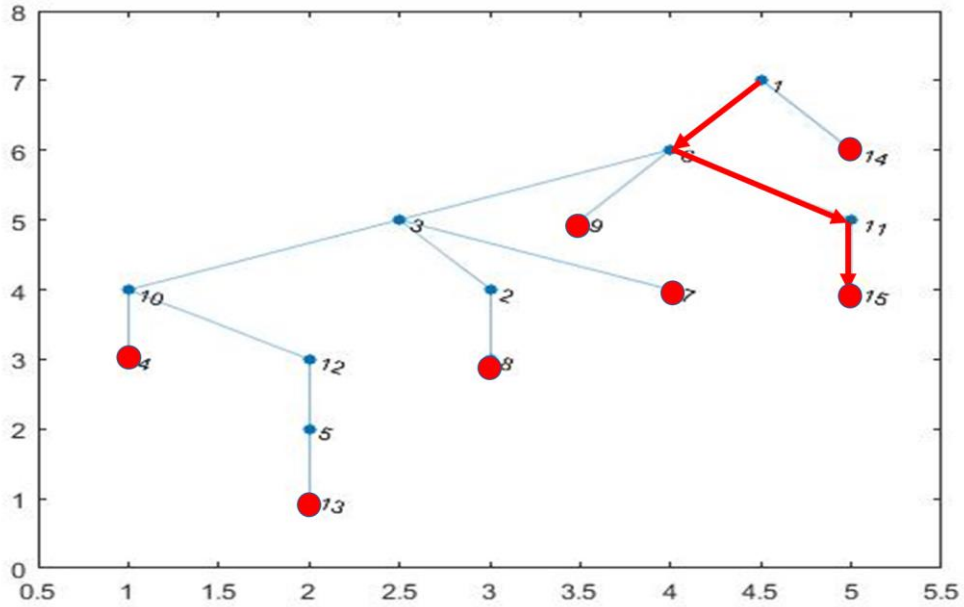

Ilustración 5-5 Cobertura heurística Inmersion 4.

El dron visita el último nodo, que es el nodo número 15. Distancia de la inmersión: 6 unidades.

Ahora, se explicarán las inmersiones de la Cobertura óptima. La primera inmersión es idéntica a la primera de la heurística. Esto es algo que siempre coincide entre la solución heurística y la óptima, ya que la batería es exactamente necesaria para llegar al nodo más profundo y luego llegar a la raíz, una de las inmersiones debe ir directamente al fondo y luego moverse hacia arriba a la raíz.

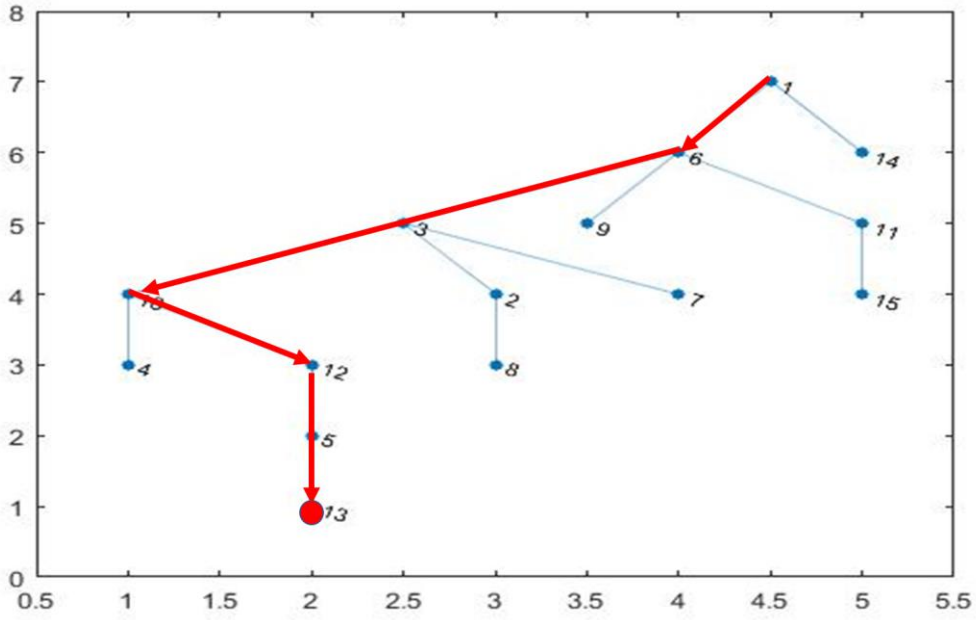

Ilustración 5-6 Cobertura óptima, Inmersion 1.

El robot visitará el nodo número 13. Distancia de la inmersión: 12 unidades.

