



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
PROGRAMA DE MAESTRÍA Y DOCTORADO EN CIENCIAS
MATEMÁTICAS Y DE LA ESPECIALIZACIÓN EN ESTADÍSTICA
APLICADA

ESTUDIO DE LA EFICIENCIA ALGORÍTMICA PARA UN PROBLEMA
DE OPTIMIZACIÓN COMBINATORIA

TESIS
QUE PARA OPTAR POR EL GRADO DE:
DOCTOR EN CIENCIAS

PRESENTA:
CALEB ERUBIEL ANDRADE SERNAS

DIRECTOR DE TESIS:
DR. GILBERTO CALVILLO VIVES
INSTITUTO DE MATEMÁTICAS U.N.A.M.

COMITÉ TUTORAL:
DR. FEDERICO ALONSO PECINA
UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

DR. ERICK TREVIÑO AGUILAR
INSTITUTO DE MATEMÁTICAS U.N.A.M.

CIUDAD DE MÉXICO, MAYO 2021

Para la gloria y honra de mi Señor y Salvador Jesucristo.

“Y todo lo que hacéis, sea de palabra o de hecho, hacedlo todo en el nombre del Señor Jesús, dando gracias a Dios Padre por medio de él.”
Colosenses 3:17

Estudio de la eficiencia algorítmica para un problema de optimización combinatoria

Introducción

Este trabajo de tesis se encuentra en la intersección de varias disciplinas matemáticas, a saber: Optimización Combinatoria, Probabilidad y Estadística, Teoría de las Gráficas y Complejidad Computacional. Inicialmente, el nombre propuesto para este trabajo fue: "Estudio de la eficiencia algorítmica para problemas de optimización combinatoria". Quizá el objetivo inicial era demasiado ambicioso, porque finalmente el trabajo no trata en general acerca de los problemas de optimización combinatoria, sino que se centra en un problema en particular. No obstante, existen indicios que señalan que los hallazgos realizados en este trabajo se podrían generalizar a una clase amplia de problemas, a saber, los problemas que pertenecen a la clase NP-completo. De esta manera, el haber centrado el estudio en un solo problema redujo la dificultad en el presente trabajo, lo cual permitió realizar algunos hallazgos interesantes que podrían dar una visión más amplia para los problemas de decisión y optimización combinatoria.

Parte de la motivación de este trabajo surgió de algunas pláticas con el Dr. Gilberto Calvillo y el Dr. David Romero en torno a preguntas como: ¿porqué las metaheurísticas parecen ser tan buenas?, o bien, ¿cuál es la complejidad computacional de una metaheurística? Durante un curso de Optimización Combinatoria en la primavera del 2018 en la UCIM, UNAM, el Dr. David Romero presentó a los alumnos un artículo de Sörensen titulado: "Metaheuristics - The metaphor exposed" [59]. En este artículo se plantea una crítica sobria a una tendencia, o un *tsunami* como lo expresa Sörensen en su artículo, de inventar metaheurísticas novedosas basadas en alguna metáfora de un proceso realizado por el hombre o de la naturaleza. En sus propias palabras así lo describe:

"El comportamiento de virtualmente cualquier especie de insecto, el flujo del agua, músicos tocando juntos. Parece ser que ninguna idea es demasiado descabellada para servir como inspiración para lanzar aún una nueva metaheurística. En este artículo se argumenta que esta línea de investigación amenaza con dirigir el área de las metaheurísticas lejos del rigor científico."

Incluso existe una biblioteca virtual dedicada a recopilar todas las metaheurísticas conocidas denominada: "Bestiario". El nombre se debe al hecho de que se pueden encontrar diversas metaheurísticas inspiradas en muchos animales de la naturaleza: bacterias, abejas, camellos, pájaros y un largo etcétera, véase <https://github.com/fcampelo/>

EC-Bestiarium. Estas reflexiones contribuyeron a motivar la búsqueda de respuestas objetivas a las preguntas antes planteadas.

Este trabajo no versa sobre las metaheurísticas, aunque sí existe una sección en el capítulo 6 que aborda tangencialmente la cuestión. Eventualmente, el objetivo se hizo más específico y se abordó un problema propuesto por el Dr. David Romero, originalmente propuesto por el Dr. Jurek Czyzowicz al Dr. David Flores. El problema lo llamábamos de manera informal: "El problema del diamante", pues así lo platicaba con mucha amenidad el Dr. Romero: *"Imagina que hay un diamante y un grupo de robots para trasladarlo que sólo se pueden mover en una línea recta hacia atrás o hacia adelante con energía limitada, ¿cuál es la estrategia óptima para mover el diamante lo más que se pueda hacia adelante?"*. El problema es engañosamente sencillo. En más de una ocasión, el Dr. Gilberto Calvillo, el Dr. David Romero y un servidor, discutieron estrategias que se pensaban óptimas, pero eventualmente se encontraron contraejemplos que lo negaban. Esto se discute en el capítulo 5. En un principio se tomó el enfoque práctico de resolver el problema del diamante con metaheurísticas para estudiar su comportamiento. Pero, ¿cuál metaheurística? ¿con cuáles parámetros? y en realidad la pregunta importante era: ¿por qué una metaheurística? Por tradición y popularidad se eligió trabajar con Recocido Simulado y Algoritmo Genético. Pero al encontrar una familia de problemas que tienen una solución trivial, que las metaheurísticas fallaban en resolver, se decidió por tanto abandonar el enfoque y estudiar con más detenimiento la estructura del problema para poderlo resolver de manera más efectiva y eficiente.

El primer capítulo de este trabajo representa entonces una introducción a los conceptos e ideas fundamentales de la Complejidad Computacional en su enfoque clásico, por tratarse de una disciplina dedicada a entender la manera más eficiente de resolver algorítmicamente problemas de decisión u optimización. Es importante señalar que en el presente trabajo se tratará únicamente con el tiempo de ejecución algorítmica y se dejará de lado el aspecto de la memoria. El lector versado en este tema puede saltar el capítulo con seguridad, quizá con excepción de las últimas dos secciones que se refieren al fenómeno físico de transición de fase de k -satisfacibilidad y a los teoremas NFL.

Durante el proceso de entender la estructura del problema del diamante y la manera más eficiente de resolverlo, se descubrió que el problema se puede trasladar a un problema de Teoría de las Gráficas. Esta interpretación resultó muy valiosa, debido a las herramientas con las que se cuenta en esta hermosa disciplina. El problema se tradujo entonces a uno de gráficas de intervalos, de manera que se estableció una relación muy estrecha entre la solubilidad del problema original con la existencia de cierta gráfica conexa de intervalos asociada al problema. Tanto el planteamiento formal del problema, así como su interpretación como un problema de gráficas de intervalos, se discuten en el capítulo 2. Asimismo, es importante señalar que el problema inicialmente se planteó como un problema de optimización, pero durante el proceso se hizo notoria la necesidad de abordar el problema como uno de decisión. Entonces, la pregunta: ¿cuál es la estrategia óptima para mover el diamante lo más que se pueda hacia adelante? se convirtió

en: ¿existe una estrategia para que los robots lleven el diamante de una posición s a una posición t ? Esto permitió llevar el problema al famoso terreno de los problemas NP-Completos. Esto en realidad no fue una idea original ya que Chalopin et al. demostraron que la versión de decisión de este problema es NP-Completo [14, 13]. No obstante esta situación, en el capítulo 2 también se plantea una versión del problema que se demuestra ser resuelta mediante un algoritmo polinomial.

Uno de los hallazgos principales en este trabajo tiene que ver con un fenómeno en la física conocido como transición de fase. Este comportamiento se ha observado y estudiado bastante en el problema de k -satisfacibilidad. La idea consiste en formular un modelo probabilístico que describa la generación de instancias aleatorias en un espacio de parámetros que gobiernan el modelo. En k -satisfacibilidad se ha observado que existe una relación en el espacio de parámetros que describe el comportamiento de transición de fase. El umbral es una curva que divide al plano en dos zonas: las instancias con alta probabilidad de ser solubles y las instancias con alta probabilidad de ser insolubles. El mismo fenómeno se observa en el problema del diamante. En este trabajo no se logró demostrar un umbral preciso, aunque se presenta evidencia empírica mediante simulaciones. Lo que sí se logró es dar una cota superior y una cota inferior a este umbral. Este asunto se discute en el capítulo 3.

Al tratar de entender la estructura de las instancias solubles e insolubles generadas a partir del modelo probabilístico del capítulo 3, se desarrolló un algoritmo exacto para resolver el problema a partir de una técnica conocida como búsqueda con retroceso. Esto se discute en el capítulo 4. Además, se incorporó una mejora sustancial a este algoritmo al aplicar las ideas de la variante polinomial del problema del diamante. Más aún, este algoritmo permitió vislumbrar una respuesta a una pregunta inherente a cualquier problema de decisión u optimización combinatoria: ¿qué caracteriza a las instancias difíciles? En el problema del diamante, la idea es caracterizar la dificultad de las instancias mediante la gráfica de árbol generada por la búsqueda con retroceso. Un árbol con pocas hojas afirmativas será una instancia difícil en general. Un árbol lleno de hojas afirmativas será una instancia fácil. Un árbol sin hojas afirmativas será una instancia fácil si existe un criterio que permita evitar iniciar la búsqueda en el árbol, como se verá en el capítulo 3. Se puede hacer una alegoría interesante con respecto a las estaciones del año en los países septentrionales. En el verano, los árboles caducifolios están rebozantes de follaje. Con el paso del tiempo hacia el otoño, los árboles comienzan a perder hojas hasta llegar a un punto donde tienen algunas pocas hojas. Al llegar el invierno, los árboles han perdido todas sus hojas con alta probabilidad. Cuando llega la primavera, comienzan a salir algunas hojas hasta que en el verano se completa el ciclo nuevamente. De manera alegórica, el paso del tiempo representa la variación de los parámetros en el espacio de parámetros del modelo probabilístico del capítulo 3. Con esta variación, el árbol generado por la búsqueda con retroceso varía también en cuanto a la cantidad de hojas afirmativas. El invierno y el verano representan las instancias fáciles, mientras que el otoño y la primavera las instancias difíciles. El fenómeno de transición es precisamente el análogo a las

transiciones que se observan en el cambio de estaciones. La idea propuesta es sencilla: cuando el árbol tiene muchas hojas, es sencillo encontrar alguna. Si el árbol tiene pocas hojas, requiere mucho trabajo encontrar una de ellas.

En el capítulo 6 se muestra un estudio experimental para comparar la eficacia de la búsqueda con retroceso y de los algoritmos parciales que se proponen en el capítulo 5. Asimismo, se mencionan tangencialmente los resultados preliminares de este estudio con las metaheurísticas y se ofrece una posible explicación al fenómeno de la popularidad de las metaheurísticas. Esta explicación está en conexión a un modelo probabilístico que se plantea para entender el muestreo aleatorio en problemas de decisión u optimización combinatoria: *el modelo de las urnas*, descrito en el apéndice A.7. Esta misma explicación también se conecta con la caracterización de la dificultad de las instancias.

Los resultados que se presentan en este trabajo arrojan luz sobre la dificultad de los problemas de decisión u optimización combinatoria, en cuanto a que tradicionalmente los problemas NP-Complejos y NP-duros son los más difíciles de resolver, pero en este trabajo se plantea un ejemplo donde las instancias difíciles representan en realidad un porcentaje bajo. Para la gran mayoría de las instancias existe una manera eficiente para su resolución, y se cree que esto puede extenderse a la clase NP-Completo. Pero, ¿porqué habrían de extenderse estas ideas a otros problemas? La respuesta a esta cuestión, que en realidad no se aborda en este trabajo sino que se deja como trabajo futuro, es porque la definición de NP-Completo es en cierto sentido la definición de una clase de equivalencia de complejidad computacional. Si se demuestra que existe un algoritmo polinomial para resolver un problema en esta clase, eso implica que todos los problemas en esa clase se pueden resolver polinomialmente. Si se demuestra que existe un problema que no es posible resolver polinomialmente, lo mismo es cierto para el resto de los problemas en esta clase. En este sentido, se cree que el fenómeno de transición de fase sería observable en toda la clase NP-Completo, así como las ideas acerca de la dificultad de las instancias de problemas de decisión u optimización combinatoria presentadas en este trabajo.

Por último, los algoritmos propuestos en este trabajo se implementaron en Python y el código está disponible para el lector interesado en su revisión y ejecución en una plataforma denominada Google Colab. En el apéndice A.1 se encuentran los detalles.

Agradecimientos

Quiero externar mi profunda gratitud a mi Señor y Salvador Jesucristo. Él me ha dado la vida, Él me ha sostenido día con día, Él me ha concedido su gracia y favor inmerecido, Él me ha dado todo.

Quiero agradecer también a mi amada familia por su constante apoyo y amor. A mi esposa Paulina, a mi hija Hannah, ¡son una hermosa bendición! A mis padres y a mis hermanas. Les amo y doy gracias a Dios por sus vidas.

Siento una profunda gratitud y deuda con mi asesor, a quien también considero mi amigo, el Dr. Gilberto Calvillo Vives. Su mentoría, apoyo y generosidad para conmigo y mi familia han sido fundamentales para la realización de este proyecto de doctorado. Nunca lo olvidaré.

En memoria de nuestro querido amigo y colega, el Dr. David Romero, externo mi sincera gratitud por haberme inspirado a *entrarle* al problema desarrollado en este trabajo, así como por haber formado parte de mi comité tutorial y de mi formación académica durante mi estancia en la Unidad de Cuernavaca del Instituto de Matemáticas de la UNAM.

Al Dr. Federico Alonso Pecina y al Dr. Erick Treviño Aguilar les agradezco por haber formado parte de mi comité tutorial, por sus valiosas observaciones y evaluaciones durante el desarrollo de este proyecto.

Quiero expresar también mi gratitud a quienes amablemente aceptaron la invitación para formar parte del jurado de sinodales para la evaluación de este trabajo: Dr. Ricardo Strausz, Instituto de Matemáticas, UNAM; Dr. Erick Treviño Aguilar, Unidad Cuernavaca del Instituto de Matemáticas, UNAM; Dr. Rafael López Bracho, Departamento de Sistemas, UAM-A; Dr. Edgar Possani Espinosa, Departamento Académico de Matemáticas, ITAM; Dr. Francisco Javier Zaragoza Martínez, Departamento de Sistemas, UAM-A; Dr. José David Flores Peñaloza, Facultad de Ciencias, UNAM. A todos y cada uno de ustedes, gracias por su buena disposición y por sus comentarios y observaciones.

A todos y cada uno de ustedes, ¡les agradezco grandemente por haberme acompañado en esta travesía doctoral, en mayor o en menor medida, que culmina con el presente trabajo!

Índice general

Introducción	III
Agradecimientos	VII
1 Complejidad computacional	3
1.1 Introducción	3
1.2 Problemas y algoritmos	5
1.3 Función de complejidad	6
1.4 Modelo de computación y máquinas de Turing	8
1.5 Análisis del peor de los casos	10
1.6 Caso promedio	13
1.7 La transición de fase de k-satisfacibilidad	13
2 Problema de entrega de datos en una línea	17
2.1 Introducción	17
2.2 Antecedentes	17
2.3 Formulación	18
2.4 Representación geométrica de EDL	20
2.5 Consideraciones adicionales	21
2.6 Teoría de las Gráficas	23
2.7 Formulación de EDL como un problema de Teoría de las Gráficas	24
2.8 Variante polinomial de EDL: EDLA	27
2.9 Regiones activas vacías y la conexidad de las gráficas de intervalos $G_S(Q')$	29
3 Modelo probabilístico del problema EDL	31
3.1 Probabilidad	31
3.2 Modelo probabilístico para EDL	35
3.3 Gráficas de intervalos aleatorias	36
3.4 Avance promedio de los robots en instancias aleatorias de EDL	36
3.5 Cota inferior para el umbral de transición de fase de EDL	39

3.6	Probabilidad de intersección de dos intervalos aleatorios	40
3.7	Umbral de transición de fase para EDLA	45
3.8	Simulaciones	57
4	Búsqueda con retroceso para resolver EDL	61
4.1	Método exacto	61
4.2	BR mejorado	63
4.3	Simulaciones	64
4.4	Indicador de dificultad para instancias de EDL	66
5	Algoritmos parciales	71
5.1	Algoritmo glotón	71
5.2	Algoritmos glotones inversos	75
6	Estudio experimental de eficiencia	85
6.1	Algoritmos parciales y BRM	85
6.2	Optimización en EDL	93
6.3	Metaheurísticas	95
6.4	Muestreo aleatorio en el árbol generado por BR	97
6.5	Estrategia definitiva	99
I	Apéndice	103
A	Apéndice	105
A.1	Google Colab	105
A.2	Breve reseña histórica de la computación	106
A.3	Ejemplo de una MTD para resolver un problema de decisión.	108
A.4	Ejemplo de una reducción polinomial	110
A.5	Dos problemas con buena caracterización	111
A.6	Metaheurísticas para resolver EDL	112
A.7	Modelo de las urnas	115
A.8	Modelo de computación en la práctica	120
	Bibliografía	121

Capítulo 1

Complejidad computacional

1.1. Introducción

En computación, el acrónimo en inglés FLOPS (Floating point Operations Per Second) se refiere a las *operaciones de punto flotante por segundo*. Esta es una medida de desempeño de los equipos de cómputo. En esta primera sección se relaciona el progreso histórico de la computación (véase apéndice A.2) con el concepto de FLOPS. El objetivo es dar una motivación clara acerca de la importancia del estudio de la complejidad computacional.

Nombre	Unidad	FLOPS
Kilo-FLOPS	KFLOPS	10^3
Mega-FLOPS	MFLOPS	10^6
Giga-FLOPS	GFLOPS	10^9
Tera-FLOPS	TFLOPS	10^{12}
Peta-FLOPS	PFLOPS	10^{15}
Exa-FLOPS	EFLOPS	10^{18}

Figura 1.1: *Tabla de prefijos para unidades de desempeño de una computadora.*

Supóngase que se tiene un conjunto de n ciudades, tal que para cualesquiera dos ciudades existe un único camino que no pasa por otra ciudad. Por otro lado, un agente de ventas desea recorrer todas las ciudades y trazar una ruta que visite cada ciudad una única vez. El viajero debe iniciar y terminar en la misma ciudad. La pregunta que emerge es ¿de qué manera debe recorrer el agente las ciudades para que la distancia total recorrida

sea mínima? A este problema se le conoce como Agente Viajero y ha sido objeto de estudio por varias décadas, véase [3, 19]. Un enfoque natural e ingenuo para resolver este problema sería el siguiente: enumérese todas las posibles rutas que recorren cada ciudad una sola vez. Súmese las distancias entre ciudad y ciudad. Al finalizar, escójase la ruta cuya distancia sea la menor. A este procedimiento de resolución se le llama *fuerza bruta* o *búsqueda exhaustiva*. Pero, ¿qué tan bueno es este procedimiento para encontrar la solución? Para un conjunto de n ciudades se tendrían que examinar $\frac{1}{2}(n-1)!$ rutas y por cada recorrido se tendrían que sumar n distancias para encontrar la ruta *óptima*. La cantidad de operaciones se vuelve entonces astronómica para valores moderados de n . En la Figura 1.2, elaborada por el autor, se muestra el tiempo de cómputo aproximado que tardarían algunas de las supercomputadoras que aparecen en el apéndice A2. Se hace la simplificación de que cada suma es una operación aritmética de punto flotante.

Supercomputadora	$n = 10$	$n = 20$	$n = 30$
ENIAC 1946 - 5 KFLOPS	6 minutos	7.7×10^4 siglos	8.4×10^{18} siglos
CDC 6600 1964 - 1 MFLOPS	1.8 segundos	385 siglos	4.2×10^{16} siglos
CRAY-2 1985 - 1.9 GFLOPS	9.5×10^{-4} segundos	20.3 años	2.2×10^{13} siglos
INTEL ASCI RED 1996 - 1.34 TFLOPS	1.3×10^{-6} segundos	10.5 días	3.1×10^{10} siglos
SUMMIT 2018 - 200 PFLOPS	9×10^{-12} segundos	6 segundos	2.1×10^5 siglos

Figura 1.2: Comparación del tiempo de cómputo de distintas supercomputadoras para resolver el problema del Agente Viajero por fuerza bruta. En la columna izquierda se encuentra el nombre, el año y la velocidad de cada supercomputadora.

La Figura 1.2 muestra claramente que este procedimiento de fuerza bruta resulta impráctico para resolver el problema en cuestión. Como se ha visto en este ejemplo, examinar cada posibilidad una por una no es una opción práctica. La pregunta de interés que surge en torno al Agente Viajero es: ¿existe un procedimiento eficiente para resolver este problema? En este primer capítulo se hablará de manera introductoria acerca de los conceptos fundamentales de la complejidad computacional que se relacionan con esta cuestión. Para un estudio más amplio sobre este tema véase [32, 4, 1, 62].

1.2. Problemas y algoritmos

Las definiciones y conceptos presentados en este capítulo siguen la línea de [4] y [32].

Un *problema* Π es una pregunta sobre la existencia de un objeto matemático que satisfaga las llamadas *condiciones del problema*. Dicha pregunta posee parámetros cuyos valores se dejan sin especificar. Un problema se define al proporcionar:

1. Una descripción general de todos sus parámetros.
2. Un conjunto de condiciones que el objeto matemático o *respuesta* debe satisfacer.

Una *instancia* de un problema se obtiene al especificar valores particulares para todos los parámetros del problema.

En la sección anterior se planteó la versión de optimización del problema del Agente Viajero, donde el objetivo es minimizar la distancia del recorrido. En la siguiente sección se presenta formalmente la versión de decisión del Agente Viajero.

Problemas de decisión

En el año 1900, David Hilbert presentó ante la comunidad matemática el ideal de basar todas las matemáticas en sistemas axiomáticos que sean *completos* (cada afirmación dentro del sistema axiomático se puede demostrar falsa o verdadera) y que sean *consistentes* (no demostrar afirmaciones que sean contradictorias dentro del sistema axiomático). Sin embargo, en 1931 Kurt Gödel dió a conocer sus teoremas de incompletitud [35], en los cuales demostró que para cada sistema axiomático consistente S , lo suficientemente rico como para realizar cierta aritmética elemental, existen proposiciones que no pueden ser demostradas falsas y tampoco verdaderas en S . Asimismo, en 1928 Hilbert y Ackermann propusieron el Entscheidungsproblem (problema de la decisión) [41], donde se pregunta si existe un algoritmo que considere una proposición y que responda verdadero o falso, dependiendo si es universalmente válida, es decir, si puede ser deducida de los axiomas utilizando las reglas de la lógica. En 1936, tanto Alan Turing [63] como Alonzo Church [15] demostraron que es imposible dar una solución general al reto lanzado por Hilbert y Ackermann. El décimo problema en la lista de Hilbert proponía encontrar un algoritmo para determinar si dada una ecuación polinomial diofantina, con coeficientes enteros, existe una solución entera. Fue hasta 1970 que Matiyasevich demostró que tal algoritmo no existe [49].

Los problemas de decisión plantean preguntas cuya respuesta es sí (verdadero) o no (falso). Como se ha discutido en el párrafo anterior, existen problemas de decisión que son *indecidibles*. No obstante, en este trabajo se trabajará únicamente con problemas de decisión que sí tienen respuesta. A continuación, el planeamiento formal del Agente Viajero como problema de decisión.

Problema del Agente Viajero (AV)

- Parámetros: Un conjunto finito $C = \{c_1, c_2, \dots, c_n\}$ de ciudades, una distancia $d : C \times C \rightarrow \mathbb{Z}^+$ para cada par de ciudades c_i, c_j en C , y una cota $B \in \mathbb{Z}^+$.
- Pregunta: ¿Existe un recorrido de todas las ciudades en C cuya longitud total sea a lo más B ? Es decir, un ordenamiento $\langle c_{\pi(1)}, c_{\pi(2)}, \dots, c_{\pi(n)} \rangle$, donde $\pi : \{1, 2, \dots, n\} \rightarrow \{1, 2, \dots, n\}$ es una permutación, tal que

$$d(c_{\pi(n)}, c_{\pi(1)}) + \sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) \leq B.$$

Algoritmos

Una vez que se ha definido lo que es un problema, se procede a hablar de los métodos para su resolución. Esto lleva al concepto de algoritmo. La siguiente definición se debe a Trajtenbrot [62].

Definición 1.1. Un *algoritmo* es la prescripción exacta sobre el cumplimiento de cierto sistema de operaciones en un orden determinado para la resolución de todos los problemas de algún tipo dado.

Así pues, los algoritmos son procedimientos que consisten en una serie de pasos bien definidos para la resolución de problemas específicos.

Definición 1.2. Un algoritmo *resuelve* un problema Π , si dicho algoritmo puede ser aplicado a cada instancia I de Π , de tal modo que siempre se produce una solución para dicha instancia I .

La función de complejidad de un algoritmo se expresa en función de una única variable, el *tamaño de la instancia* a resolver, la cuál representa una medida de la información necesaria para describir a la instancia. Para definir el tamaño de una instancia se necesita un buen *esquema de codificación*.

Definición 1.3. Un *esquema de codificación* consiste en convertir una instancia I de un problema Π en una cadena finita de símbolos escogidos de un alfabeto finito Σ . El *tamaño de una instancia* I de un problema Π se define como el número de símbolos de la cadena que resultó de su codificación.

Se denota por Σ^* el conjunto de todas las cadenas finitas cuyo alfabeto es Σ .

1.3. Función de complejidad

En 1965, Jack Edmonds propuso una definición de lo que es un *buen algoritmo* como aquél cuyo tiempo de ejecución depende algebraicamente del tamaño de la instancia

[26]. Otra contribución pionera en este ámbito se debe a Alan Cobham [16].

Cada algoritmo tiene asociada una *función de complejidad computacional*, la cual expresa la cantidad de pasos necesarios para resolver una instancia en función de su tamaño. La función de complejidad computacional depende del modelo de cómputo y el esquema de codificación. Lo cierto es que al día de hoy no se ha logrado establecer un esquema de codificación universal. Por este motivo, se evita entrar en los detalles de dicha discusión.

Distintos algoritmos poseen una variedad amplia de funciones de complejidad computacional. Sin embargo, la distinción básica propuesta por Cobham-Edmonds entre algoritmos polinomiales y algoritmos exponenciales, resulta ser la más utilizada entre la comunidad científica para describir la dificultad inherente de los algoritmos y problemas. A continuación se presenta un ejemplo adaptado de [32] donde se ilustra esta situación.

Comparativo: funciones de complejidad.

Se podría pensar que el incremento dramático en la velocidad de las computadoras actuales podría disminuir la importancia de los algoritmos eficientes, pero la realidad es distinta. Mientras que las computadoras se vuelven más rápidas y se pueden manejar problemas cada vez más grandes, la complejidad computacional de un algoritmo sigue siendo el factor determinante del tamaño de los problemas que se pueden resolver. Considérese una computadora personal de alto desempeño capaz de operar a 1 TFLOP (procesador Intel Core i7 de última generación). Supóngase que en esta computadora se ejecutan distintos algoritmos. Cada algoritmo tiene su respectiva función de complejidad y se ejecutan en instancias de distintos tamaños. Se calcula el tiempo aproximado de cómputo que tardaría dicha computadora en realizar estos experimentos. La Figura 3.3 resume los resultados.

La primer columna de la tabla representa las funciones de complejidad de los seis distintos algoritmos teóricos que se ejecutarían en este experimento imaginario. Resulta evidente que los dos primeros algoritmos con funciones de complejidad polinomiales se ejecutan velozmente. Por otro lado, aunque el cuarto algoritmo tiene una función de complejidad polinomial, resulta totalmente impráctico, por lo que *polinomial* no es necesariamente sinónimo de *eficiente*. Incluso, los dos últimos algoritmos con funciones de complejidad exponenciales son más eficientes que el cuarto algoritmo para valores de $n \leq 50$. Aunque es cierto que también resultan totalmente imprácticos para valores moderados de n . Esta tabla revela las razones porqué los algoritmos cuya función de complejidad es polinomial son, en general, más deseables que aquellos cuya función de complejidad no es polinomial.

	$n = 10$	$n = 20$	$n = 50$	$n = 100$	$n = 500$
n^2	1×10^{-10} segundos	4×10^{-10} segundos	2.5×10^{-9} segundos	1×10^{-8} segundos	2.5×10^{-7} segundos
n^4	1×10^{-8} segundos	1.6×10^{-7} segundos	6.2×10^{-6} segundos	0.0001 segundos	0.0625 segundos
n^8	0.0001 segundos	0.0256 segundos	39 segundos	2.77 horas	123.8 años
n^{16}	2.77 horas	20.7 años	4.8×10^5 siglos	3.1×10^{10} siglos	4.8×10^{21} siglos
2^n	1×10^{-9} segundos	1×10^{-6} segundos	18.7 minutos	4×10^8 siglos	1×10^{129} siglos
3^n	5.9×10^{-8} segundos	0.003 segundos	2.2×10^4 años	1.6×10^{26} siglos	1×10^{217} siglos

Figura 1.3: Comparación de distintas funciones de complejidad polinomiales y exponenciales.

Análisis asintótico

Se introduce a continuación una notación asintótica, adaptada del análisis matemático, cuyo uso fue sugerido por Donald Knuth en 1976 para el análisis de algoritmos [47]. Esta notación se ha vuelto el estándar para hacer afirmaciones sobre las funciones de complejidad de los algoritmos y se utilizará en este trabajo.

Definición 1.4. Si g y f son dos funciones cuyo dominio es \mathbb{N} y cuyo codominio es \mathbb{R} , se define

- $g(n) \in O(f(n))$ si existen constantes $c > 0$ y n_0 tal que $g(n) \leq c \cdot f(n)$ para toda $n \geq n_0$.
- $g(n) \in \Omega(f(n))$ si existen constantes $c > 0$ y n_0 tal que $g(n) \geq c \cdot f(n)$ para toda $n \geq n_0$.
- $g(n) \in \Theta(f(n))$ si existen constantes $c_1, c_2 > 0$ y n_0 tal que $c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n)$ para toda $n \geq n_0$. Equivalentemente, si $g(n) \in O(f(n))$ y $g(n) \in \Omega(f(n))$.
- $g(n) \in o(f(n))$ si para toda constante $\varepsilon > 0$ existe n_0 tal que $|g(x)| \leq \varepsilon \cdot f(n)$ para toda $n \geq n_0$.

1.4. Modelo de computación y máquinas de Turing

Se tienen registros de que el hombre tiene casi seis mil años haciendo cálculos mediante la aplicación de reglas mecánicas para manipular números (véase apéndice A.2).

En este proceso de cálculo, usualmente se requería de cierto lugar para anotar los resultados intermedios. La máquina de Turing es un formalismo matemático que encierra esta noción. El modelo de cómputo más estándar y básico es la *máquina de Turing determinista de una cinta*, que se abrevia como MTD. Dicho modelo consiste de un *conjunto de estados*, una *cabeza de escritura-lectura*, y una *cinta infinita a la izquierda y a la derecha*, compuesta de *cuadros* etiquetados como $\dots, -2, -1, 0, 1, 2, \dots$.

Definición 1.5. Una MTD se especifica con la siguiente información:

1. Un conjunto finito Γ de *símbolos de cinta*, que incluye un alfabeto o subconjunto $\Sigma \subset \Gamma$ de *símbolos de entrada*, y un *símbolo blanco* $b \in \Gamma - \Sigma$.
2. Un conjunto finito Q de *estados*, que incluye un *estado inicial* q_0 y dos *estados de paro* diferenciados q_S y q_N .
3. Una función de transición $\delta : (Q - \{q_S, q_N\}) \times \Gamma \rightarrow Q \times \Gamma \times \{-1, +1\}$.

Esta definición admite la posibilidad que una MTD no pare en su cálculo. No obstante, para los propósitos de esta tesis se consideran únicamente MTDs que siempre paran.

Definición 1.6. Sea Π un problema de decisión. Una MTD M *resuelve* Π si M se detiene en algún estado de paro, q_S o q_N , para cada cadena de entrada en Σ^* que codifica una instancia de Π .

El subconjunto de Σ^* que consiste en las cadenas que codifican instancias afirmativas de Π se denota por S_Π . El conjunto de cadenas que codifican instancias negativas se denota por N_Π .

El *tiempo* utilizado en la ejecución de una MTD M en una cadena de entrada x es el número de pasos que ocurren en dicha computación hasta que se llega a un estado de paro.

Definición 1.7. Si M es una MTD que se detiene para todas las cadenas de entrada en Σ^* , su *función de complejidad computacional* $T_M : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ está dada por

$$T_M(n) = \max \{m : \exists x \in \Sigma^*, \text{ con } |x| = n, \text{ tal que la ejecución de } M \text{ en } x \text{ requiere tiempo } m\}$$

Definición 1.8. Se dice que M es una MTD de *complejidad polinomial* cuando $T_M(n) \in O(p(n))$, donde $n_0 = 0$ y $p(n)$ es un polinomio.

En el apéndice A.3 se encuentra un ejemplo de una MTD para resolver un problema de decisión.

1.5. Análisis del peor de los casos

El análisis del peor de los casos plantea que si para resolver un problema Π con un algoritmo A , donde dicho algoritmo tiene una función de complejidad $g(n) \in O(f(n))$, entonces el número de pasos para resolver algunas instancias de Π puede ser tan grande como $\Theta(f(n))$. Esto es en el peor caso, y en general se requieren menos de $\Theta(f(n))$ pasos, lo cual se observa con bastante frecuencia en la práctica. Así que, tal vez esta medida del peor de los casos no describa de manera justa el desempeño de un algoritmo del todo. El caso paradigmático es el del algoritmo simplex [23] que ha demostrado ser en la práctica muy eficiente para resolver problemas de programación lineal. No obstante, en 1972 Klee y Minty demostraron la existencia de instancias del problema de programación lineal que hacen que, en el peor de los casos, el simplex realice un número de pasos exponencial en el tamaño de la instancia [45].

La clase P

Una clase de complejidad básica e importante es P. Esta clase encierra a los problemas que se pueden resolver con un algoritmo cuya función de complejidad está acotada por un polinomio.

Definición 1.9. $P = \{\Pi : \exists \text{ una MTD } M \text{ de complejidad polinomial que resuelve a } \Pi\}$

Entonces, un problema de decisión Π pertenece a P si existe una MTD cuya complejidad es polinomial. Es una costumbre generalizada omitir la mención del esquema de codificación. La práctica estándar, que seguiremos en este trabajo, es hablar de los algoritmos de una manera independiente al esquema de codificación. De este modo, la clase P es un formalismo de las ideas propuestas por Edmonds y Cobham para clasificar a los problemas que se pueden resolver *eficientemente*.

La clase NP

En 1966 Jack Edmonds conjeturó que "*no existe un buen algoritmo para el problema del Agente Viajero*" [27]. Al parecer, su conjetura ha resultado válida hasta el día de hoy. No se ha podido demostrar la existencia de un algoritmo polinomial para resolver el problema del Agente Viajero, y el consenso general es que lo más probable es que tal algoritmo no exista. No obstante la dificultad inherente del problema del Agente Viajero, dado un conjunto de ciudades $C = \{c_1, c_2, \dots, c_n\}$ con sus respectivas distancias y $B \in \mathbb{Z}^+$, verificar que un ordenamiento $\langle c_{\pi(1)}, c_{\pi(2)}, \dots, c_{\pi(n)} \rangle$ satisfaga la condición de que al recorrer las ciudades en dicho orden la distancia total sea menor o igual que B , es una tarea bastante sencilla. La complejidad computacional para realizar dicha tarea es $O(n)$. Este conjunto de ciudades es una evidencia o *certificado* de una respuesta afirmativa. Entonces, la clase de problemas de decisión cuyas respuestas afirmativas poseen un

certificado que se puede verificar polinomialmente se le denomina NP.

Definición 1.10. Un *certificado* es una cadena de símbolos que representa una prueba para verificar que la respuesta a una instancia I de un problema de decisión Π es afirmativa.

Definición 1.11. Un problema de decisión Π está en NP si existe una MTD polinomial M tal que para cada instancia $I \in S_\Pi$ existe un certificado polinomial, en el tamaño de la instancia de I , tal que M verifica que $I \in S_\Pi$ utilizando el certificado como prueba.