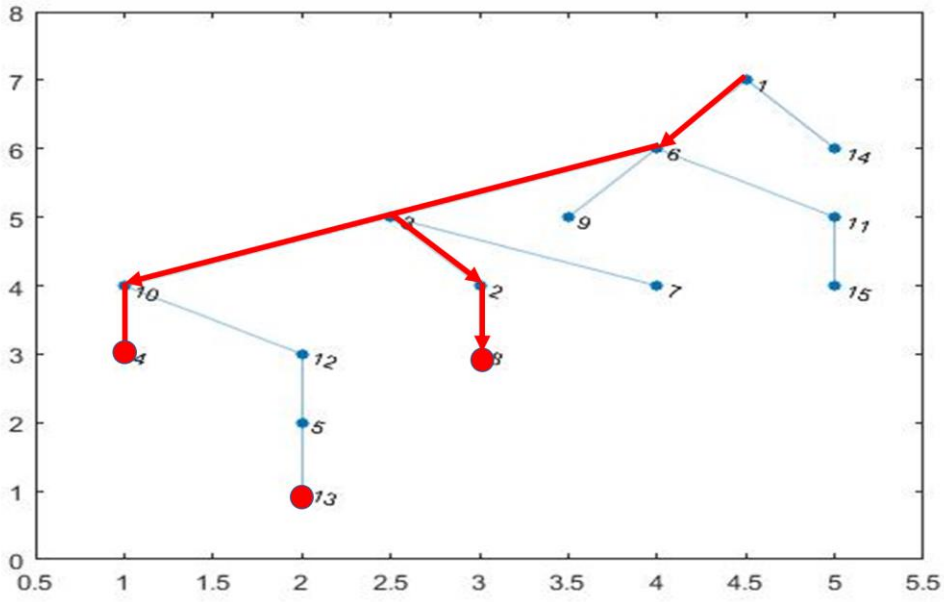

Ilustración 5-7 Cobertura óptima, Inmersion 2.

El dron visitará los nodos 4 y 8. Esta es la primera diferencia entre lo heurístico y lo óptimo. En este caso, incluso si la inmersión tiene la misma distancia que la inmersión heurística 2, los nodos visitados son más profundos que en la heurística. Distancia de inmersión: 12 unidades.

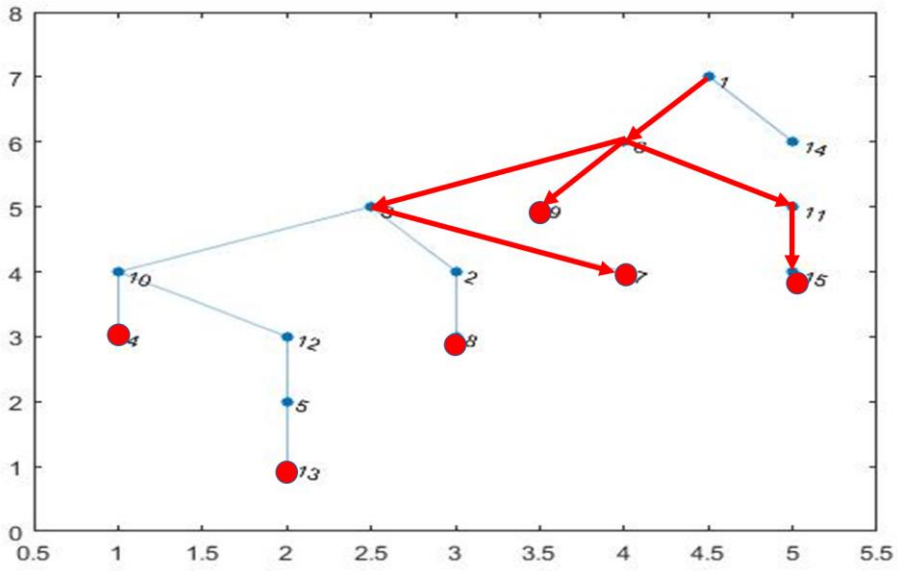

Ilustración 5-8 Cobertura óptima, Inmersion 3.

El dron visitará los nodos 7, 9 y 15. Distancia de inmersión: 12 unidades.

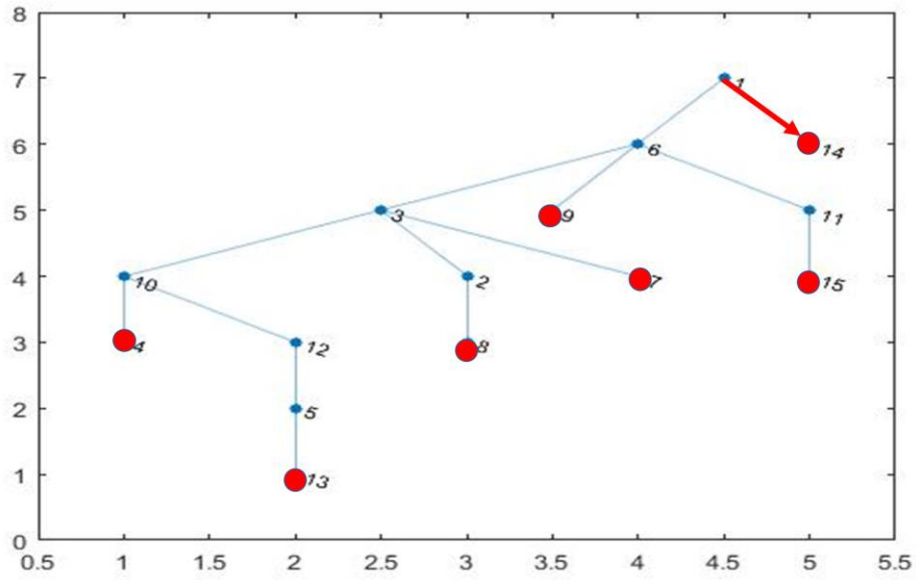

Ilustración 5-9 Cobertura óptima, Inmersion 4.

Último nodo a visitar. Distancia de inmersión: 2 unidades.

Se puede apreciar en este ejemplo que la distancia en la solución óptima es 2 unidades menor que la heurística, incluso el número de inmersiones el mismo. También podría ocurrir que la cantidad de inmersión fuera menor en la solución óptima.