La clase coNP

Una propiedad importante de la clase NP es que existe una asimetría con respecto a las instancias cuya respuesta es sí, de aquellas cuya respuesta es no. Esto no sucede en la clase P. Para un problema de decisión $\Pi \in P$ la pregunta: dada una instancia $I \in \Pi$, ¿la propiedad X es verdadera para I ? se puede responder con una MTD en tiempo polinomial. Entonces, *el problema complementario*: dada una instancia $I \in \Pi$, ¿la propiedad X es *falsa* para I ?, también se puede responder con una MTD en tiempo polinomial. Esto se logra al intercambiar los estados q_S y q_N en la primer MTD. Por otro lado, no existe evidencia de que algo similar suceda en la clase NP. Los indicios apuntan a que esto no es así. Esta situación originó una nueva clase de complejidad computacional, coNP, que se define a continuación.

Definición 1.12. $\text{coNP} = \{\Pi : \text{el problema complementario de } \Pi \text{ está en NP}\}$

Pero, ¿qué relación existe entre NP y coNP? El consenso general es que $\text{NP} \neq \text{coNP}$. Dado un problema Π , el que sus instancias afirmativas tengan certificados polinomiales, no necesariamente quiere decir que sus instancias negativas tengan certificados polinomiales. De este modo, se puede decir que un problema Π que se encuentre en $\text{NP} \cap \text{coNP}$ tiene una *buena caracterización*, dado que hay una manera eficiente para verificar tanto sus instancias afirmativas como sus instancias negativas. Este término fue propuesto por Edmonds en 1964 en su artículo "Maximum partition of a matroid into independent subsets" [25]. Dado que $P \subset \text{NP} \cap \text{coNP}$, entonces es necesario que un problema tenga una buena caracterización para que exista un algoritmo polinomial que lo resuelva. Por otro lado, no existe aún evidencia que la buena caracterización sea también suficiente para asegurar la existencia de dicho algoritmo. Fue Edmonds quien propuso su *conjetura de la roca* donde afirma que en efecto la buena caracterización es equivalente a que un problema esté en P, es decir $P = \text{NP} \cap \text{coNP}$. Esta conjetura está grabada en una roca afuera de su casa, de ahí el nombre. Para concluir esta sección, se remite al lector al apéndice A.5 para ejemplificar dos problemas con buena caracterización, donde uno se sabe que está en P y del otro no se sabe.

Reducciones polinomiales

Definición 1.13. Una *transformación polinomial* de un problema de decisión Π_1 a un problema de decisión Π_2 es una función $f : \Pi_1 \rightarrow \Pi_2$ que satisface las siguientes dos condiciones:

1. La función f es calculable por un algoritmo polinomial.
2. Para toda instancia I de Π_1 , $I \in S_{\Pi_1}$ si y sólo si $f(I) \in S_{\Pi_2}$.

Si existe una transformación polinomial de Π_1 a Π_2 , escribimos $\Pi_1 \propto \Pi_2$, o bien decimos que Π_1 se *reduce polinomialmente* a Π_2 . El significado de esta definición resulta evidente en el siguiente resultado cuya demostración básicamente consiste en componer funciones polinomiales.

Lema 1.1. Si $\Pi_1 \propto \Pi_2$, entonces $\Pi_2 \in P$ implica que $\Pi_1 \in P$.

La técnica de transformar un problema en otro, mediante una transformación polinomial, es lo que se conoce como una *reducción polinomial*. Para concluir esta sección, se remite al lector al apéndice A.4 donde se ejemplifica cómo reducir polinomialmente un problema en otro.

La clase NP-completo

Una propiedad muy útil y deseable de las reducciones polinomiales es la transitividad, como lo expresa el siguiente lema.

Lema 1.2. Si $\Pi_1 \propto \Pi_2$ y $\Pi_2 \propto \Pi_3$, entonces $\Pi_1 \propto \Pi_3$.

La demostración se sigue de manera directa al componer las transformaciones polinomiales $f : \Pi_1 \rightarrow \Pi_2$ y $g : \Pi_2 \rightarrow \Pi_3$.

A continuación, la definición formal de la clase NP-completo.

Definición 1.14. Un problema Π se define como *NP-completo* si $\Pi \in \text{NP}$, y si para cualquier otro problema $\Pi' \in \text{NP}$ se tiene que $\Pi' \propto \Pi$.

El lema 1.1 implica lo siguiente: si algún problema NP-completo puede ser resuelto en tiempo polinomial, entonces todos los problemas en NP también. Si un problema en NP es intratable, entonces todos los problemas NP-completos también. A continuación, un lema que da una estrategia para demostrar que un problema es NP-completo.

Lema 1.3. Si Π_1 y Π_2 pertenecen a NP, Π_1 es NP-completo, y $\Pi_1 \propto \Pi_2$, entonces Π_2 es NP-completo.

Para demostrar que algún problema Π es NP-completo, la estrategia es la siguiente:

1. Demostrar que $\Pi \in \text{NP}$.

2. Reducir un problema conocido Π' que sea NP-completo en el problema Π .

Stephen Cook fue quien exhibió el primer problema NP-completo. En 1971 demostró que el problema de satisfacibilidad es NP-completo [18]. A partir de ese momento, se ha utilizado la estrategia descrita para demostrar que miles de problemas son NP-completos. El primer trabajo notable siguiendo esta estrategia es el de Richard Karp [43].

El consenso general es que $P \neq NP$. Esto sugiere que no existe una manera eficiente de resolver a los problemas NP-completos. Asimismo, resulta cierto que hay problemas que pertenecen a esta clase, como el problema de la mochila, pero se han podido resolver de manera eficiente en casos particulares [71]. Lo anterior nos dice que aunque los problemas NP-completos representen la clase de problemas más difíciles en NP, no se debe desestimar la posibilidad de encontrar una manera eficiente para resolver ciertos casos de interés para un problema en particular.

1.6. Caso promedio

En un esfuerzo por dar una explicación teórica sobre la eficiencia del simplex, Steve Smale demostró que el número de pivoteos requeridos para resolver un problema de programación lineal es, en promedio, proporcional al número de variables [58]. Lo que hizo Smale para analizar el algoritmo simplex es un ejemplo de lo que se conoce como *análisis del caso promedio*. Poco tiempo después, Leonid Levin hizo una exposición general del análisis del caso promedio [48].

1.7. La transición de fase de k-satisfacibilidad

Se concluye este capítulo con la discusión sobre un fenómeno que ha llamado la atención en el ámbito de la complejidad computacional en las décadas recientes. Este fenómeno es análogo con un fenómeno físico denominado *transición de fase*, véase [51]. Las transiciones de fase representan cambios súbitos en el estado de un sistema físico, como el resultado del cambio de alguna condición externa, como la temperatura o la presión. Por lo general, este cambio se produce de manera discontinua. La medición de las condiciones externas en las que se produce esta transformación es lo que se denomina transición de fase. Por ejemplo, cuando un líquido se convierte en gas al ser elevada su temperatura hasta el punto de ebullición, lo cual da como resultado un cambio abrupto en el volumen. En este caso, las condiciones de presión y temperatura determinan la transición de fase.

Como se ha visto, la complejidad computacional se centra en la dificultad para resolver problemas con estructuras combinatorias. El paradigma actual para definir la complejidad de un problema combinatorio es el del peor de los casos, donde la complejidad de un problema se determina con base en la dificultad para resolver la peor instancia

posible de dicho problema. Pero también existe el enfoque alternativo del análisis del caso promedio. En este caso, se estudia la dificultad promedio para resolver instancias generadas aleatoriamente. Esto lleva a hacer la observación de que, dado un problema Π , existe una diversidad de instancias $I \in \Pi$ que varían en términos de la dificultad para ser resueltas. El gran misterio es poder determinar las propiedades que caracterizan la dificultad de las instancias.

El fenómeno de transición de fase, en el ámbito de la complejidad computacional, es que en ciertos problemas combinatorios se pueden definir parámetros de control para generar instancias aleatorias. Al variar estos parámetros de control el problema en cuestión exhibe una transición de fase con respecto a la solubilidad de las instancias generadas. El problema en donde este fenómeno se ha estudiado de manera extensa es k -satisfacibilidad, véase por ejemplo [36, 9, 21].

El problema de k -satisfacibilidad consiste en responder si existe una asignación válida para n variables booleanas, de tal modo que la conjunción de m cláusulas booleanas sea verdadera. En cada cláusula booleana hay k literales en forma disjunta, que se toman de las n variables booleanas y sus negaciones. El caso más sencillo es cuando $k = 2$. El problema 2-satisfacibilidad se encuentra en P [5], pero se ha demostrado que para $k \geq 3$ k -satisfacibilidad es NP-completo [18].

Ahora bien, si se generan instancias aleatorias de k -satisfacibilidad, donde m cláusulas se escogen aleatoriamente de manera uniforme entre todas las posibles cláusulas de longitud k (sus literales provienen del conjunto de n variables booleanas y sus negaciones), y consideramos el cociente $\varphi(k) = \frac{m}{n}$, existe una cantidad considerable de evidencia empírica que sugiere que hay una transición de fase para k -satisfacibilidad para cierto valor $\varphi_c(k)$ del parámetro $\varphi(k)$, véase por ejemplo [50, 21].

Si bien es cierto que 2-satisfacibilidad se encuentra en P, se ha demostrado que $\varphi_c(2) = 1$, véase [36, 54]. El resultado dice que si $\varphi(2) < 1$, las instancias aleatorias generadas con este parámetro son satisfacibles con alta probabilidad. Si $\varphi(2) > 1$ las instancias aleatorias generadas con este parámetro son insatisfacibles con alta probabilidad. De manera intuitiva, cuando la cantidad de variables booleanas n es mayor que la cantidad de cláusulas m , es más factible que exista una asignación verdadera pues hay mayor libertad para asignar valores a las variables. Por el contrario, si la cantidad de variables booleanas es menor que la cantidad de cláusulas, las instancias tienen más restricciones que limitan la asignación de valores a las variables.

Por su parte, Bollobás et al. [9], demostraron que para 2-satisfacibilidad existe una ventana $W(n, \delta) = (\varphi_-(n, \delta), \varphi_+(n, \delta))$ alrededor del valor crítico $\varphi_c(2)$, tal que la probabilidad de satisfacibilidad es mayor que $1 - \delta$ para $\varphi < \varphi_-$ y es menor que δ para $\varphi > \varphi_+$. De esta manera, se observa que la transición de fase para 2-satisfacibilidad es en realidad una banda alrededor del parámetro $\varphi_c(2)$.

No se ha podido determinar un umbral exacto para $k = 3$, como en el caso de 2-satisfacibilidad, mucho menos para valores $k > 3$, pero sí se han encontrado cotas, tanto inferiores como superiores, para el conjeturado valor crítico $\varphi_c(3)$, véase [9].

En este trabajo, se ha identificado el fenómeno de transición de fase para el problema Entrega de Datos en una Línea (EDL) que se describe en el capítulo 2. La observación básica es que al generar instancias aleatorias mediante cierto modelo probabilístico, existen dos parámetros de control que se relacionan mediante cierta función. Esta función divide el espacio de parámetros en dos zonas: las instancias cuya respuesta es afirmativa con alta probabilidad y las instancias cuya respuesta es negativa con alta probabilidad. Asimismo, se ha encontrado que las instancias difíciles se encuentran cerca de la zona de transición de fase.

Capítulo 2

Problema de entrega de datos en una línea

2.1. Introducción

La producción de robots móviles económicos y de construcción cada vez más simple se ha convertido en una realidad. Actualmente, los drones se pueden utilizar para diferentes tareas antes insospechadas. Por ejemplo, el uso de una red de drones para entregar paquetes a clientes desde tiendas minoristas o servicios de mensajería, como el proyecto piloto de Amazon *prime Air*. El problema de Entrega de Datos (ED) es una abstracción matemática de tal escenario que se ha estudiado bastante últimamente, véase [2, 13, 7, 6, 14, 22].

En este trabajo, se trata con uno de los problemas de entrega de datos más simples. Los robots están obligados a moverse en una línea. En esta versión, Entrega de Datos en una Línea (EDL), un conjunto de n robots se colocan en las *posiciones* $x_i \geq 0$ con *capacidades* $\rho_i > 0$ con $i = 1, \dots, n$. La capacidad indica la longitud máxima de un recorrido para cada robot. La pregunta es si existe un orden en el que los robots deben moverse para recoger los datos en un punto s llamado *salida*, para llevarlos a un punto predeterminado llamado la *terminal* $t > s$, por lo que EDL es un problema de decisión.

2.2. Antecedentes

El problema EDL fue introducido por Chalopin et al [14], y se demostró que es NP-completo. Para los casos en que todos los valores de entrada son enteros, demostraron que existe un algoritmo de tiempo pseudo-polinomial.

EDL proviene de un caso más general en gráficas con pesos en sus aristas y múltiples salidas [13]. Los robots se colocan inicialmente en los vértices de una gráfica y pueden viajar a lo largo de sus aristas con energía limitada. El objetivo es recopilar datos de un conjunto de salidas distintas y entregarlos a una sola terminal. Los pesos en las aristas

representan la energía necesaria para cruzar dicha arista. En dicho trabajo, se demuestra que el problema en cuestión es NP-completo. Un enfoque ligeramente diferente con respecto al último problema se dio en [2]. Esta vez, el objetivo es realizar una *transmisión convergente eficiente*, lo que significa que los robots colaboran para que la información inicial de todos los robots eventualmente se transmita a algún robot y no a un nodo terminal. La pregunta en este caso es cuál es el valor mínimo de la energía de la batería para que se pueda lograr la transmisión convergente. Se demostró que este problema es NP-completo y se proporcionó un algoritmo parcial. En esta configuración de gráficas con aristas con pesos, también se estudió un enfoque en el que los robots pueden transferir no solo información sino también energía cuando se encuentran, ya sea para entregar datos de una salida a una terminal o para lograr una transmisión convergente [22]. Una versión simplificada con una sola salida y una sola terminal, donde la energía se limita a dos unidades para cada robot, se estudió en [6], donde también se demostró que una variante de su planteamiento está en P. Una vez más, se demostró que estos problemas son NP-completos, aunque existen algoritmos polinomiales para casos especiales como cuando la gráfica es un árbol o una trayectoria [22]. Por lo tanto, existen trabajos previos en los que ya se han estudiado diferentes variantes del problema. En su mayoría se trata de problemas NP-completos, con algunos casos especiales que pueden ser resueltos en tiempo polinomial.

2.3. Formulación

La formulación del problema de decisión EDL es como se describe a continuación.

Entrega de Datos en una Línea (EDL)

- Parámetros: $I = (\{(x_i, \rho_i)\}_{i=1}^n, s, t)$, donde cada pareja (x_i, ρ_i) representa un robot i con posición $x_i \in \mathbb{Q}$ y capacidad $\rho_i \in \mathbb{Q}^+$; $s \in \mathbb{Q}$ representa la posición inicial de los datos, o la salida, y $t \in \mathbb{Q}$, $s < t$, representa la terminal.
- Pregunta: ¿Existe alguna subsucesión de robots $\langle i_1, \dots, i_k \rangle$ que en ese orden puedan mover los datos de s a t ?

Debido a que la precisión de una computadora es limitada, resulta imposible trabajar con números reales. Si bien es cierto que los resultados analíticos presentados en este trabajo son válidos para posiciones y capacidades en \mathbb{R} , la parte computacional está restringida a los números racionales \mathbb{Q} . De ahí que en la formulación del problema se restrinja a trabajar en \mathbb{Q} . En este punto, es importante remitir al lector al apéndice A.8 donde se dan algunos detalles del modelo de cómputo que se considera en este trabajo.

A continuación, se definen los dos únicos tipos de movimiento que se consideran para los robots en este trabajo. Restringir el movimiento de los robots de esta manera será suficiente para entender y dar solución al problema.

Algoritmo 2.1 Algoritmo polinomial de orden $O(n)$ para verificar un certificado de EDL. En la libreta de Python: EDLA/**DataDelivery.moveRobots()**.

```

1: verificar_certificado ( $I$ , certificado):
2:   para  $(x_i, \rho_i)$  en certificado:
3:     distancia =  $|d - x_i|$ 
4:     si distancia  $< \rho_i$  y  $(x_i, \rho_i)$  no se ha utilizado antes:
5:       Se etiqueta  $(x_i, \rho_i)$  como usado;
6:       capacidad_restante =  $\rho_i - \text{distancia}$ ;
7:       Se actualiza  $d = d + \text{capacidad\_restante}$ ;
8: terminar;
   Si  $d \geq t$ : imprimir («Sí es un certificado»);
   en otro caso: imprimir («No es un certificado»);

```

Definición 2.1. Si el robot está antes de la posición d donde están los datos, éste se debe mover únicamente hacia adelante, primero para recoger los datos y luego para moverlos todo lo que pueda hacia adelante. Si está después de d , entonces tiene que moverse primero hacia atrás para llegar a d y luego hacia adelante todo lo que pueda. En ambos casos, el robot cubre el intervalo $[x_i - a\rho_i, x_i + (1 - 2a)\rho_i]$ donde $a = 0$ para el primer caso y $a = (x_i - d)/\rho_i$ para el segundo. Cuando $a = 0$ el *alcance superior* del robot i es $x_i + \rho_i$. Cuando $a = 1$ el *alcance inferior* del robot i es $x_i - \rho_i$.

Para justificar esta simplificación, considérese un robot (x_i, ρ_i) en un certificado C para alguna instancia I de EDL que se ha resuelto sin restricciones para el movimiento de los robots. Sea p_i el punto donde (x_i, ρ_i) recogió los datos y q_i la última posición donde dejó los datos. Sin importar de qué manera se movió el robot, podemos sustituir sus movimientos por únicamente dos movimientos: moverse desde x_i hasta p_i y desde p_i hasta q_i . Debe ser claro que esto no altera la solubilidad.

Esta definición es crucial para definir gráficas asociadas al problema, como se verá en la sección 2.7. Con las reglas del movimiento de los robots descritas en la definición anterior, un certificado para EDL es una descripción del subconjunto de robots que se mueven y el orden en que lo hacen. En dicho orden, el primer robot se mueve al inicio, recoge los datos y los mueve hacia adelante. El segundo robot se mueve hasta el punto donde está el primer robot, toma los datos y se mueve hacia adelante. Un tercer robot se hace cargo, y así sucesivamente hasta que los datos llegan, de ser posible, a la terminal t . De acuerdo a lo anterior, se puede verificar un certificado de EDL en tiempo $O(n)$. Esto es debido a que cada robot en el certificado se procesa una sola vez y para cada robot se realiza un número constante de operaciones, de esta manera EDL está en NP. Esto se visualiza en el pseudocódigo del Algoritmo 2.1.

2.4. Representación geométrica de EDL

Resulta muy útil visualizar una instancia del problema en el plano cartesiano al representar cada robot i con las coordenadas (x_i, ρ_i) . Esta visualización del problema divide al semiplano superior en dos regiones. Esto se muestra en la Figura 2.1. Dichas regiones dependen de la posición de los datos $d \in [s, t]$.

- Activa: $A(d) = \{(x, \rho) \in \mathbb{R}^2 : |x - d| \leq \rho\}$. Los datos están al alcance de los robots que se encuentran en esta región.
- Inactiva: $B(d) = \{(x, \rho) \in \mathbb{R}^2 : |x - d| > \rho\}$. Los datos están fuera del alcance de los robots que se encuentran en esta región.

Definición 2.2. El conjunto de robots de una instancia I de EDL en la región activa $A(d)$ se denota por $A(d, I)$. Asimismo, se denota $A^-(d) = \{(x_i, \rho_i) \in A(d) | x_i \leq d\}$ y $A^+(d) = \{(x_i, \rho_i) \in A(d) | x_i > d\}$. De manera análoga, los robots de una instancia I en $A^-(d)$ y $A^+(d)$ se denotan por $A^-(d, I)$ y $A^+(d, I)$ respectivamente. Por último, se denota $A([s, t]) = \cup_{d \in [s, t]} A(d)$.

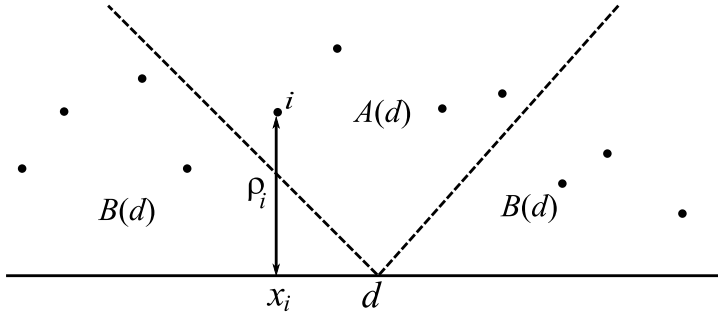


Figura 2.1: En este gráfico se aprecian los robots representados como puntos en el semiplano superior. La posición de los datos d define dos regiones: activa $A(d)$ e inactiva $B(d)$.

En la Figura 2.2 se muestra el alcance de cuatro robots distintos, a, b, c, e . Los robots a y e se encuentran en $B(d)$. El alcance superior de a y el alcance inferior de e , indicado por la punta de sus flechas, no llegan a los datos. Los robots b y c se encuentran en $A(d)$, por lo cuál son capaces de recoger los datos y llevarlos al punto indicado por la punta de sus flechas.

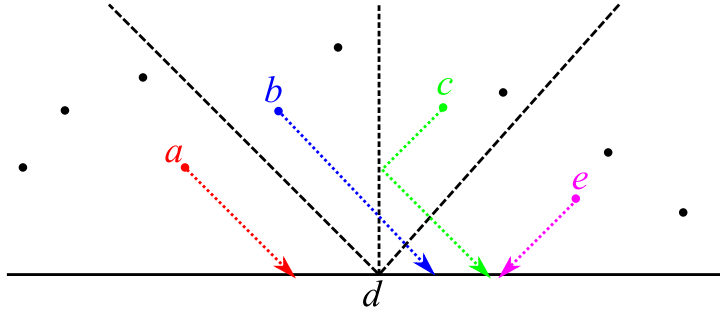


Figura 2.2: Las puntas de las flechas indican hasta dónde pueden llegar los robots. Los robots b y c pueden llegar a los datos y moverlos, mientras que los robots a y e son incapaces de llegar a los datos.

Es importante señalar que los robots en $B(s)$ que se encuentran antes de s quedan descartados. Análogamente, los robots en $B(t)$ que se encuentran después de t quedan descartados. En ambos casos, los robots son incapaces de alcanzar los datos para moverlos dentro de $[s, t]$, véase Figura 2.3. En lo sucesivo se hará la suposición, sin pérdida de generalidad, que las instancias de EDL no tienen robots que caigan en alguno de estos dos casos, sino que solamente hay robots en $A([s, t]) = \cup_{d \in [s, t]} A(d)$.

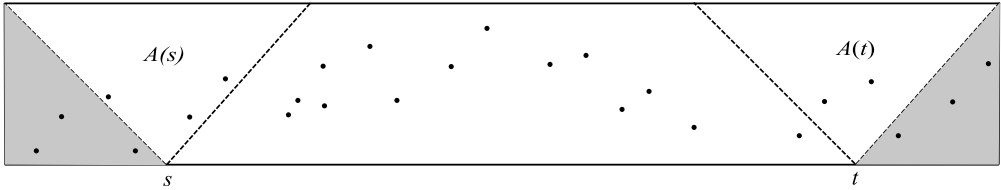


Figura 2.3: Los robots que se encuentran representados en las regiones sombreadas son incapaces de colaborar en el traslado de los datos desde s hasta t .

2.5. Consideraciones adicionales

Una consideración, implícita en la sección 2.3, es que aunque es factible pensar en robots que se mueven simultáneamente, sólo se considera el caso en que los robots se mueven uno por uno, pues no es el objetivo optimizar el tiempo para la realización de la tarea. Hay otras observaciones importantes que hacer que se enuncian como tres lemas.

Lema 2.1. *Ningún robot transmite los datos dos veces.*

Demostración. Es una consecuencia de la definición 2.1. Cuando un robot mueve los datos, agota toda su capacidad para llevarlos lo más adelante posible en un solo movimiento. ■

A continuación se verá la importancia de la definición 2.1 que evita que un robot pueda transmitir los datos dos veces. Se ilustra con un ejemplo.

Ejemplo 2.1. Sea $I = (\{i = (2, 7), j = (5, 5)\}, 0, 10)$ una instancia de EDL. Es claro que el primer robot que debe mover los datos es i . En este caso, se tiene que $a_i = \frac{2-0}{7} = \frac{2}{7}$, y el intervalo que cubre el robot i es $[0, 5]$, de acuerdo con la definición 2.1. Esto quiere decir que el punto máximo en el que puede dejar los datos el robot i es 5. De acuerdo con la definición 2.1, el robot j cubre el intervalo $[5, 10]$ y de esta manera se lograría el objetivo de llevar los datos desde $s = 0$ hasta $t = 10$. El certificado de nuestra instancia es $\langle i, j \rangle$. Ahora bien, si se ignora por un momento las reglas de movimiento dadas en la definición 2.1, y se hace la suposición que los robots se mueven del siguiente modo: primero, el robot i recoge los datos en 0 y los lleva hasta 3. Enseguida, el robot j se mueve hacia 3, recoge los datos y los mueve hasta 4. Después, el robot i se mueve desde su posición en 3, recoge los datos en 4 y los deja en 5. Por último, el robot j se mueve desde su posición en 4, recoge los datos en 5 y los lleva hasta 6, donde agota toda su capacidad, sin poder lograr el objetivo. Esto se ilustra gráficamente en la Figura 2.4.

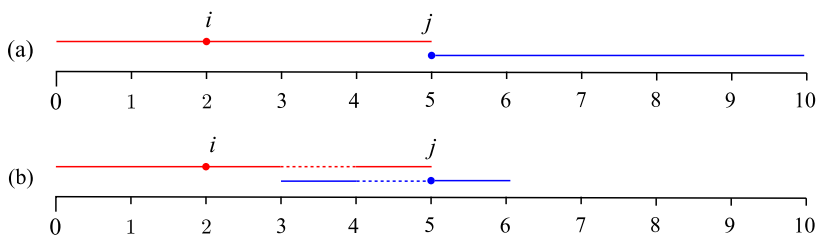


Figura 2.4: En el gráfico se observan las dos situaciones descritas en el ejemplo 2.1. Las líneas punteadas simbolizan los trayectos que los robots recorrieron sin los datos.

Como se aprecia en el ejemplo 2.1, cuando el robot i no lleva los datos a $x_i + (1 - 2a)\rho_i$ en un solo movimiento, otros robots se pueden ver en la necesidad de gastar parte de su capacidad para ir hacia atrás por los datos. Esta capacidad desperdiciada les podría ser útil más adelante, como se ha visto.

Lema 2.2. Sea $I = (\{(x_i, \rho_i)\}_{i=1}^n, s, t)$ una instancia de EDL. Sea $d \in [s, t]$. Sea $M = \max \{x_i + \rho_i \mid (x_i, \rho_i) \in A^-(d, I)\}$. Entonces, el conjunto de robots $A^-(d, I)$ pueden mover los datos desde d a lo más hasta M .

Demostración. Sea i_0 un robot tal que $x_{i_0} + \rho_{i_0} = M$. Aplicando la definición 2.1, dicho robot puede llevar los datos desde d hasta M . Por último, los robots en $A^-(d, I)$ satisfacen $x_i + \rho_i \leq x_{i_0} + \rho_{i_0} = M$ por lo cual es imposible llevar los datos más allá de M . ■

Este último lema sugiere una posible estrategia para resolver EDL. Dada la posición de los datos d , escójase el robot en $A^-(d, I)$ cuyo alcance superior sea mayor y mover. Se repite el proceso. El problema surge cuando $A^-(d, I)$ es vacía. Este algoritmo da la motivación para una variante de EDL que se puede resolver en tiempo polinomial y tiene una buena caracterización. Esto se verá en la sección 2.9.

2.6. Teoría de las Gráficas

En esta sección se introducen algunos conceptos básicos de Teoría de las Gráficas que resultarán útiles para la conversión del problema EDL en un problema de gráficas. Se sigue la notación estándar de [10].

Definición 2.3. Una gráfica G es un par ordenado $(V(G), E(G))$ que consiste en un conjunto de vértices $V(G)$ y un conjunto de aristas $E(G)$, disjunto de $V(G)$, junto con una función de incidencia ψ_G que asocia con cada arista de G una pareja no ordenada de vértices (no necesariamente distintos). Si e es una arista y u y v son vértices tal que $\psi_G(e) = (u, v)$, entonces se dice que e une a u con v , y los vértices u y v son los extremos de e .

Definición 2.4. Sea G una gráfica. Una gráfica F se denomina subgráfica de G si $V(F) \subset V(G)$, $E(F) \subset E(G)$ y ψ_F es la restricción de ψ_G a $E(F)$.

Definición 2.5. Una arista con extremos idénticos se denomina un lazo.

Definición 2.6. Dos o más aristas con el mismo par de extremos se denominan aristas paralelas.

Definición 2.7. Una gráfica es simple si no tiene lazos ni aristas paralelas.

En este trabajo únicamente utilizaremos gráficas simples, por lo que omitiremos el adjetivo *simples*.

Definición 2.8. Dos vértices u y v son *adyacentes* si $\psi_G(e) = (u, v)$ para alguna arista e . Dos aristas e y g son *adyacentes* si $\psi_G(e) \cap \psi_G(g) = u$ para algún vértice u .

Definición 2.9. Una *trayectoria* es una gráfica simple cuyos vértices pueden acomodarse en una secuencia lineal, tal que dos vértices son adyacentes si son consecutivos en la secuencia, de otro modo no son adyacentes.

Definición 2.10. Un *ciclo*, de tres o más vértices, es una gráfica simple cuyos vértices pueden acomodarse en una secuencia cíclica de tal modo que dos vértices son adyacentes si son consecutivos en la secuencia, de otro modo no son adyacentes. El ciclo de k aristas se denota como C_k .

Definición 2.11. El *tamaño* de una trayectoria o ciclo es el número de sus aristas.

Definición 2.12. Una gráfica G es *conexa* si para cada pareja de vértices u y v de la gráfica existe una trayectoria como subgráfica de G que inicia en u y termina en v , es decir, que *conecta* a u con v .

Definición 2.13. Dos gráficas G y F son *isomorfas*, se escribe $G \cong F$, si existen biyecciones $\theta : V(G) \rightarrow V(F)$ y $\phi : E(G) \rightarrow E(F)$ tal que $\psi_G(e) = (u, v)$ si y sólo si $\psi_F(\phi(e)) = (\theta(u), \theta(v))$.

Ahora bien, dado un conjunto V de intervalos cerrados en \mathbb{Q} , se puede construir una gráfica de la siguiente manera: El conjunto de vértices es V y dos vértices son adyacentes si se intersectan como intervalos. Una gráfica que se construye de esta manera se denomina *gráfica de intervalos*. Es importante mencionar que no cualquier gráfica es isomorfa a una gráfica de intervalos, por ejemplo, C_k para $k > 3$ no es una gráfica de intervalos.

2.7. Formulación de EDL como un problema de Teoría de las Gráficas

Existe una conexión natural entre EDL y las gráficas de intervalos. A continuación se enuncian dos modelos que surgen al definir una colección de intervalos para una instancia de EDL de dos maneras distintas:

Definición 2.14. El *modelo simétrico* de una gráfica de intervalos se genera por intervalos i definidos por sus centros x_i y sus radios ρ_i en la forma $[x_i - \rho_i, x_i + \rho_i]$. La gráfica de intervalos asociada con el modelo simétrico será denotada por $G_S(Q)$, donde $Q = \{(x_i, \rho_i), i = 1, \dots, n\}$.

Definición 2.15. El *modelo asimétrico* de una gráfica de intervalos se genera por intervalos i definidos como $[x_i - a_i \rho_i, x_i + (1 - 2a_i) \rho_i]$, y será denotado por $G_A(Q, a)$, donde $Q = \{(x_i, \rho_i), i = 1, \dots, n\}$ y a es un vector en $[0, 1]^n \cap \mathbb{Q}$.

En ambos modelos, Q es el conjunto de robots de una instancia EDL. En la Figura 2.5 se ilustra el modelo asimétrico.

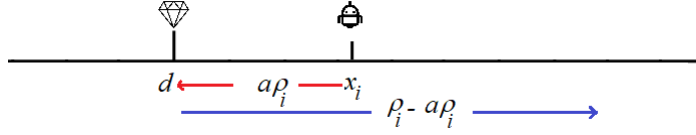


Figura 2.5: En el gráfico se observa un robot y un diamante que representa los datos.

A continuación, se demuestra como EDL es equivalente a un problema de existencia en gráficas de intervalos. El problema de decisión equivalente de Teoría de las Gráficas es el siguiente:

Problema Existencial de Conexidad (PEC)

- Parámetros: una colección finita $Q = \{(x_i, \rho_i), i = 1, \dots, n\}$ de puntos en \mathbb{R}^2 .
- Pregunta: ¿Existe un vector $a = (a_1, \dots, a_n)$ tal que la gráfica de intervalos $G_A(Q, a)$, definida por el modelo asimétrico, sea conexa?

Dos problemas de decisión son equivalentes si para cada instancia de uno existe una instancia del otro, tal que ambas tienen respuesta afirmativa o ambas tienen respuesta negativa.

Teorema 2.3. *EDL y PEC son equivalentes.*

Demostración. Primero se demuestra que para cada instancia (Q, s, t) de EDL existe una instancia de PEC cuya solución se adecúa a la de la instancia EDL. En primer lugar, se añaden dos nuevos puntos $(s, 0)$ y $(t, 0)$. Se llama a este nuevo conjunto Q' , lo cual define una instancia de PEC. Una respuesta positiva a esta instancia tiene asociado un vector a^* de tal modo que $G_A(Q', a^*)$ es una gráfica conexa. De esta manera, existe una trayectoria T en $G_A(Q', a^*)$ que conecta a los vértices definidos por $(s, 0)$ y $(t, 0)$. Los vértices interiores de T corresponden a puntos en Q , es decir, a robots en la instancia EDL. El orden en que los vértices de T son recorridos, desde s hasta t , definen el orden en el que los robots deben moverse. La adyacencia de vértices consecutivos en T garantiza que los robots correspondientes pueden encontrarse en una posición en común para transferir los datos de uno a otro. Así que la existencia de una respuesta afirmativa para una instancia de PEC se traduce en una solución para la instancia correspondiente de EDL.

Queda por demostrar que si la solución de una instancia de PEC es negativa, así lo es para su instancia correspondiente de EDL. La contrapositiva es que cualquier respuesta «sí» a una instancia de EDL se traduce como una respuesta afirmativa de PEC. Para demostrar esto, sea (Q, s, t) una instancia de EDL con una respuesta afirmativa y una secuencia r_{i_1}, \dots, r_{i_k} de robots que llevan los datos de s a t . Esta secuencia de robots determinan un conjunto de parámetros a_{i_1}, \dots, a_{i_k} . Para el resto de los robots, que no participan en el transporte de los datos, se define $a_i = 0$ si $x_i < t$ y $a_i = 1$ si $x_i \geq t$. Se afirma que la gráfica $G_A(Q, a)$ es conexa. Los parámetros a_{i_1}, \dots, a_{i_k} definen un conjunto de intervalos, una trayectoria T' en $G_A(Q, a)$, que cubren el segmento $[s, t]$. Por último, se demuestra que cualquier otro intervalo intersecta a T' . Para un robot (x_i, ρ_i) tal que $x_i < t$ el intervalo correspondiente es $[x_i, x_i + \rho_i]$, si $x_i \geq s$, entonces x_i está en $[s, t]$ así que intersecta algún intervalo de T' ; si $x_i < s$, entonces $x_i + \rho_i \geq s$, y por lo tanto el intervalo intersecta a T' también. El caso $x_i \geq t$ se resuelve de manera análoga. ■

Los siguientes corolarios son una consecuencia directa del teorema anterior.

Corolario 2.4. *PEC es NP-completo.*

Demostración. El teorema 2.3 demuestra que EDL, que es NP-completo, es reducible a PEC y la reducción es polinomial. ■

Es importante recalcar la relevancia de la conexidad de una gráfica de intervalos en relación con la solubilidad de su instancia asociada de EDL. Para cualquier instancia (Q, s, t) de EDL, si existe un vector a en la caja unitaria tal que $G_A(Q', a)$ es conexa, entonces la instancia EDL es soluble. Lo opuesto no es cierto, es decir, dado un vector a si la gráfica $G_A(Q', a)$ es disconexa, eso no implica necesariamente que la instancia asociada EDL no sea soluble, como se verá en el contraejemplo de la Figura 2.6.