En las tablas que siguen hay un análisis de estas distancias. En la mayoría de los casos, el rango de aproximación no excede los límites del 1.1, pero es cierto que este rango podría llegar a ser enorme; eso depende de la configuración del árbol *T*. Para árboles balanceados, el rango es bastante bajo, pero para árboles extendidos con una rama única conectada a la raíz, si esta rama es lo suficientemente larga, los resultados son considerablemente peores. En la siguiente tabla, la segunda columna es la desviación media para 10.000 intentos en árboles de cantidad de nodos indicados en la primera columna.

Nº Nodos	*Desviacion media H-V2 - solución óptima*	*Desviación máxima*
8	1	1
9	1	1
10	1.0002	1.0909
11	1.0003	1.0909
12	1.0005	1.0909
13	1.0009	1.1429
14	1.0011	1.1765
15	1.0015	1.1000

Tabla 5-1 Desviacion media entre HeuristicoV2 la solución óptima 10.000 ejecuciones.

Como ya dijimos, hubo una versión previa del heursístico (V1); en la siguiente tabla están los resultados de la primera versión del heurístico para árboles con el mismo número de nodos. Con con el paso de V1 a V2 podemos ver que la mejora es bastante sustancial.

Nº Nodes	*Desviacion media H-V1 - solución óptima*	*Worst Deviation*
8	1	1
9	1.0005	1.1111
10	1.0005	1.1111
11	1.0010	1.1000
12	1.0015	1.1667
13	1.0017	1.1429
14	1.0023	1.1765
15	1.0037	1.1667

Tabla 5-2. Desviacion media entre HeuristicoV1 y la solución óptima. 10.000 ejecuciones

Destacamos que la mejora de la versión V2 es considerable, y está alrededor de un 0,12% más cerca de la solución óptima para los árboles estudiados. En la figura siguiente se ilustra el comportamiento de ambas heurísticas al aumentar el tamaño del árbol.

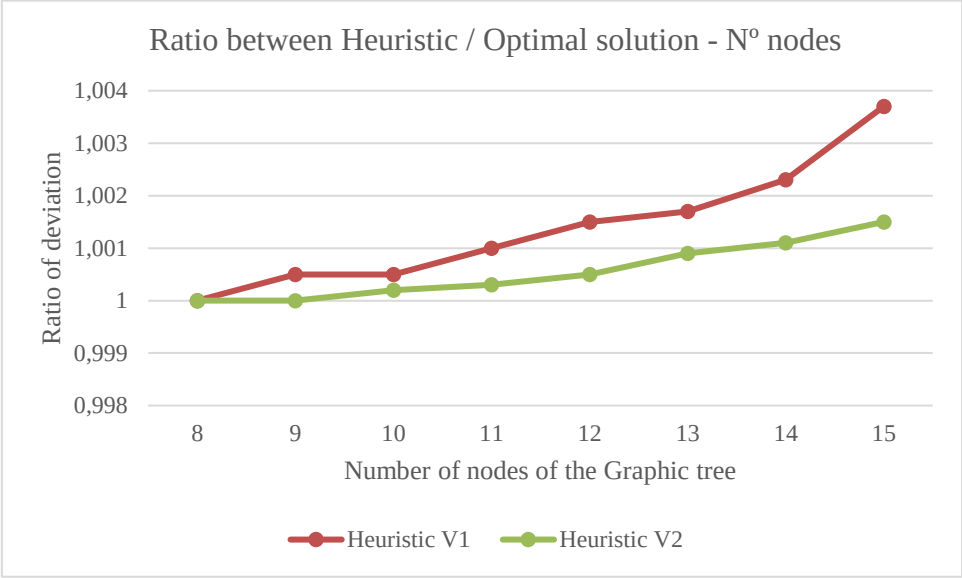

Gráfico 5-1. Ratio between Heuristic / Optimal solution - Nº nodes

5.1.2 Comparación del número total de inmersiones con respecto a la solución óptima

Si comparamos aquí el número de inmersiones y no la distancia total, por ejemplo, en el siguiente ejemplo la solución Heurística tiene una inmersión más que la cobertura óptima.