Por otro lado, dada una instancia de EDL (Q, s, t) , si para cualquier vector a en la caja unitaria ninguna de las gráficas asociadas $G_A(Q', a)$ es conexa, entonces se asegura que la instancia EDL no es soluble. Esto se enuncia en el siguiente corolario.

Corolario 2.5. *Dada una instancia EDL (Q, s, t) si su gráfica de intervalos simétrica asociada $G_S(Q')$ es disconexa, la instancia no es soluble.*

Demostración. Para cada vector a en la caja unitaria tenemos que

$$[x_i - a\rho_i, x_i + (1 - 2a)\rho_i] \subset [x_i - y_i, x_i + y_i].$$

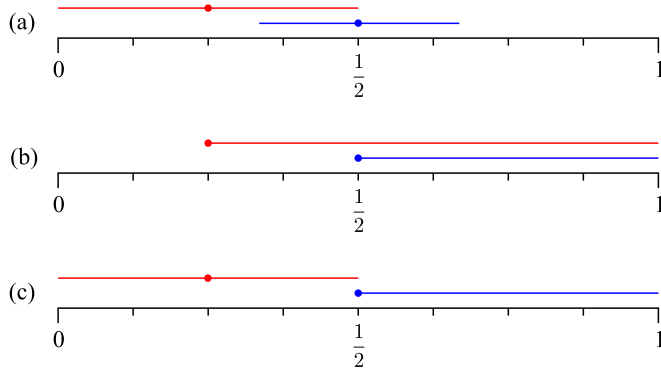


Figura 2.6: En este ejemplo, $Q = \{(\frac{1}{4}, \frac{3}{4}), (\frac{1}{2}, \frac{1}{2})\}$. En a) tenemos $a = (\frac{1}{3}, \frac{1}{3})$, en b) tenemos $a = (0, 0)$, y en c) tenemos $a = (\frac{1}{3}, 0)$. En los primeros dos casos $G_A(Q', a)$ es desconexa, pero en el tercer caso es conexa, por lo que la instancia es en realidad soluble.

Ahora, por la definición de Q' , si $G_S(Q')$ es desconexa, eso significa que para cualquier vector a en la caja unitaria $G_A(Q', a)$ es desconexa también. Esto significa que PEC no es soluble, así que por el Teorema 2.3 EDL tampoco es soluble. ■

2.8. Variante polinomial de EDL: EDLA

En EDL, los robots tienen capacidades restringidas, esto es lo que hace que el problema sea difícil de resolver. Si los robots tuviesen capacidad ilimitada el problema se resuelve trivialmente. Pero si se añade otra restricción al problema, se verá cómo el problema está bien caracterizado y se encuentra en P. En esta variante del problema original, la restricción es que los robots solamente pueden avanzar hacia adelante, así que este nuevo problema se denomina "Entrega de datos en una línea con avance hacia adelante" (EDLA). Formalmente:

Entrega de Datos en una Línea con Avance hacia Adelante (EDLA)

- Parámetros: $I = (\{(x_i, \rho_i)\}_{i=1}^n, s, t)$, donde cada pareja (x_i, ρ_i) representa un robot i con posición $x_i \in \mathbb{Q}$ y capacidad $\rho_i > 0$, $s \in \mathbb{Q}$ representa la posición inicial de los datos, o la salida, y $t \in \mathbb{Q}$, $s < t$, representa la terminal.
- Pregunta: ¿Existe una subsucesión de robots $\langle i_1, \dots, i_k \rangle$ que en ese orden, y avanzando únicamente hacia adelante, puedan mover los datos de s a t ?

Algoritmo 2.2 Algoritmo polinomial para resolver EDLA. En la libreta de Python: EDLA/polinomialEDLA.

```

1: EDLA ( $I$ , robots_disponibles,  $t$ ):
2:   Inicializamos:  $d = s$ , certificado =  $\langle \rangle$ ;
3:   mientras  $S \neq \emptyset$ ;
4:     Calculamos  $S = A^-(d, I) \cap \text{robots\_disponibles}$ ;
5:     Seleccionamos robot  $(x_i, \rho_i) \in S$ ;
6:     Insertamos el robot  $i$  a certificado y lo etiquetamos como usado;
7:     Actualizamos  $d = x_i + \rho_i$ ;
8:   terminar;
   Si  $d \geq t$ : imprimir («Sí»), devolver (certificado);
   en otro caso: imprimir («No»); devolver ( $d$ );

```

El problema está bien caracterizado, en el sentido de Edmonds, pues EDLA está en NP y en coNP, más aún, se dará un algoritmo polinomial para resolver este problema.

Recuérdese que los intervalos en el modelo asimétrico son $\{[x_i - a_i \rho_i, x_i + (1 - 2a_i) \rho_i]\}_{i=1}^n$. En EDLA los robots pueden desplazarse únicamente hacia adelante por lo que $a_i = 0$ para cada robot. En este caso, los intervalos asociados tienen la forma $\{(x_i, x_i + \rho_i)\}_{i=1}^n$. Debe ser claro que EDLA está en NP. Un certificado para un problema Π en EDLA se verifica de la misma manera que uno para EDL. Para verificar que el problema está en coNP, lo que se tiene que verificar es que $G_A(Q', (0, \dots, 0))$ es disconexa. Lo cual se verifica en tiempo polinomial. Así pues, EDLA está bien caracterizado.

A continuación, el algoritmo polinomial para construir el certificado y dar respuesta al problema.

Ahora bien, ¿cuál robot se debe escoger en $A^-(d, I)$? De acuerdo al lema 2.2 los robots en $A^-(d, I)$ pueden llevar los datos hasta $M = \max \{x_i + \rho_i \mid (x_i, \rho_i) \in A^-(d, I)\}$. Esto se puede realizar con un único robot i_0 que cumpla $M = x_{i_0} + \rho_{i_0}$. Sin embargo, en la línea 5 del pseudocódigo del Algoritmo 2.2 no es necesario elegir el robot que maximice $x_i + \rho_i$. Es verdad que el certificado sería más corto, pero se necesitaría incluir una subrutina para elegir el robot que maximice $x_i + \rho_i$. Es suficiente escoger cualquiera y de esa manera se simplifica el código. Ahora bien, ¿cómo se calcula eficientemente $A^-(d, I)$? Para ejecutar esta subrutina es preciso realizar primero un *preprocesamiento*, es decir, realizar un trabajo previo en la instancia antes de utilizar el Algoritmo 2.2. Es necesario ordenar a los robots de acuerdo a $x_i - \rho_i$, lo cual toma tiempo $O(n \log n)$. Para una instancia $I = (\{(x_i, \rho_i)\}_{i=1}^n, s, t)$ de EDLA, sea I_o dicha instancia ordenada. En el Algoritmo 2.3 se describe la subrutina para calcular $A^-(d, I)$.

Aunque el Algoritmo 2.2 tiene tiempo de ejecución $O(n)$, debido al ordenamiento de los robots necesitamos a lo más $O(n \log n)$ operaciones para resolver EDLA. A continuación, el teorema que resume la buena caracterización de EDLA.

Algoritmo 2.3 Subrutina para calcular $A(d, I)$. En la libreta de Python: EDLA/dataTriangle.

```

1: Entrada:  $d, I_o$ ;
2: Inicializamos:  $A^-(d, I) = \{\}$ ,  $A^+(d, I) = \{\}$ , copia_robots = robots_en_ $I_o$ ;
3: para robot en copia_robots:
4:   si  $x_i + \rho_i \leq d$ : quitar robot de robots_en_ $I_o$ ; // se quita un robot inútil.
5:   en otro caso:
6:     si  $x_i \leq d$ : insertar  $(x_i, \rho_i) \in A^-(d, I)$ ;
7:     en otro caso:
8:       si  $x_i - \rho_i < d$ : insertar  $(x_i, \rho_i) \in A^+(d, I)$ ; // el robot está en la región activa y adelante de  $d$ .
9:       en otro caso:
10:      terminar; // el robot está adelante de  $d$  pero inactivo. El resto de los robots en la lista también, debido a que están ordenados por  $x_i - \rho_i$ 
11: devolver  $A^-(d, I), A^+(d, I)$ ;

```

Teorema 2.6. Sea $I = (\{(x_i, \rho_i)\}_{i=1}^n, s, t)$ una instancia de EDLA. O bien existe una subsucesión de robots que pueden llevar los datos de s a t , avanzando únicamente hacia adelante, o bien existe $d_0 \in [s, t]$ tal que $A^-(d, I) = \emptyset$, pero no ambos.

Demostración. El Algoritmo 2.2, o bien encuentra la subsucesión de robots que pueden llevar los datos de s a t , avanzando únicamente hacia adelante, o bien se detiene cuando encuentra una $A^-(d, I)$ vacía. ■

La libreta de Python **EDLA.ipynb** para acceder al código: <https://colab.research.google.com/drive/14Lo202rS7c5WFDpE5YJKPTeZsnPTrJt1?usp=sharing>

2.9. Regiones activas vacías y la conexidad de las gráficas de intervalos $G_S(Q')$

Este capítulo concluye con un teorema que resume la relación que existe entre conexidad de la gráfica de intervalos y regiones activas vacías de instancias de EDL.

Teorema 2.7. Sea $I = (Q, s, t)$ una instancia de EDL. Sea $G_S(Q')$ la gráfica de intervalos asociada con el modelo simétrico. Entonces, $G_S(Q')$ es desconexa si y sólo si existe $p \in (s, t)$ tal que $A(p) = \emptyset$.

Demostración. Supóngase primero que $G_S(Q')$ es desconexa. Esto implica que existe $p \in (s, t)$ tal que $p \cap [x_i - \rho_i, x_i + \rho_i] = \emptyset$ para todo $i = 1, \dots, n$. Se hará la prueba por contradicción. Supóngase que existe un robot j tal que $(x_j, \rho_j) \in A(p)$. Ahora bien, se

tienen dos casos: $x_j < p$ o $p < x_j$. Considérese el primer caso. Como $(x_j, \rho_j) \in A(p)$ por hipótesis, entonces $|x_j - p| < \rho_j$, por lo cual $p - x_j < \rho_j$ lo que implica que $p < x_j + \rho_j$, esto quiere decir que $p \in (x_j, x_j + \rho_j)$ por lo que $p \cap (x_j - \rho_j, x_j + \rho_j) \neq \emptyset$ lo cual es una contradicción. El caso $p < x_j$ se demuestra de manera similar.

Ahora supóngase que existe $p \in (s, t)$ tal que $A(p) = \emptyset$. Tómese cualquier robot (x_i, ρ_i) , suponer que $p < x_j$ (el caso $p > x_j$ se demuestra de manera análoga). Ahora bien, por cuanto $A(p) = \emptyset$ eso quiere decir que $|x_j - p| > \rho_j$, así que $x_j - p > \rho_j$, entonces $x_j - \rho_j > p$ es decir $p \cap (x_j - \rho_j, x_j) = \emptyset$ lo cual implica que $p \cap (x_j - \rho_j, x_j + \rho_j) = \emptyset$ para todo $i = 1, \dots, n$, en particular esto significa que la gráfica de intervalos es disconexa. ■

Este teorema tiene su versión para EDLA que se enuncia como un corolario del mismo, sin demostración.

Corolario 2.8. Sea $I = (Q, s, t)$ una instancia de EDLA. Sea $G_A(Q', a = (0, 0, \dots, 0))$ la gráfica de intervalos asociada con el modelo asimétrico. Entonces, $G_A(Q', a = (0, 0, \dots, 0))$ es disconexa si y sólo si existe $p \in (s, t)$ tal que $A^-(p, I) = \emptyset$.

Capítulo 3

Modelo probabilístico del problema EDL

3.1. Probabilidad

En esta sección se introducen algunos conceptos básicos de la teoría de la probabilidad necesarios para los resultados de este capítulo.

Definición 3.1. Sea Ω un conjunto y sea 2^Ω el conjunto de todos los subconjuntos de Ω . Un subconjunto $\mathcal{F} \subset 2^\Omega$ es una σ -álgebra si satisface las siguientes tres propiedades

1. $\Omega \in \mathcal{F}$.
2. Si $E \in \mathcal{F}$ entonces $E^c \in \mathcal{F}$ también (donde $E^c = \Omega - E$).
3. Si $E_i \in \mathcal{F}$ para $i = 1, 2, 3, \dots$ entonces también $\cup E_i \in \mathcal{F}$.

Los elementos de \mathcal{F} se denominan *conjuntos medibles*. La pareja (Ω, \mathcal{F}) se denomina un *espacio medible*.

Es de notar que al utilizar las leyes de De Morgan se tiene que $(\cup E_i^c)^c = \cap E_i$. Por lo cual, de la propiedad 3 de la definición anterior se obtiene también lo siguiente:

4. Si $E_i \in \mathcal{F}$ para $i = 1, 2, 3, \dots$ entonces también $\cap E_i \in \mathcal{F}$.

Definición 3.2. Dada una colección \mathcal{A} de subconjuntos de Ω , se define $\sigma(\mathcal{A})$ como la σ -álgebra más pequeña que la contiene. Entonces, si \mathcal{F} es otra σ -álgebra y $\mathcal{A} \subset \mathcal{F}$, se tiene que $\sigma(\mathcal{A}) \subset \mathcal{F}$.

Definición 3.3. Sea $\mathcal{B}_0 = \{[a, b) : a, b \in \mathbb{R}\}$. Entonces $\sigma(\mathcal{B}_0) = \mathcal{B}$ se denomina la σ -álgebra de Borel en \mathbb{R} .

Definición 3.4. Un espacio de probabilidad es una terna (Ω, \mathcal{F}, P) , donde el *espacio muestral* Ω es el conjunto de todos los posibles resultados de un experimento aleatorio.

El *espacio de eventos* \mathcal{F} es una σ -álgebra de los subconjuntos de Ω . Por último, P es una función de probabilidad $P : \mathcal{F} \rightarrow [0, 1]$. Esta terna cumple con los siguientes tres axiomas de Kolmogorov:

1. $0 \leq P(E) \leq 1$ para todo $E \in \mathcal{F}$. A este número $P(E)$ se le denomina la *probabilidad* del evento E .
2. $P(\Omega) = 1$.
3. Para cualquier conjunto numerable en \mathcal{F} de *eventos mutuamente excluyentes* E_1, E_2, \dots (eventos para los cuales $E_i \cap E_j = \emptyset$ si $i \neq j$), se satisface

$$P(\cup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} P(E_i).$$

Lema 3.1. Sean E y F dos eventos en el espacio de eventos \mathcal{F} . Se tiene entonces que

- $P(E^c) = 1 - P(E)$.
- Si $E \subset F$ entonces $P(E) \leq P(F)$.
- $P(E \cup F) = P(E) + P(F) - P(E \cap F)$.

Definición 3.5. Sean E y F dos eventos en el espacio de eventos \mathcal{F} tal que $P(F) > 0$. La *probabilidad condicional* de que el evento E ocurra dado que el evento F ha ocurrido se define como

$$P(E|F) = \frac{P(E \cap F)}{P(F)}.$$

Teorema 3.2 (Ley de la Probabilidad Total). Sea E un evento en el espacio de eventos \mathcal{F} y sea $\{F_i \in \mathcal{F}\}$ una partición finita o numerable de Ω , es decir, $\Omega = \cup F_i$. Se tiene entonces que

$$P(E) = \sum_i P(E|F_i) P(F_i).$$

Definición 3.6. Dos eventos E y F se dicen *independientes* si y sólo si

$$P(E \cap F) = P(E) P(F).$$

Definición 3.7. Sean $(\Omega_1, \mathcal{F}_1)$, $(\Omega_2, \mathcal{F}_2)$ dos espacios medibles. Una función $X : \Omega_1 \rightarrow \Omega_2$ se dice que es *medible* si para todo $B \in \mathcal{F}_2$ se tiene que

$$X^{-1}(B) = \{\omega : X(\omega) \in B\} \in \mathcal{F}_1.$$

Definición 3.8. Sea (Ω, \mathcal{F}, P) un espacio de probabilidad. Una *variable aleatoria real* es una función medible $X : \mathcal{F} \rightarrow \mathbb{R}$ tal que

$$P(X \in S) = P\{\omega \in \Omega | X(\omega) \in S\}$$

con $S \subset \mathbb{R}$, donde \mathbb{R} está equipado con la σ -álgebra de Borel.

Debido a que en este trabajo únicamente se utilizan variables aleatorias reales, en lo sucesivo se omite la mención del contradominio. Asimismo, en lo sucesivo la notación $X(\omega) \leq t$ se simplificará como $X \leq t$, lo cual se refiere a los valores en la imagen de X menores a un valor real t . Esto es una práctica estándar.

Definición 3.9. Sea X una variable aleatoria. Cuando la imagen de X es un conjunto numerable, X se denomina una *variable aleatoria discreta*. Cuando la imagen de X es la unión de una colección de intervalos propios en \mathbb{R} , X se denomina una *variable aleatoria continua*.

Definición 3.10. Sea X una variable aleatoria, su *función de distribución de probabilidad* se define como

$$F_X(t) = P(X \leq t) \quad -\infty < t < \infty.$$

Lema 3.3. *Propiedades de $F_X(t)$*

- $0 \leq F_X(t) \leq 1$.
- $\lim_{t \rightarrow -\infty} F_X(t) = 0$ y $\lim_{t \rightarrow \infty} F_X(t) = 1$.
- Es una función monótona no decreciente.
- Es una función continua por la derecha.

Definición 3.11. Sea X una variable aleatoria discreta, su *función de probabilidad en un punto t* se define como

$$p_X(t) = P(X = t).$$

Definición 3.12. Sea X una variable aleatoria continua tal que $F_X(t)$ es su función de distribución de probabilidad, y $F_X(t)$ es diferenciable. La *función de densidad* de X se define como

$$f_X(t) = \frac{d}{dt} F_X(t).$$

Lema 3.4. *Propiedades de $f_X(t)$*

- $f_X(t) \geq 0$.
- $\int_{-\infty}^{\infty} f(t) dt = 1$.
- $P(a \leq t \leq b) = \int_a^b f(t) dt$.