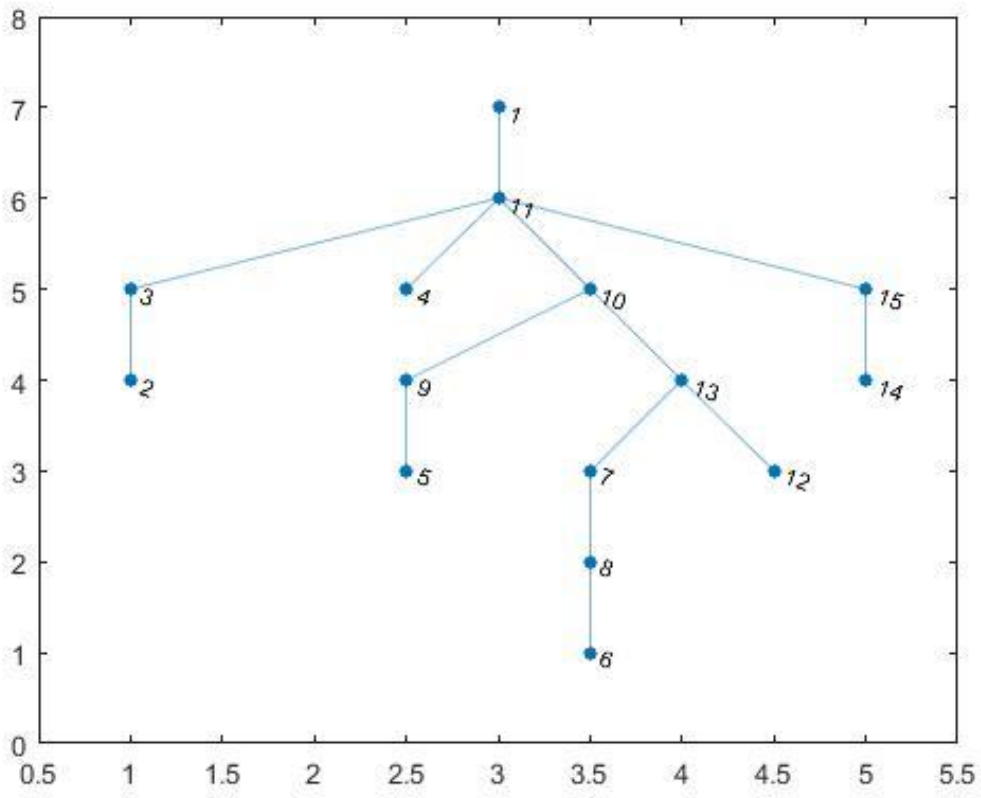

Ilustración 5-10 Árbol aleatorio para estudiar el número de inmersiones.

```
Heuristic_cover = [1,6,1,5,4,1,12,2,1,14,1]
Optimus_cover   = [1,6,1,12,5,1,14,4,2,1]


Heuristic_distance = 40
Optimal_distance   = 36


Distance_Deviation = 1.1111
HeuristicNumberInmersions = 4;
OptimalNumberInmersions   = 3;
```

En la siguiente tabla se muestra cuántas soluciones heurísticas tienen más inmersiones que la solución óptima para la misma cantidad de 10.000 árboles aleatorios. Como se puede ver [Gráfico 5-2] el número aumenta de manera exponencial a partir árboles de tamaño 11 nodos.

Nº Nodes	*Numero de soluciones heurísticas con más inmersiones que la solución optima*
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	17
12	21
13	29
14	40
15	60

Tabla 5-3 Número de soluciones no óptimas.10.000 repeticiones

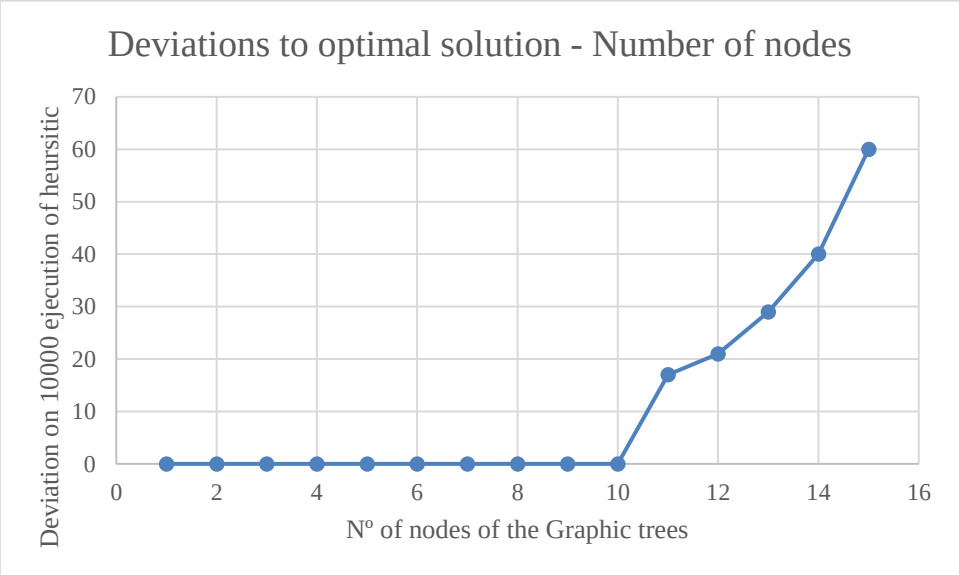

Gráfico 5-2. Soluciones heurísticas con más inmersiones que la óptima.

5.1.3 Comparación del tiempo computacional

Durante esta subsección, el estudio se centrará en el tiempo computacional. Para ese propósito, se ejecutará el siguiente programa que nuevamente genera aleatoriamente árboles con un número dado de nodos, y luego calcula la cobertura con la heurística y la óptima, obteniendo ambos tiempos de cálculo.

Esta captura del código es la parte que se ejecutará 10.000 veces:

```
[G,leaves,dist]    = random_tree(n_nodes) ;
tic;
[HeuristicCover,d_inmersions] = heuristic_cover(G,leaves,dist);
HeuristicTime = HeuristicTime+toc;
tic;
[OptimalCover]    = optimal_cover(G,leaves,dist);
OptimalTime = OptimalTime+toc;
```

Los resultados se registraron en la siguiente tabla y con su gráfico relacionado.

Nº Nodes on T	*Tiempo solución heurística*	*Tiempo solución óptima*
9	2.77	2.80
10	3.54	4.75
11	3.80	9.82
12	4.13	26.91
13	4.61	101.91
14	5.97	289.42
15	22.26	1630.79

Tabla 5-4. Tiempo computacional S.Heurística vs S.Óptima

Y en la siguiente figura, se puede apreciar que el tiempo de computación aumenta de manera exponencial con el número de nodos del árbol, haciendo imposible que nuestro ordenador encuentre la solución óptima para árboles de más de 15 nodos. En cambio, el tiempo para encontrar soluciones en la heurística es prácticamente

lineal, por lo que es razonable encontrar soluciones para árboles mucho más grandes, incluso de más de 1000 nodos.