Definición 3.13. Sea X una variable aleatoria discreta. La *esperanza* o el *valor esperado* de X se define como

$$E(X) = \sum_{t \in \Omega} t p_X(t).$$

Definición 3.14. Sea X una variable aleatoria continua. La *esperanza* o el *valor esperado* de X se define como

$$E(X) = \int_{-\infty}^{\infty} t f_X(t) dt.$$

Definición 3.15. Sea X una variable aleatoria. La *varianza* de X se define como

$$\text{Var}(X) = E(X^2) - (E(X))^2.$$

Teorema 3.5 (Desigualdad de Chebyshev). *Sea X una variable aleatoria tal que $E(X) < \infty$, entonces para todo $k > 0$ se tiene que*

$$P(|X - E(X)| \geq k) \leq \frac{\text{Var}(X)}{k^2}.$$

Definición 3.16. Se dice que X es una variable aleatoria con distribución de probabilidad uniforme en $[a, b]$ si

$$F_X(t) = \begin{cases} 0 & \text{si } t < a \\ \frac{t-a}{b-a} & \text{si } a \leq t \leq b \\ 1 & \text{si } b \leq t \end{cases}.$$

Su función de densidad es

$$f_X(t) = \begin{cases} \frac{1}{b-a} & \text{si } a \leq t \leq b \\ 0 & \text{en otro caso} \end{cases}.$$

A continuación se define la función gamma, que aparece en varias funciones de distribución de probabilidad, la cual extiende el concepto de factorial a los números reales y complejos.

Definición 3.17. Sea z un número complejo. Se define la función gamma de z como

$$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t} dt.$$

Si $n \in \mathbb{Z}^+$ entonces $\Gamma(n) = (n-1)!$.

Definición 3.18. Se dice que X es una variable aleatoria con distribución de probabilidad beta con parámetros (r, s) si la función de densidad de X está dada por

$$f_X(t) = \frac{1}{\beta(r, s)} t^{r-1} (1-t)^{s-1},$$

donde $\beta(r, s) = \frac{\Gamma(r)\Gamma(s)}{\Gamma(r+s)}$.

En lo sucesivo, se omite la mención explícita del espacio de probabilidad. Al hablar de eventos o de variables aleatorias, se hace la suposición implícita de la existencia de (Ω, \mathcal{F}, P) . Por otro lado, si X es una variable aleatoria, utilizaremos la notación \hat{X} para fijar la variable en un valor real. Por último, una convención de notación importante en este trabajo es que se denota el logaritmo natural (o logaritmo base e) de un número x como $\log x$.

3.2. Modelo probabilístico para EDL

En esta sección se expone un modelo probabilístico para describir instancias aleatorias de EDL. La motivación para trabajar con instancias aleatorias es identificar el umbral de transición de fase en la solubilidad de instancias de EDL obtenidas con este modelo, análogo a la transición de fase de k -satisfacibilidad.

En el modelo probabilístico propuesto en este trabajo se sigue el principio de parsimonia. Las ubicaciones de los robots $\{x_i\}_{i=1}^n$ se modelan mediante variables aleatorias independientes y con distribución uniforme en $[0, 1]$. La restricción al intervalo unitario es sin pérdida de generalidad. Las capacidades de los robots $\{\rho_i\}_{i=1}^n$ se modelan mediante variables aleatorias independientes y con distribución uniforme en $[0, r]$. El valor de r representa la máxima capacidad permitida. Para algún $\varepsilon \in (0, 1)$ se define $s = \varepsilon$ y $t = 1 - \varepsilon$. Dicha elección es arbitraria pero en términos asintóticos resulta intrascendente. En el análisis asintótico de este capítulo se tiene que $r = r(n) \rightarrow 0$ cuando $n \rightarrow \infty$. Así pues, el estudio se centra en determinar cuál es la relación de n y r para modelar la probabilidad de obtener instancias solubles e instancias no solubles con alta probabilidad. Por último, al modelar instancias aleatorias de EDL y al fijar un vector unitario $\{a_i\}_{i=1}^n$, $a_i \in [0, 1] \cap \mathbb{Q}$, se modelan también gráficas de intervalos aleatorias, como se discute en la siguiente sección.

3.3. Gráficas de intervalos aleatorias

Las gráficas aleatorias fueron introducidas por primera vez por Gilbert [34], pero el trabajo de Erdős y Renyi [28] sentó las bases para el estudio de su evolución. Un estudio exhaustivo en gráficas aleatorias se puede encontrar en [8]. Más tarde, Cohen estudió la probabilidad asintótica de que una gráfica aleatoria sea una gráfica de intervalos unitarios [17]. Scheinerman, por su parte, replicó las ideas de Erdős y Renyi para estudiar la evolución de las gráficas de intervalos aleatorias [56].

Scheinerman estudió gráficas de intervalos aleatorias con el modelo simétrico. En su trabajo definió dos tipos de variables aleatorias, los centros $\{x_i\}_{i=1}^n$ y los radios de los intervalos $\{\rho_i\}_{i=1}^n$. Los centros con distribución uniforme en $[0,1]$ y los radios con distribución uniforme en el intervalo $[0, r]$ con $r < 1$. De este modo, los intervalos son construidos como $[x_i - \rho_i, x_i + \rho_i]$. En el modelo asimétrico, si se fijan los parámetros $\{a_i\}_{i=1}^n$ con $a_i \in [0, 1]$, se obtienen intervalos aleatorios del tipo $\{[x_i - a_i \rho_i, x_i + (1 - 2a_i) \rho_i]\}_{i=1}^n$.

A partir de la siguiente sección, se discuten los resultados en torno al análisis asintótico del modelo probabilístico que se ha descrito. En primer lugar, se dará a conocer el primer intento para describir el fenómeno de transición de fase para EDL, que dio como resultado un algoritmo aleatorio. En la sección subsecuente se describe porqué este primer intento no puede describir correctamente el fenómeno de transición de fase, en cambio, se da una cota inferior para el conjeturado umbral de transición. En la penúltima sección se discute un resultado que se refiere a la probabilidad de la intersección de dos intervalos en una gráfica de intervalos aleatoria. En la última sección se demuestra que EDLA tiene un umbral de transición de fase, el cual es una cota superior para el conjeturado umbral de transición de fase para EDL.

3.4. Avance promedio de los robots en instancias aleatorias de EDL

En esta sección se hablará del primer intento realizado en este trabajo para encontrar el umbral de transición de fase de EDL. Dada una instancia aleatoria de EDL $I = (\{(x_i, \rho_i)\}_{i=1}^n, s, t)$, interesa saber cuál sería el valor esperado del avance de un robot para mover los datos ubicados en la posición d . El robot se escoge aleatoriamente en $A(d, I)$. Aquellos robots (x_i, ρ_i) que se encuentren en la zona activa $A(d)$ son los únicos que tienen la capacidad de mover los datos ubicados en la posición d hasta una nueva posición d' . Se define de esta manera la variable aleatoria $\delta = d' - d$, *el avance*. Se procede ahora a determinar el valor esperado de δ .

Es claro que $P(\delta \in (-\infty, \infty)) = P(\delta \in [0, r])$, por lo que si se desea calcular la distribución de probabilidad de δ se debe notar que

$$P(\delta < 0) = 0$$

$$P(\delta < r) = 1.$$

En este sentido, el caso de interés es $P(\delta < x)$ cuando $x \in (0, r)$. Sea $x \in (0, r)$, se puede calcular $P(\delta < x)$ como la probabilidad de que un robot aparezca en la región sombreada de la Figura 3.1. Debe ser claro que los robots en dicha región sombreada pueden realizar un avance $\delta < x$ a partir de la posición d . La probabilidad es entonces la fracción del área sombreada con respecto a la región activa en d :

$$P(\delta < x) = \frac{r^2 - (r-x)^2}{r^2} = 1 - \frac{(r-x)^2}{r^2}$$

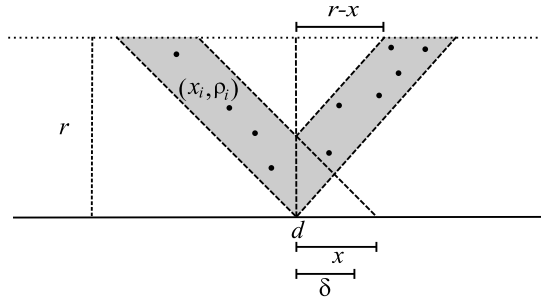


Figura 3.1: En la zona sombreada aparecen los robots que pueden mover los datos, a partir de la posición d , un trayecto de tamaño δ no mayor a x .

Se obtiene de esta manera la distribución de probabilidad de δ

$$F_{\delta}(x) = P(\delta < x) = \begin{cases} 0 & x \leq 0 \\ 1 - \frac{(r-x)^2}{r^2} & x \in (0, r) \\ 1 & x \geq r \end{cases}$$

Al derivar $F_{\delta}(x)$ se obtiene la función de densidad $f_{\delta}(x)$, es decir,

$$f_{\delta}(x) = \frac{2(r-x)}{r^2}.$$

Se calcula el valor esperado de δ :

Algoritmo 3.1 Algoritmo aleatorio para EDL.

-
- 1: *Inicializar:* $d=s$, *subsucesión* = $\langle \rangle$, $\text{robots_disponibles} = \text{robots}(I)$;
 - 2: Calcular $A(d)$;
 - 3: **mientras** $A(d) \cap \text{robots_disponibles} \neq \emptyset$;
 - 4: *Se escoge aleatoriamente robot* $i = (x_i, \rho_i) \in A(d) \cap \text{robots_disponibles}$;
 - 5: *Se inserta el robot* i *a subsucesión* *y se quita de* $\text{robots_disponibles}$;
 - 6: *Se actualiza* $d = x_i + (1 - 2a)\rho_i$ *donde* $a = 0$ *si* $x_i \leq d$ *y* $a = (x_i - d)/\rho_i$ *si* $d < x_i$;
 - 7: **terminar**;
- Si** $d \geq t$: **imprimir** («Sí»), **devolver** (*subsucesión*);
- en otro caso:** *Reinicializar y repetir el algoritmo hasta resolver afirmativamente, o hasta agotar el número de intentos.*
-

$$\begin{aligned}
 E(\delta) &= \int_0^r f_\delta(x) \cdot x dx = \int_0^r \frac{2(r-x)}{r^2} \cdot x dx \\
 &= \frac{2}{r^2} \int_0^r (r-x)x \cdot dx = \frac{2}{r^2} \left[\frac{rx^2}{2} - \frac{x^3}{3} \right] \Big|_0^r \\
 &= \frac{2}{r^2} \left[\frac{r^3}{2} - \frac{r^3}{3} \right] = \frac{r}{3}
 \end{aligned}$$

Por último, se propone un algoritmo aleatorio (Algoritmo 3.1) para buscar respuesta a una instancia de EDL. Este algoritmo entra en una categoría que se denomina *algoritmos Monte Carlo con sesgo afirmativo*. Estos algoritmos deben el nombre "Monte Carlo" al hecho de que tienen alguna etapa aleatoria (en alusión al casino de Monte Carlo). Al decir "con sesgo afirmativo" se refiere a que cuando clasifican una instancia como afirmativa, la respuesta es correcta. Pero cuando clasifican una instancia como negativa, existe cierta probabilidad de que la respuesta sea incorrecta. El algoritmo comienza cuando $d = s$ y se escoge al azar un robot en $A(d)$ que no se haya utilizado previamente. Se utiliza el robot y se actualiza d . Se repite este proceso hasta que no haya robots disponibles en $A(d)$ o hasta que se haya llegado a la terminal t . Al tratarse de una *búsqueda aleatoria*, el algoritmo se puede reinicializar y repetir hasta que se encuentre la solución o hasta que se hayan realizado un número de intentos predefinido por el usuario.

Si se consideran instancias aleatorias de EDL que se resuelven afirmativamente con el algoritmo propuesto, ¿cuántos robots $n_0(r, s, t)$ se utilizarían en promedio para llevar la carga de s a t ? La idea es cubrir el intervalo $[s, t]$ con avances de tamaño $\frac{t}{3}$, por lo cual:

$$n_0(r, s, t) \approx \frac{t-s}{\frac{1}{3}r} = \frac{3(t-s)}{r}.$$

En otras palabras, en promedio se necesitan tres robots en cada región activa. Ahora bien, una región activa representa la mitad del área del rectángulo que inscribe a la región activa, así que se necesitan en realidad seis robots en promedio en estos rectángulos de base $2r$ y altura r . A partir de este razonamiento, para valores fijos de r , s y t , se propuso inicialmente que se necesitarían alrededor de $\frac{6(t-s)}{r}$ robots para que una instancia aleatoria sea soluble. Esta intuición acerca del umbral de transición de fase para EDL resultó ser errónea, como se verá en la siguiente sección. No obstante, en la experimentación se observó que el tamaño de los certificados era precisamente $n_0(r, s, t)$ en promedio.

3.5. Cota inferior para el umbral de transición de fase de EDL

Como se discutió al final del primer capítulo, se ha observado que existe un fenómeno de transición de fase en modelos aleatorios para ciertos problemas combinatorios. En el problema EDL dicho fenómeno emerge en la relación entre n y r . Para ciertas combinaciones de valores de n y r se modelan instancias solubles con alta probabilidad. Para otras combinaciones de valores de n y r se modelan instancias insolubles con alta probabilidad. La zona crítica en la relación de n y r es donde se transiciona de la solubilidad a la insolubilidad.

A continuación, se menciona un resultado de Scheinerman [56] que establece que el umbral no puede ser $r = \frac{6(t-s)}{n}$.

Teorema 3.6 (Scheinerman). *Si $nr \rightarrow c$, donde c es una constante real positiva, entonces el número de componentes en casi todas las gráficas de intervalos del modelo simétrico es $ne^{-c} + o(n)$.*

Lo que este teorema dice es que el umbral para el modelo simétrico no puede ser de la forma $r = \frac{c}{n}$. En ese punto, las gráficas de intervalos simétricas son desconexas casi con certeza. En el corolario 2.5 quedó establecido que dada una instancia $I = \{Q, s, t\}$ de EDL, si $G_S(Q')$ es desconexa, entonces I no es soluble. Ahora bien, Scheinerman [56] demostró también que existe una transición de fase para la conexidad en las gráficas de intervalos aleatorias con el modelo simétrico en el intervalo unitario ($s = 0$, $t = 1$). Se enuncia a continuación su teorema.

Teorema 3.7 (Scheinerman). *Suponer que con el modelo simétrico $c = \lim_{n \rightarrow \infty} \frac{nr}{\log n}$ existe. Si $c < 1$ entonces casi todas las gráficas de intervalos tienen vértices aislados, mientras que si $c > 1$ casi todas las gráficas de intervalos son conexas.*

Debe ser claro que si la instancia es desconexa en el modelo de Scheinerman, lo será también en EDL. Así pues, se puede utilizar el resultado de Scheinerman para dar una cota inferior al umbral para EDL, es decir, si $r < \frac{\log n}{n}$ las instancias aleatorias tendrán gráficas de intervalos $G_S(Q')$ desconexas con alta probabilidad, y por lo tanto no serán

solubles.

3.6. Probabilidad de intersección de dos intervalos aleatorios

Esta sección inicia con algunos lemas previos al resultado principal, presentado en el teorema 3.12, que consiste en el cálculo de la probabilidad de que dos intervalos aleatorios se intersecten.

Lema 3.8. *Si X y Y son variables aleatorias independientes con distribución uniforme en $[0, 1]$, se tiene que*

$$F_{X-Y}(t) = \begin{cases} \frac{t^2}{2} + t + \frac{1}{2} & \text{si } -1 \leq t \leq 0 \\ \frac{-t^2}{2} + t + \frac{1}{2} & \text{si } 0 < t < 1 \\ 0 & \text{si } t < -1 \\ 1 & \text{si } t > 1 \end{cases}$$

Demostración. En primer lugar

$$f_X(x) = \begin{cases} 1 & \text{si } 0 < x < 1 \\ 0 & \text{en otro caso} \end{cases}, \quad f_Y(y) = \begin{cases} 1 & \text{si } 0 < y < 1 \\ 0 & \text{en otro caso} \end{cases}.$$

En segundo lugar

$$\begin{aligned} F_{X-Y}(t) &= P(X - Y \leq t) \\ &= \iint_{x-y \leq t} f_X(x) f_Y(y) dx dy \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{t+y} f_X(x) f_Y(y) dx dy \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{t+y} f_X(x) dx f_Y(y) dy \\ &= \int_{-\infty}^{\infty} F_X(t+y) f_Y(y) dy \end{aligned}$$

Ahora bien, $f_Y(y) = 1$ si $0 < y < 1$ y $f_Y(y) = 0$ si $y \in (-\infty, 0] \cup [1, \infty)$ entonces la expresión anterior quedan como $F_{X-Y}(t) = \int_0^1 F_X(t+y) dy$. Por otro lado, si se desea calcular $\frac{dF_{X-Y}(t)}{dt}$ se puede aplicar la regla de la integral de Leibniz para intercambiar el orden de integración y diferenciación, de modo que

$$\frac{d}{dt} F_{X-Y}(t) = f_{X-Y}(t) = \frac{d}{dt} \int_0^1 F_X(t+y) dy = \int_0^1 f_X(t+y) dy.$$

Ahora bien, dado que $X - Y \in [-1, 1]$ entonces $-1 < t < 1$. Además, $f_X(t+y) = 1$ si $0 < t+y < 1$. Por lo cual, al intersectar $0 < y < 1$ con $-t < y < 1-t$ se obtiene $-t < y < 1$ cuando $-1 \leq t \leq 0$, y se obtiene $0 < y < 1-t$ cuando $0 < t < 1$. Se tiene entonces que

$$f_{X-Y}(t) = \begin{cases} \int_{-t}^1 f_X(t+y) dy = \int_{-t}^1 dy & \text{si } -1 \leq t \leq 0 \\ \int_0^{1-t} f_X(t+y) dy = \int_0^{1-t} dy & \text{si } 0 < t < 1 \end{cases}.$$

Por lo anterior

$$f_{X-Y}(t) = \begin{cases} 1+t & \text{si } -1 \leq t \leq 0 \\ 1-t & \text{si } 0 < t < 1 \\ 0 & \text{en otro caso} \end{cases}.$$

La función de densidad $f_{X-Y}(t)$ es simétrica y triangular como lo muestra la Figura 3.2.

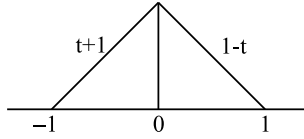


Figura 3.2: Función de densidad de $X - Y$.