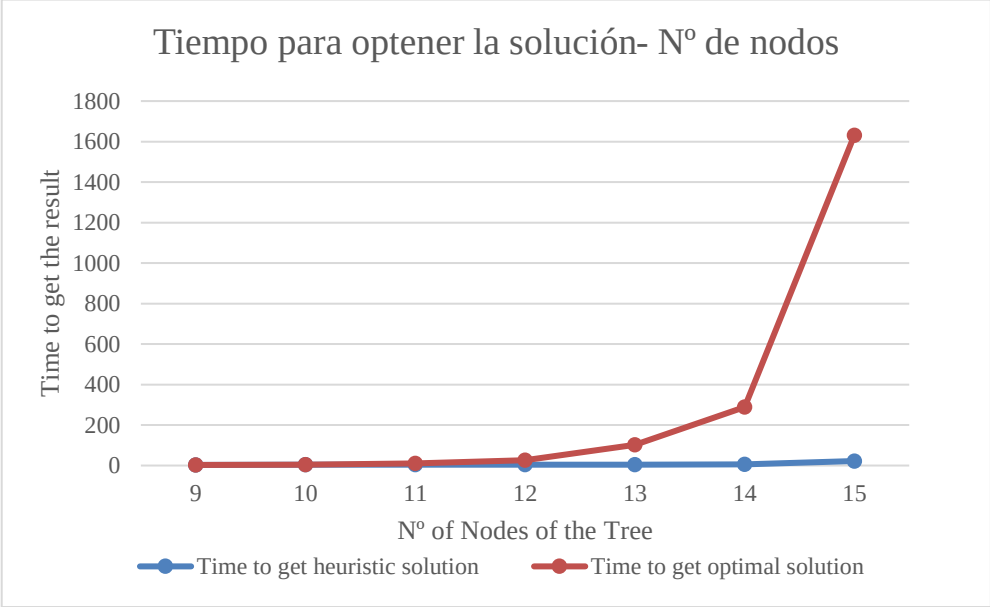

Gráfico 5-3 Tiempo computacional.

5.2 Algoritmo greedy para el problema de asignación de recursos

Recordamos que en el método usado para el problema auxiliar se obtendran las inmersiones con los algoritmos de optimización de la distancia, y se asignarán dichas inmersiones entre el número de drones usando un algoritmo tipo *greedy*.

Recordemos la definición del problema:

TREE − INSPECTION FIXED INMERSIONS − k − MIN − TIME

Entrada:

$un\ árbol\ T = (V, E)\ con\ un\ nodo\ raíz\ r \in V$

$k\ robots$

$p \in \mathbb{N}\ tal\ que\ p \geq 2h(T)$

$Conjunto\ de\ inmersiones\ I_i\ que\ cubren\ T\ (\forall I_i \in I: d(I_i) \leq p)$

Solución:

$La\ asignación\ de\ inmersiones\ a\ cada\ robot\ i: \{S_1, \dots, S_k\}$

Objetivo:

$minimizar\ \ max\ _{i=1}^{k}\ t(S_i)$

Para estudiar el comportamiento de las soluciones heurísticas definimos una métrica comprarando con una solución *ideal* del problema en la cual donde todos los drones invertirían el mismo tiempo. Definimos el concepto de *desviación relativa* como la ratio entre el tiempo de cobertura obtenido por el heurístico y una cobertura en la que todos los drones tardaran lo mismo en recorrer las inmersiones asignadas.

$$Balanced\ solution: O_t = \frac{t_{cover}}{k}$$

$$Relative\ Deviation: O_t = \frac{Heuristic_{time}}{O_t}$$

Se encontró que los resultados de dicha desviación relativa fueron mejores al aumentar el tamaño del árbol. Esto se debe a que en los árboles de gran tamaño hay más inmersiones y se pueden distribuir mejor, o de manera más uniforme, la carga de trabajo, por lo que la desviación relativa tiende a 1.

Veamos un caso concreto: comenzamos con un árbol de 15 nodos, que es el más grande que nuestros ordenadores están capacitados para encontrar la cobertura óptima, y un número de drones igual a tres. Este es el árbol aleatorio que ha calculado nuestro TreeGeneratorHeuristic:

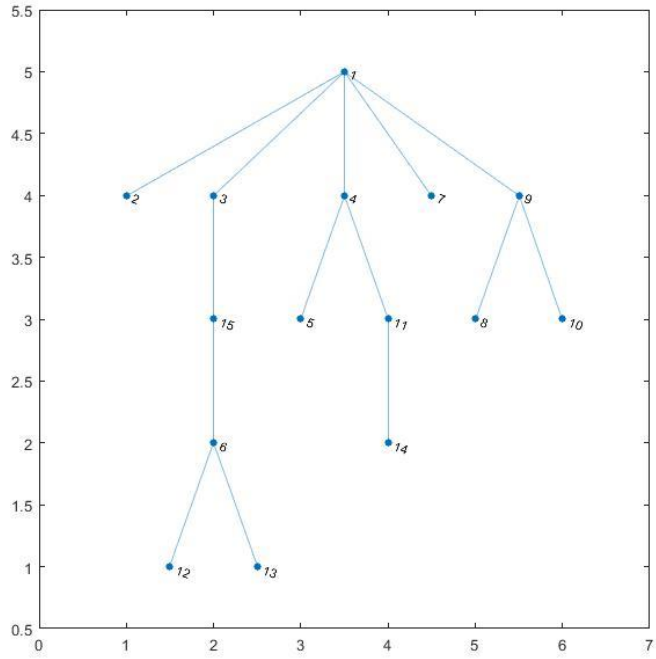

Ilustración 5-11 Árbol aleatorio para estudiar el algoritmo greedy.