Por último, al integrar $\int_{-1}^t (\tau+1) d\tau = \frac{\tau^2}{2} + t + \frac{1}{2}$ y $\frac{1}{2} + \int_0^t (-\tau+1) d\tau = \frac{-\tau^2}{2} + t + \frac{1}{2}$ se obtiene el resultado esperado. ■

Lema 3.9. Sean $\mathcal{I}_i = [x_i - a\rho_i, x_i + (1-2a)\rho_i]$, $\mathcal{I}_j = [x_j - a\rho_j, x_j + (1-2a)\rho_j]$ intervalos aleatorios con el modelo asimétrico. Se define $\alpha = a\rho_i + (1-2a)\rho_j$, $\beta = a\rho_j + (1-2a)\rho_i$. Entonces

$$-1 \leq \alpha \leq 1 \quad \text{y} \quad -1 \leq \beta \leq 1$$

Demostración. Se hace la demostración para α . Nótese primero que si $a \in [\frac{1}{2}, 1]$ entonces $(1-2a) \leq 0$ y si $a \in [0, \frac{1}{2}]$ entonces $(1-2a) \geq 0$. Así que se tienen dos casos.

1. Si $(1-2a) \leq 0$ entonces $\alpha = a\rho_i + (1-2a)\rho_j \leq ar + (1-2a)\rho_j \leq ar \leq 1$ puesto que $0 \leq a, r \leq 1$. También

$$-1 \leq (1-2a)r \leq (1-2a)\rho_j \leq a\rho_i + (1-2a)\rho_j = \alpha.$$

2. Si $(1 - 2a) \geq 0$ entonces $\alpha = a\rho_i + (1 - 2a)\rho_j \leq ar + (1 - 2a)r = (1 - a)r \leq 1$ puesto que $0 \leq a, r \leq 1$. También

$$-1 < 0 \leq a\rho_i + (1 - 2a)\rho_j.$$

La demostración para β es análoga. ■

Lema 3.10. Sean $\alpha = a\rho_i + (1 - 2a)\rho_j$, $\beta = a\rho_j + (1 - 2a)\rho_i$. Entonces se tiene que:

1. Si $a \in [0, \frac{1}{2}]$ entonces $\alpha, \beta \geq 0$.
2. Si $a \in [\frac{1}{2}, 1]$ entonces $\alpha\beta \leq 0$.

Demostración. Dado que $(1 - 2a) \geq 0$ si $a \in [0, \frac{1}{2}]$, entonces por definición de α y β se tiene que $\alpha, \beta \geq 0$. Por otro lado, $(1 - 2a) \leq 0$ si $a \in [\frac{1}{2}, 1]$, en cuyo caso resulta posible que $\alpha \leq 0$ si $a\rho_i \leq |(1 - 2a)|\rho_j$ y $\beta \leq 0$ si $a\rho_j \leq |(1 - 2a)|\rho_i$. Ahora bien, supóngase que $\alpha, \beta < 0$, entonces $\alpha + \beta < 0$, es decir

$$\begin{aligned} a\rho_i + (1 - 2a)\rho_j + a\rho_j + (1 - 2a)\rho_i &< 0 \\ (1 - a)\rho_j + (1 - a)\rho_i &< 0, \end{aligned}$$

lo cual es una contradicción dado que $(1 - a), \rho_i, \rho_j \geq 0$, es decir, es imposible que α y β sean ambos negativos. Si α es negativo, necesariamente β es positivo y viceversa. Esto concluye la demostración. ■

Lema 3.11. Sean

$$\begin{aligned} \mathcal{I}_i &= [x_i - a\rho_i, x_i + (1 - 2a)\rho_i] \\ \mathcal{I}_j &= [x_j - a\rho_j, x_j + (1 - 2a)\rho_j] \end{aligned}$$

intervalos aleatorios con el modelo asimétrico. Entonces, el evento $\mathcal{I}_i \cap \mathcal{I}_j \neq \emptyset$ es equivalente a

$$-a\rho_j - (1 - 2a)\rho_i \leq x_i - x_j \leq a\rho_i + (1 - 2a)\rho_j.$$

Demostración. Dos intervalos $[a, b]$ y $[c, d]$ se intersectan si y sólo si $c \leq b$ y $a \leq d$, entonces $\mathcal{I}_i \cap \mathcal{I}_j \neq \emptyset$ si y sólo si las siguientes desigualdades se cumplen.

$$x_i - a\rho_i \leq x_j + (1 - 2a)\rho_j \quad \text{y} \quad x_j - a\rho_j \leq x_i + (1 - 2a)\rho_i$$

lo cual es equivalente a

$$x_i - x_j \leq a\rho_i + (1 - 2a)\rho_j \quad (3.1)$$

$$x_j - x_i \leq a\rho_j + (1 - 2a)\rho_i. \quad (3.2)$$

■

Ahora se procede con el resultado principal de esta sección.

Teorema 3.12. Sean \mathcal{I}_i e \mathcal{I}_j dos intervalos asimétricos, aleatorios e independientes con $r \leq 1$ y $a \in [0, 1]$. Entonces

$$P(\mathcal{I}_i \cap \mathcal{I}_j \neq \emptyset) \leq (1 - a)r - \frac{r^2}{6}(4a^2 - 5a + 2).$$

Demostración. Primeramente, si x_i y x_j son variables aleatorias independientes con distribución uniforme en $[0, 1]$, se tiene por el lema 3.9 que

$$P(x_i - x_j \leq t) = \begin{cases} \frac{t^2}{2} + t + \frac{1}{2} & \text{si } -1 \leq t \leq 0 \\ -\frac{t^2}{2} + t + \frac{1}{2} & \text{si } 0 \leq t \leq 1 \\ 0 & \text{si } t < -1 \\ 1 & \text{si } t > 1 \end{cases} \quad (3.3)$$

Por otro lado, la función de densidad para ρ_i y ρ_j está dada por

$$f_{\rho_i}(\rho) = f_{\rho_j}(\rho) = \begin{cases} \frac{1}{r} & \text{si } 0 < \rho < 1 \\ 0 & \text{en otro caso} \end{cases}.$$

Sean $\alpha = a\rho_i + (1 - 2a)\rho_j$, $\beta = a\rho_j + (1 - 2a)\rho_i$ y $\theta = x_i - x_j$, entonces por el lema 3.11 se tiene que

$$P(\mathcal{I}_i \cap \mathcal{I}_j \neq \emptyset) = P(-\beta \leq \theta \leq \alpha).$$

En el siguiente paso, se calcula la probabilidad anterior al condicionar en el valor de ρ_i y ρ_j y fijarlos como $\hat{\rho}_i$ y $\hat{\rho}_j$. Se define también $\hat{\alpha} = a\hat{\rho}_i + (1 - 2a)\hat{\rho}_j$ y $\hat{\beta} = a\hat{\rho}_j + (1 - 2a)\hat{\rho}_i$. Entonces

$$\begin{aligned} P(-\beta \leq \theta \leq \alpha) &= \int_0^r \int_0^r P(-\beta \leq \theta \leq \alpha | \rho_i = \hat{\rho}_i, \rho_j = \hat{\rho}_j) dP(\rho_i \leq \hat{\rho}_i, \rho_j \leq \hat{\rho}_j) \\ &= \int_0^r \int_0^r P(-\hat{\beta} \leq \theta \leq \hat{\alpha}) \frac{1}{r^2} d\hat{\rho}_i d\hat{\rho}_j \end{aligned}$$

Ahora bien, por los lemas 3.9 y 3.10, si $a \in [0, \frac{1}{2}]$ se tiene que $0 \leq \alpha, \beta \leq 1$. En tal caso, la probabilidad $P(-\hat{\beta} \leq \theta \leq \hat{\alpha})$ se calcula de acuerdo a 3.3. Esta probabilidad está representada como la región sombreada de la Figura 3.3.

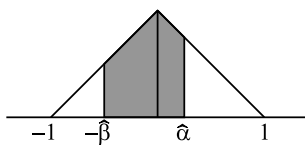


Figura 3.3: La probabilidad $P\left(-\hat{\beta} \leq \theta \leq \hat{\alpha}\right)$ está representada por el área sombreada para $a \in \left[0, \frac{1}{2}\right]$ y $0 \leq \hat{\alpha}, \hat{\beta} \leq 1$.

Se puede verificar entonces que

$$P\left(-\hat{\beta} \leq \theta \leq \hat{\alpha}\right) = \frac{2\hat{\beta} - \hat{\beta}^2 + 2\hat{\alpha} - \hat{\alpha}^2}{2}.$$

Ahora se puede continuar con el cálculo de la probabilidad

$$P(-\beta \leq \theta \leq \alpha) = \int_0^r \int_0^r \frac{2\hat{\beta} - \hat{\beta}^2 + 2\hat{\alpha} - \hat{\alpha}^2}{2} \cdot \frac{1}{r^2} d\hat{\rho}_i d\hat{\rho}_j.$$

Al sustituir $\hat{\alpha} = a\hat{\rho}_i + (1-2a)\hat{\rho}_j$ y $\hat{\beta} = a\hat{\rho}_j + (1-2a)\hat{\rho}_i$ y resolver la integral se obtiene

$$P(-\beta \leq \theta \leq \alpha) = (1-a)r - \frac{r^2}{6} (4a^2 - 5a + 2).$$

Si $a \in \left[\frac{1}{2}, 1\right]$, de acuerdo al lema 3.10 tenemos que $\hat{\alpha} \leq 0$ y $\hat{\beta} \geq 0$ ó $\hat{\beta} \leq 0$ y $\hat{\alpha} \geq 0$. En cada caso se tienen dos posibilidades, lo que da un total de cuatro subcasos, ilustrados en la Figura 3.4. Para los subcasos b) y d) se tiene que $P\left(-\hat{\beta} \leq \theta \leq \hat{\alpha}\right) = 0$.

Para el subcaso a) se tiene que $P\left(-\hat{\beta} \leq \theta \leq \hat{\alpha}\right) < P\left(-\hat{\beta} \leq \theta \leq -\hat{\alpha}\right)$. Para el subcaso c) se tiene que $P\left(-\hat{\beta} \leq \theta \leq \hat{\alpha}\right) < P\left(\hat{\beta} \leq \theta \leq \hat{\alpha}\right)$. Lo anterior significa que la probabilidad $P\left(-\hat{\beta} \leq \theta \leq \hat{\alpha}\right)$ es menor que aquella representada por el área sombreada en la Figura 3.2 para el caso $a \in \left[0, \frac{1}{2}\right]$ y $0 \leq \hat{\alpha}, \hat{\beta} \leq 1$. Por lo tanto

$$P(-\beta \leq \theta \leq \hat{\alpha}) < (1-a)r - \frac{r^2}{6} (4a^2 - 5a + 2).$$

■

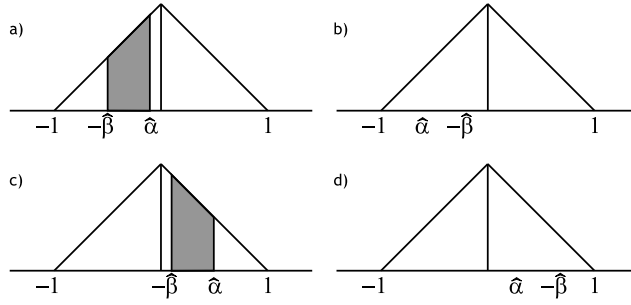


Figura 3.4: Los subcasos a) y b) provienen del caso $\hat{\alpha} \leq 0, \hat{\beta} \geq 0$. Los subcasos c) y d) provienen del caso $\hat{\beta} \leq 0, \hat{\alpha} \geq 0$.

La razón para seleccionar $r < 1$ es porque se tiene interés en el comportamiento de la conexidad cuando $r \rightarrow 0$. Desde la perspectiva de EDL, si las capacidades de los robots son muy grandes entonces la instancia del problema podría resolverse trivialmente. Cualquier robot con una capacidad suficientemente grande podría completar la tarea solo.

La Figura 3.5 representa la superficie de $P(-\beta \leq \theta \leq \alpha)$ en función de a y r cuando $a \in [0, \frac{1}{2}]$. Como era de esperar, la probabilidad máxima se alcanza en $r = 1, a = 0$.

3.7. Umbral de transición de fase para EDLA

Como ya se ha visto, el resultado de Scheinerman proporciona una cota inferior para la transición de fase de EDL. Por otro lado, se ha visto que EDLA es una variante polinomial de EDL que consiste en restringir el vector $a = (0, 0, \dots)$ de una instancia de EDL. En este sentido, si una instancia de EDLA es soluble, eso indica que la instancia equivalente en EDL es soluble. Sin embargo, la afirmación inversa es falsa. Puede haber una instancia EDL-soluble que no sea EDLA-soluble, como se ilustra en la Figura 2.6 b). Así, en esta sección se determina una cota superior para el umbral de la transición de fase de EDL: el umbral de transición de fase para EDLA. Las instancias que se encuentren por encima de este umbral, tienen alta probabilidad de ser EDLA-solubles y por lo tanto EDL-solubles. Las instancias que se encuentren por debajo de este umbral tienen alta probabilidad de ser EDLA-insolubles. Esto no significa que no sean solubles en EDL, como se ha comentado. Por esta razón el umbral de solubilidad para EDLA es una cota superior para el umbral de solubilidad para EDL. A continuación se enuncia una serie de lemas necesarios para llegar al resultado principal sobre el umbral de transición de fase para EDLA, que se encuentra al final de esta sección.

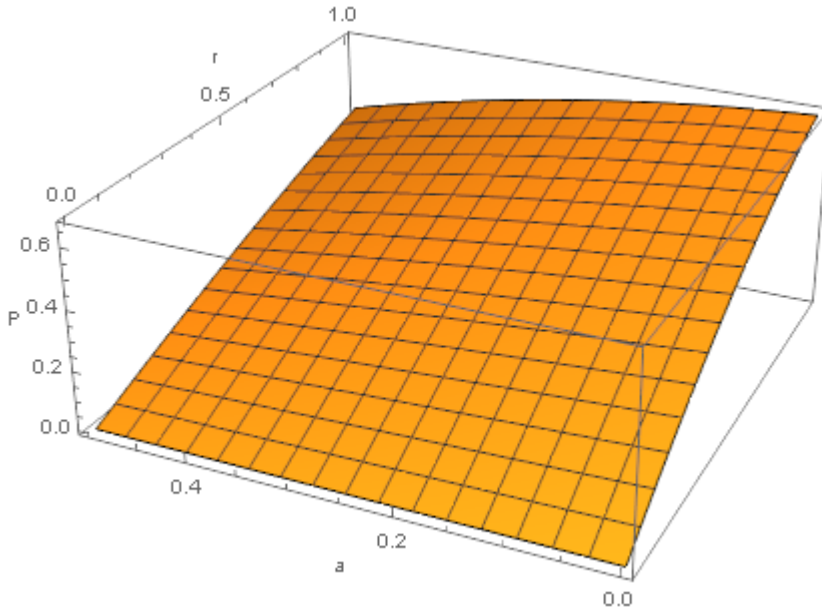


Figura 3.5: $P(-\beta \leq \theta \leq \alpha)$ en función de a y r cuando $a \in [0, \frac{1}{2}]$.

Lema 3.13. Sean $\hat{x}_i \in [1/3, 2/3]$, $r \ll \frac{1}{3}$, $\hat{\rho}_i \leq r$, ρ_j una variable aleatoria que se distribuye de manera uniforme en $[0, r]$ y x_j una variable aleatoria que se distribuye de manera uniforme en $[0, 1]$. Entonces

$$P([x_j, x_j + \rho_j] \cap [\hat{x}_i, \hat{x}_i + \hat{\rho}_i] = \emptyset) = 1 - \frac{r}{2} - \hat{\rho}_i.$$

Demostración. La primera observación es que el evento $[x_j, x_j + \rho_j] \cap [\hat{x}_i, \hat{x}_i + \hat{\rho}_i] = \emptyset$ es equivalente al evento $x_j \notin [\hat{x}_i - \rho_j, \hat{x}_i + \hat{\rho}_i]$ como lo ilustra la Figura 3.6.

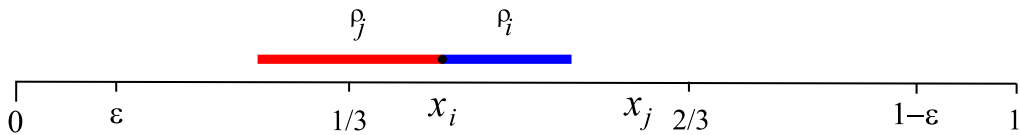


Figura 3.6: $x_j \notin [\hat{x}_i - \rho_j, \hat{x}_i + \hat{\rho}_i]$.

Para $\hat{\rho}_j$ fijo es claro que $P(x_j \notin [\hat{x}_i - \hat{\rho}_j, \hat{x}_i + \hat{\rho}_j]) = 1 - \hat{\rho}_i - \hat{\rho}_j$. Entonces se tiene que

$$\begin{aligned} P(x_j \notin [\hat{x}_i - \rho_j, \hat{x}_i + \hat{\rho}_j]) &= \int_0^r (1 - \hat{\rho}_i - \hat{\rho}_j) \frac{1}{r} d\hat{\rho}_j \\ &= \frac{-1}{2r} (1 - \hat{\rho}_i - \hat{\rho}_j)^2 \Big|_0^r \\ &= \frac{1}{2r} \left[(1 - \hat{\rho}_i)^2 - (1 - \hat{\rho}_i - r)^2 \right] \\ &= 1 - \frac{r}{2} - \hat{\rho}_i \end{aligned}$$

■

Lema 3.14. Sean $\hat{x}_i, \hat{x}_j \in [1/3, 2/3]$, $r \ll \frac{1}{3}$, $\hat{\rho}_i, \hat{\rho}_j \leq r$, ρ_k una variable aleatoria que se distribuye de manera uniforme en $[0, r]$ y x_k una variable aleatoria que se distribuye de manera uniforme en $[0, 1]$. Además $[\hat{x}_j, \hat{x}_j + \hat{\rho}_j] \cap [\hat{x}_i, \hat{x}_i + \hat{\rho}_i] = \emptyset$. Entonces

$$P([x_k, x_k + \rho_k] \cap [\hat{x}_i, \hat{x}_i + \hat{\rho}_i] = \emptyset \text{ y } [x_k, x_k + \rho_k] \cap [\hat{x}_j, \hat{x}_j + \hat{\rho}_j] = \emptyset) = 1 - r - \hat{\rho}_i - \hat{\rho}_j.$$

Demostración. La primera observación es que el evento $[x_k, x_k + \rho_k] \cap [\hat{x}_i, \hat{x}_i + \hat{\rho}_i] = \emptyset$ es equivalente al evento $A = \{x_k \notin [\hat{x}_i - \rho_k, \hat{x}_i + \hat{\rho}_i]\}$. De manera análoga, $[x_k, x_k + \rho_k] \cap [\hat{x}_j, \hat{x}_j + \hat{\rho}_j] = \emptyset$ es equivalente al evento $B = \{x_k \notin [\hat{x}_j - \rho_k, \hat{x}_j + \hat{\rho}_j]\}$ como lo muestra la Figura 3.8.