Ejecutando el algoritmo para minimizar la distancia recorrida para 1 robot se obtiene la siguiente secuencia de inmersiones:

```
Inmersion 1 = [1,12,1]
Inmersion 2 = [1,13,1]
Inmersion 3 = [1,14,5,1]
Inmersion 4 = [1,8,10,2,1]
Inmersion 5 = [1,7,1]

NumberOfInmersions   = 5
DistanceOfInmersions = [8,8,8,8,2]
DistanceOfCover      = 34
```

Ahora, asignando para $k = 3$ $robots$ de manera greedy, quedarían las inmersiones asignadas como se indica en el siguiente gráfico:

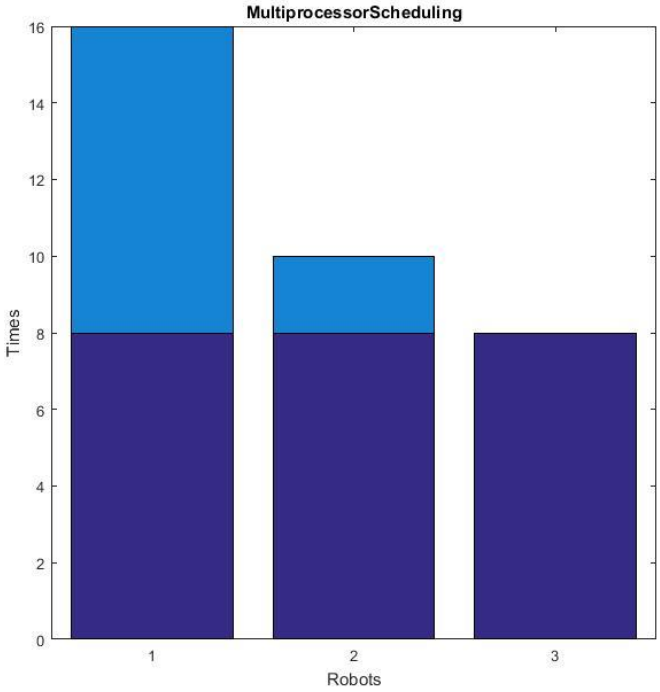

Ilustración 5-12 Asignación de tareas para 3 robots y árbol de 15 nodos.

```
Heuristic_cover = [1,12,1,13,1,14,5,1,8,10,2,1,7,1]
Optimal_cover =   [1,12,1,13,1,14,5,1,8,10,2,1,7,1]


Heuristic_distance = 34
Optimal_distance   = 34

NumberOfInmersions   = 5
DistanceOfInmersions = [8,8,8,8,2]
DistanceOfCover      = 34

TimeToCover = 16
DesviaciónRelativa   = 1.4118
```

Vemos que el tiempo de cobertura total es de 16 unidades. Si calculamos la desviación relativa:

$$Solución\ balanceda = \frac{34}{3} = 11{,}33$$

$$desviación\ relativa = \frac{16}{11{,}33} = 1{,}41$$

Como se muestra en los resultados, la desviación es bastante grande (41%), pero esto no significa que, la Heurística para el problema 3 no sea óptima para este árbol, ya que no es posible distribuir 5 inmersiones de [8,8,8,8,2] de distancia en grupos de 3 y se invierta el mismo tiempo cada uno de los grupos.

En el siguiente ejemplo tomaremos un árbol aleatorio que tenga 1000 nodos y 7 robots. El árbol y las asignaciones dadas por el algoritmo se muestran a continuación:

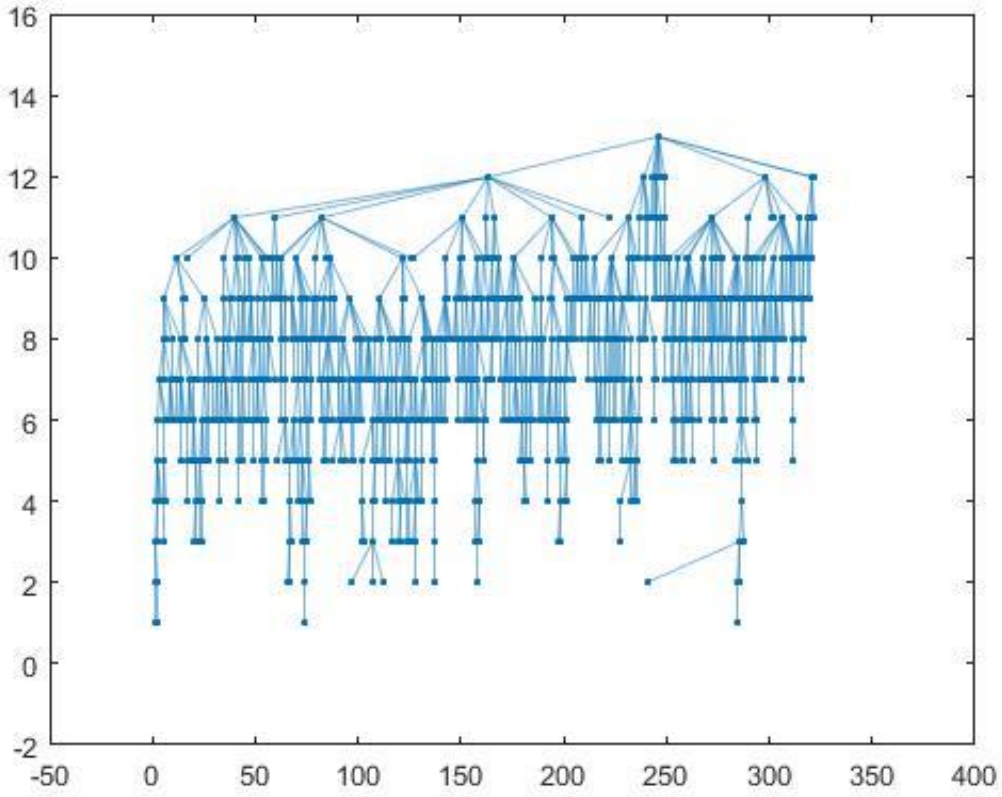

Ilustración 5-13 Árbol aleatorio de 1000 nodos.

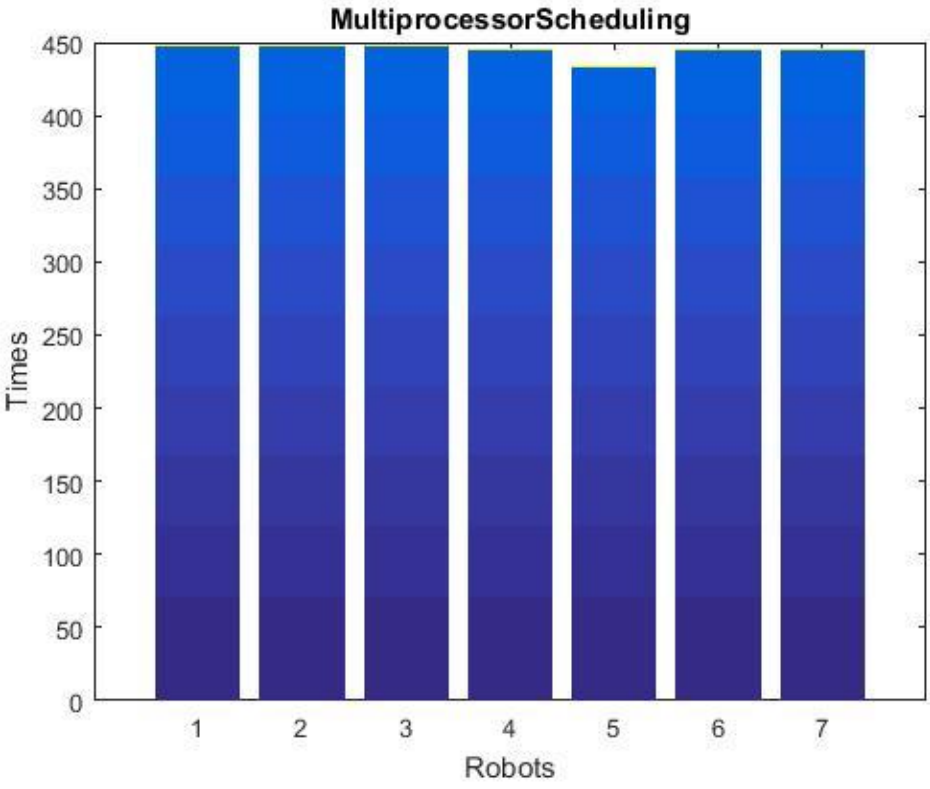

Ilustración 5-14 Asignación de tareas para 7 robots en un árbol de 1000 nodos.

```
Heuristic_distance = 3116


TimeToCover = 448
DesviaciónRelativa   = 1.0064
```

Como podemos ver en los resultados, la desviación relativa es mucho menor para este tamaño de árboles que en los árboles con menos nodos (<1%). Hay que reseñar que en estos árboles con un número tan alto de nodos el ordenador no puede encontrar la cobertura óptima en distancia.

6 CONCLUSIONES Y TRABAJO FUTURO

The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.

Claude Shannon, 1948

En este proyecto se ha presentado un estudio sobre diferentes problemas de optimización que surgen para cubrir un árbol sobre el que se desplazarán un equipo de drones que han de visitar todos los nodos. La idea era hacerlo de la manera más eficiente posible teniendo en cuenta la limitación de la batería, lo que supone que cada dron puede ejecutar un número limitado de movimientos.

Se han propuesto heurísticos con buen rendimiento para aproximar la solución óptima. Estos algoritmos garantizan la visita de todos los puntos dentro de un rango de distancia y tiempo cercanos al óptimo para árboles genéricos lo que permite su posible aplicación en el mundo real. A nivel de coste computacional, los heurísticos aquí presentados escalan linealmente cuando el tamaño del árbol tiende a infinito. Los autores de [16] han probado que la solución óptima no puede calcularse en tiempo polinomial.

Se han planteado distintas situaciones con diferentes números de robots y tamaño de árboles, lo que ha permitido llegar a las siguientes afirmaciones:

- Conforme aumenta el número de robots disminitue el tiempo de cobertura hasta un valor mínimo, que es el tiempo de la inmersión más profunda.
- Cuanto mayor es el árbol a explorar, para un mismo número de robots, aumenta el tiempo de cobertura.
- Cuanto mayor es el árbol a explorar, más distribuida queda la carga de trabajo entre los drones.
- Se ha observado que pueden coexistir dos soluciones óptimas distintas que minimicen la distancia, teniendo número de inmersiones distinto.

Este trabajo inicial da pie a distintas extensiones, entre las que podremos mencionar algunas ideas para trabajos futuros:

- Los algoritmos propuestos tienen que ser probados más en profundidad y con diferentes condiciones de contorno y restricciones, como que no pueda haber cruce entre drones que viajan simultáneamente por la misma galería. Para ello planteríamos la combinación con algoritmos para evitar colisiones, o la búsqueda de rutas alternativas.
- La velocidad de los drones puede hacerse variable y dependiente del tiempo y/o la posición y los árboles ponderados, esto es, con aristas de distinta longitud.
- Los heurísticos aquí propuestos son simples y aunque el problema es duro computacionalmente, pueden encontrarse otros procedimientos con garantía en la solución aproximada o con mejor rendimiento.

Puede plantearse también el estudio de la optimización de otros problemas, como por ejemplo minimizar el nímero de inmersiones que se realizan en una cobertura, la definición de este problema quedaría así:

Por último, definiremos otra caracterísitica que se podría adicional que se puede llegar a optimizar con heurísiticos en el que, dado un árbol *T*, una batería para los drones de *p pasos*, se obtendrá el número mínimo de inmersiones a realizar por un robot de forma que se recorra todo el árbol.

$\boldsymbol{TREE - INSPECTION\ MIN - INMERSIONS}$

Entrada:

$un\ árbol\ T = (V, E)\ con\ un\ nodo\ raíz\ r \in V$

$1\ robot$

$p \in \mathbb{N}\ tal\ que\ p \geq 2h(T)$

Solución:

$Conjunto\ de\ inmersiones\ I_i\ que\ cubren\ T\ (\forall I_i \in I: d(I_i) \leq p)$

Objetivo:

$minimizar\ \ dim(I)$

REFERENCIAS

1. Albers, S. and Henzinger, M. R. *Exploring unknown environments*, SIAM Journal on Computing, (2000), 1164-1188. 29
2. Rao, N.S.V., Hareti, S., Shi, W., and Iyengar, S. S, *Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms*, Tech. Rep. ORNL/TM-12410, Oak Ridge National Laboratory, July 1993.
3. Bender, M., Fernández, A., Ron, D., Sahai, A. and Vadhan, S.: *The power of a pebble: exploring and mapping directed graphs.* Proc. 30th Symp. Theory of Computing, ACM (1998) 269–278
4. Bender, M., Slonim, D.: *The power of team exploration: two robots can learn unlabeleddirected graphs.* Proc. FOCS 1994. (1994) 75–85
5. Deng, X. and Papadimitriou, C. H., *Exploring an unknown graph*, J. of Graph Th., 32 (1999), 265-297.
6. Awerbuch, B., Betke, M., Rivest, R., Singh, M. *Piecemeal graph exploration by a mobile robot.* Information and Computation 152(2) (1999) 321-328
7. Betke, M., Rivest, R. and Singh, M., *Piecemeal learning of an unknown environment*, Machine, Learning, 18 (1995), 231-254.
8. Duncan, C.A., Kobourov, S.G., Kumar, V.S.A, *Optimal constrained graph exploration*, In 12th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'01), 807-814.
9. Dessmark, A., Pelc, A.*: Optimal graph exploration without good maps*. Proc. ESA 2002. Volume 2461. (2002) 374
10. Diks, K., Fraigniaud, P., Kranakis, E., Pelc, A, *Tree exploration with little memory,* In 13th Ann. ACM-SIAM Symposium on Discrete Algorithms (SODA'02), 588-597.
11. Duncan, C.A., Kobourov, S.G., Kumar, V.S.A, *Optimal constrained graph exploration*, In 12th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'01), 807-814.
12. Panaite, P., Pelc, A.: *Exploring unknown undirected graphs*. Society for Industrial and Applied Mathematics (1999) 281
13. Frederickson, G. N., Hecht, M. S. and Kim, C. E., *Approximation algorithms for some routing problems*, SIAM J. on Computing, 7 (1978), 178-193.
14. Averbakh, I., Berman, O.: *(p - 1)/(p + 1)-approximate algorithms for p-traveling salesmen problems on a tree with minmax objective*. En: Discrete Applied Mathematics 75(3) (1997) 201–216
15. Díaz-Báñez, J.M., Fabila-Monroy, R., Heredia, M.A., Duque, F., Urrutia, J., Ramírez-Vigueras, A. *On optimal coverage of a tree with multiple robots*, European Journal of Operational Research, 285, 3, 844-852, 2020.
16. Caraballo L.E., Díaz-Báñez J.M. *Efficient Inspection of Underground Galleries Using k Robots with Limited Energy*. ROBOT 2017: Third Iberian Robotics Conference. ROBOT (2017) 706-717
17. Borodin, A., El-Yaniv, R: *Online computation and competitive analysis.* Cambridge University Press, New York, NY, USA (1998) `
18. Fraigniaud, P., Gasieniec, L., Kowalski, D., Pelc, A.: *Collective tree exploration*. Proc. LATIN 2004. Volume 2976. (2004) 141–151
19. Nagamochi, H., Okada, K.: *A faster 2-approximation algorithm for the minmax p-traveling salesmen problem on a tree.* Discrete Applied Mathematics 140(1-3) (2004) 103–114
20. Cormen, T.H., Leiserson, C.E., Rivest, R.L., *Introduction to Algorithms* (1st edición). MIT Press and McGraw-Hill. (1990)