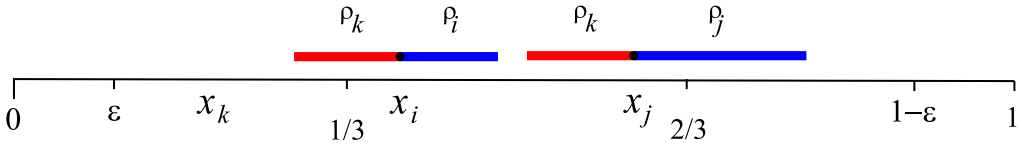


Figura 3.7: $A \cap B$.

Para $\hat{\rho}_k$ fijo es claro que $P(A \cap B) = 1 - 2\hat{\rho}_k - \hat{\rho}_i - \hat{\rho}_j$. Entonces se tiene que

$$\begin{aligned} P(A \cap B) &= \int_0^r (1 - 2\hat{\rho}_k - \hat{\rho}_i - \hat{\rho}_j) \frac{1}{r} d\hat{\rho}_k \\ &= \frac{-1}{2r} (1 - 2\hat{\rho}_k - \hat{\rho}_i - \hat{\rho}_j)^2 \Big|_0^r \\ &= \frac{1}{2r} \left[(1 - \hat{\rho}_i - \hat{\rho}_j)^2 - (1 - 2r - \hat{\rho}_i - \hat{\rho}_j)^2 \right] \\ &= 1 - r - \hat{\rho}_i - \hat{\rho}_j. \end{aligned}$$



A continuación se da una definición útil en el análisis asintótico de esta sección.

Definición 3.19. Sean f y g dos funciones tal que $f, g : \mathbb{Z}^+ \rightarrow \mathbb{R}$ ó $f, g : \mathbb{R} \rightarrow \mathbb{R}$. Se dice que f es *asintóticamente equivalente* a g , y se escribe $f \sim g$, si

$$\lim_{x \rightarrow \infty} \frac{f(x)}{g(x)} = 1 \quad \text{ó} \quad \lim_{x \rightarrow 0} \frac{f(x)}{g(x)} = 1$$

La siguiente proposición de Scheinerman (proposición 17 [57]), que se enuncia como lema aquí, demuestra la equivalencia asintótica $n^{-c} \sim \left(1 - \frac{c \log n}{n}\right)^n$ que será útil en la demostración del teorema 3.17.

Lema 3.15. Sea $c > 0$ entonces

$$n^{-c} \sim \left(1 - \frac{c \log n}{n}\right)^n.$$

Demostración. Se tiene que

$$\begin{aligned} F(n) &= \left(1 - \frac{c \log n}{n}\right)^n \\ \log F(n) &= n \log \left(1 - \frac{c \log n}{n}\right). \end{aligned}$$

Ahora bien, se utiliza el hecho $\log(1+y) = y + O(y^2)$ [57] por lo que se tiene

$$\begin{aligned} \log F(n) &= n \left[-\frac{c \log n}{n} + O\left(\frac{c^2 \log^2 n}{n^2}\right) \right] \\ &= -c \log n + O\left(\frac{c^2 \log^2 n}{n}\right) \\ &= \log n^{-c} + o(1), \end{aligned}$$

entonces $F(n) = n^{-c} \cdot e^{o(1)} \sim n^{-c}$.



Lema 3.16. Sea $\{X_i\}_{i=1}^n$ un conjunto de variables aleatorias idénticamente distribuidas e independientes tal que $X_i \in \{0, 1\}$ para toda i . Sea $X = \sum_{i=1}^n X_i$. Entonces

$$E(X^2) = E(X) + n(n-1)E(X_i X_j).$$

Demostración. Por definición tenemos que

$$\begin{aligned} E(X^2) &= E\left(\left(\sum_{i=1}^n X_i\right)\left(\sum_{i=1}^n X_i\right)\right) \\ &= E\left(\sum_{i=1}^n X_i^2\right) + E\left(\sum_{i \neq j} X_i X_j\right) \\ &= \sum_{i=1}^n E(X_i^2) + \sum_{i \neq j} E(X_i X_j) \end{aligned}$$

y puesto que $X_i^2 = 1$ si y sólo si $X_i = 1$ implica $E(X_i) = E(X_i^2)$, entonces

$$E(X^2) = E(X) + n(n-1)E(X_i X_j).$$

■

Definición 3.20. Sea $I = (Q, \varepsilon, 1 - \varepsilon)$ una instancia de EDL o EDLA con $Q = \{(x_i, \rho_i)\}_{i=1}^n$. Sea i el vértice que representa al i -ésimo intervalo en la gráfica de intervalos con el modelo asimétrico $G_A(Q, a)$. Se denota por $d(i)$ el grado del vértice i , es decir, el número de intersecciones que el i -ésimo intervalo tiene con otros intervalos.

A continuación, se discuten los resultados en torno al umbral de transición de fase para EDLA.

Teorema 3.17. *Supóngase que $r = \frac{c \log n}{n}$. Si $0 < c < 2$ entonces casi todas las gráficas de intervalos asociadas a instancias de EDLA tienen vértices aislados y por lo tanto son desconexas. Si $c > 2$ casi todas las gráficas de intervalos son conexas.*

Demostración. **A. Caso $c < 2$.**

Para la primer parte, se demuestra que si $c < 2$ existen vértices aislados cuyos x_i se encuentran en el intervalo $[\frac{1}{3}, \frac{2}{3}]$ en casi todas las gráficas de intervalos. Se definen las siguientes variables aleatorias

$$X_i = \begin{cases} 1 & \text{si } d(i) = 0 \text{ y } x_i \in [\frac{1}{3}, \frac{2}{3}] \\ 0 & \text{en otro caso} \end{cases}, \quad X = \sum_{i=1}^m X_i$$

La elección del intervalo $[\frac{1}{3}, \frac{2}{3}]$ es arbitraria y para facilitar el cálculo de la probabilidad. Así, la variable aleatoria X cuenta el número de vértices aislados cuyos x_i se encuentran en $[\frac{1}{3}, \frac{2}{3}]$ cuando r es pequeño. Si $x_i \in [\frac{1}{3}, \frac{2}{3}]$, se tiene por el lema 3.13 que la probabilidad de que $n-1$ intervalos aleatorios independientes no se intersecten con el intervalo \mathcal{I}_i es $\frac{1}{r} \int_0^r (1 - \frac{r}{2} - \rho_i)^{n-1} d\rho_i$. Asimismo, $P(x_i \in [\frac{1}{3}, \frac{2}{3}]) = \frac{1}{3}$. Por lo cual

$$\begin{aligned}
E(X_i) &= 1 \cdot P(X_i = 1) + 0 \cdot P(X_i = 0) \\
&= P\left(d(i) = 0 \text{ y } x_i \in \left[\frac{1}{3}, \frac{2}{3}\right]\right) \\
&= P\left(d(i) = 0 | x_i \in \left[\frac{1}{3}, \frac{2}{3}\right]\right) P\left(x_i \in \left[\frac{1}{3}, \frac{2}{3}\right]\right) \\
&= \left[\frac{1}{r} \int_0^r \left(1 - \frac{r}{2} - \rho_i\right)^{n-1} d\rho_i\right] \frac{1}{3} \\
&= \frac{\left(1 - \frac{r}{2}\right)^n - \left(1 - \frac{3r}{2}\right)^n}{3rn}
\end{aligned}$$

Ahora bien, dado que $r = \frac{c \log n}{n}$ se tiene que $\left(1 - \frac{r}{2}\right)^n = \left(1 - \frac{c \log n}{2n}\right)^n$ y por el lema 3.15 se tiene que $\left(1 - \frac{r}{2}\right)^n \sim n^{-c/2}$. De manera similar $\left(1 - \frac{3r}{2}\right)^n \sim n^{-3c/2}$. Entonces, se tiene que $E(X_i) \rightarrow \frac{n^{1-c/2} - n^{1-3c/2}}{3c \log n}$ y dado que $E(X) = \sum X_i$ se obtiene

$$E(X) \rightarrow \frac{n^{1-c/2} - n^{1-3c/2}}{3c \log n}.$$

Si se toma el límite de $E(X)$ cuando $n \rightarrow \infty$ y se aplican las leyes de límites y la regla de L'Hôpital se tiene que

$$\begin{aligned}
\lim_{n \rightarrow \infty} E(X) &= \lim_{n \rightarrow \infty} \frac{n^{1-c/2} - n^{1-3c/2}}{3c \log n} \\
&= \lim_{n \rightarrow \infty} \frac{\frac{n^{1-c/2}}{n^{1-c/2}} - \frac{n^{1-3c/2}}{n^{1-c/2}}}{\frac{3c \log n}{n^{1-c/2}}} \\
&= \lim_{n \rightarrow \infty} \frac{1 - n^{-c}}{\frac{3c \log n}{n^{1-c/2}}} \\
&= \lim_{n \rightarrow \infty} \frac{1 - n^{-c}}{\frac{3c}{(1-c/2)n^{1-c/2}}} = \infty
\end{aligned}$$

puesto que $0 < c < 2$.

Por otro lado, dada la independencia de las variables aleatorias X_i , para cualesquiera i y j con $i \neq j$ se tiene que

$$\begin{aligned}
E(X_i X_j) &= E(X_i X_j | \mathcal{I}_i \cap \mathcal{I}_j = \emptyset) P(\mathcal{I}_i \cap \mathcal{I}_j = \emptyset) + E(X_i X_j | \mathcal{I}_i \cap \mathcal{I}_j \neq \emptyset) P(\mathcal{I}_i \cap \mathcal{I}_j \neq \emptyset) \\
&= P(X_i X_j = 1 | \mathcal{I}_i \cap \mathcal{I}_j = \emptyset) P(\mathcal{I}_i \cap \mathcal{I}_j = \emptyset) + 0 \cdot P(\mathcal{I}_i \cap \mathcal{I}_j \neq \emptyset)
\end{aligned}$$

Si $x_i, x_j \in [\frac{1}{3}, \frac{2}{3}]$, se tiene por el lema 3.14 que la probabilidad de que $n-2$ intervalos aleatorios independientes no se intersecten con el intervalo \mathcal{I}_i ni con el intervalo \mathcal{I}_j es $\frac{1}{r^2} \int_0^r \int_0^r (1-r-\rho_i-\rho_j)^{n-2} d\rho_i d\rho_j$. Asimismo, $P(x_i, x_j \in [\frac{1}{3}, \frac{2}{3}]) = \frac{1}{9}$. Por lo cual

$$P(X_i X_j = 1 | \mathcal{I}_i \cap \mathcal{I}_j = \emptyset) = \frac{1}{9r^2} \int_0^r \int_0^r (1-r-\rho_i-\rho_j)^{n-2} d\rho_i d\rho_j$$

Entonces

$$\begin{aligned} E(X_i X_j) &= \frac{P(\mathcal{I}_i \cap \mathcal{I}_j = \emptyset)}{9r^2} \int_0^r \int_0^r (1-r-\rho_i-\rho_j)^{n-2} d\rho_i d\rho_j \\ &= \frac{(1-P(\mathcal{I}_i \cap \mathcal{I}_j \neq \emptyset))[(1-r)^n + (1-3r)^n - 2(1-2r)^n]}{9r^2 n(n-1)} \end{aligned}$$

Por el teorema 3.12 se tiene que $P(\mathcal{I}_i \cap \mathcal{I}_j \neq \emptyset) \leq (1-a)r - \frac{r^2}{6}(4a^2 - 5a + 2)$ por lo que $P(\mathcal{I}_i \cap \mathcal{I}_j \neq \emptyset) \rightarrow 0$ si $r \rightarrow 0$. Entonces $1 - P(\mathcal{I}_i \cap \mathcal{I}_j \neq \emptyset) \rightarrow 1$ si $n \rightarrow \infty$. Ahora bien, dado que $r = \frac{c \log n}{n}$ se tiene por el lema 3.15 que $(1-r)^n \sim n^{-c}$, $(1-2r)^n \sim n^{-2c}$ y $(1-3r)^n \sim n^{-3c}$. Por lo cual

$$\begin{aligned} E(X_i X_j) &\rightarrow \frac{n^{-c} + n^{-3c} - 2n^{-2c}}{9 \frac{c^2 \log^2 n}{n^2} n(n-1)} \\ &= \frac{n^{2-c} + n^{2-3c} - 2n^{2-2c}}{9c^2 n(n-1) \log^2 n} \end{aligned}$$

De manera que al aplicar el lema 3.16 se obtiene

$$\begin{aligned} E(X^2) &\rightarrow E(X) + n(n-1) \frac{n^{2-c} + n^{2-3c} - 2n^{2-2c}}{9c^2 n(n-1) \log^2 n} \\ &= E(X) + \frac{n^{2-c} + n^{2-3c} - 2n^{2-2c}}{9c^2 \log^2 n}. \end{aligned}$$

Ahora, si se calcula $E(X^2) - E(X)^2$

$$\begin{aligned} E(X^2) - E(X)^2 &\rightarrow E(X) + \frac{n^{2-c} + n^{2-3c} - 2n^{2-2c}}{9c^2 \log^2 n} - \left(\frac{n^{1-c/2} - n^{1-3c/2}}{3c \log n} \right)^2 \\ &= E(X) \end{aligned}$$

Al utilizar la desigualdad de Chebyshev se tiene que

$$\begin{aligned}
 P(|X - E(X)| \geq E(X)) &\leq \frac{\text{Var}(X)}{E(X)^2} = \frac{E(X)}{E(X)^2} = \frac{1}{E(X)} \\
 1 - P(|X - E(X)| < E(X)) &\leq \frac{1}{E(X)} \\
 1 - \frac{1}{E(X)} &\leq P(|X - E(X)| < E(X))
 \end{aligned}$$

Así que, puesto que $E(X) \rightarrow \infty$ se tiene que $P(|X - E(X)| < E(X)) \rightarrow 1$. Es importante señalar que X es una variable aleatoria discreta, de manera que el evento $|X - E(X)| < E(X)$ implica necesariamente que $X \geq 1$, es decir, $P(X \geq 1) \rightarrow 1$.

B. Caso $c > 2$.

A continuación, la demostración de la segunda parte. Se definen las siguientes variables aleatorias:

$$A_i = \begin{cases} 1 & \text{si } A^-(x_i + \rho_i, I) = \emptyset \text{ y } (x_i, \rho_i) \notin B \\ 0 & \text{en otro caso} \end{cases}, \quad A = \sum_{i=1}^n A_i$$

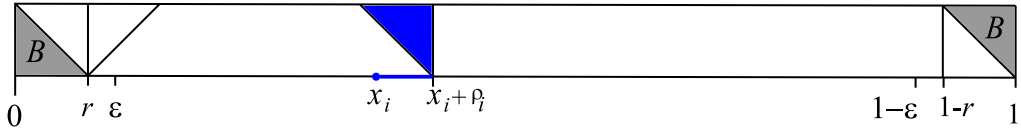


Figura 3.8: La probabilidad de que ningún robot se encuentre en $A^-(x_i + \rho_i)$ es $P(A^-(x_i + \rho_i, I) = \emptyset) = (1 - \frac{r}{2})^{n-1}$.

De modo que A cuenta el número de regiones activas izquierdas vacías. Ahora, dado un robot $(x_i, \rho_i) \notin B$, se observa que la probabilidad de que otro robot esté en $A^-(x_i + \rho_i)$ es $\frac{r^2/2}{r} = \frac{r}{2}$, como se ilustra en la Figura 3.8. La probabilidad de que no esté en $A^-(x_i + \rho_i)$ es $1 - \frac{r}{2}$. La probabilidad de que un robot esté en B es de $\frac{r^2}{r} = r$. Entonces, la probabilidad de que ninguno de los $n - 1$ robots restantes estén en $A^-(x_i + \rho_i)$ es $(1 - \frac{r}{2})^{n-1}$. A partir de esto se obtiene

$$\begin{aligned}
E(A_i) &= 1 \cdot P(A_i = 1) + 0 \cdot P(A_i = 0) \\
&= P(A^-(x_i + \rho_i, I) = \emptyset \text{ y } (x_i, \rho_i) \notin B) \\
&= P(A^-(x_i + \rho_i, I) = \emptyset | (x_i, \rho_i) \notin B) P((x_i, \rho_i) \notin B) \\
&= \left(1 - \frac{r}{2}\right)^{n-1} (1-r)
\end{aligned}$$

Pero $r = \frac{c \log n}{n}$ implica que $E(A_i) = \frac{(1-r)(1-\frac{c \log n}{2n})^n}{(1-\frac{c \log n}{2n})}$, entonces por el lema 3.15 $E(A_i) \rightarrow n^{-c/2}$. Por lo cual

$$E(A) = \sum_{i=1}^n E(A_i) \rightarrow n^{1-c/2}$$

Ahora bien, por hipótesis $c > 2$, entonces $E(A) \rightarrow 0$.

Por otro lado, por el lema 3.16, $E(A^2) = E(A) + n(n-1)E(A_i A_j)$. La probabilidad de que los robots (x_i, ρ_i) y (x_j, ρ_j) no estén en B es $(1-r)^2$. En ese caso, la probabilidad de que los otros $n-2$ robots no estén en $A^-(x_i + \rho_i)$ ni en $A^-(x_j + \rho_j)$ es $(1-r)^{n-2}$, como lo ilustra la Figura 3.9, por lo cual se tiene que

$$E(A_i A_j) = 1 \cdot P(A_i = 1 \text{ y } A_j = 1) + 0 \cdot P(A_i A_j = 0) = (1-r)^2 (1-r)^{n-2} = (1-r)^n,$$

entonces dado que $r = \frac{c \log n}{n}$ y por el lema 3.15 se tiene que $E(A_i A_j) \rightarrow n^{-c}$, por lo cual

$$E(A^2) \rightarrow n^{1-c/2} + n(n-1)n^{-c} = n^{1-c/2} + n^{2-c} - n^{1-c}$$

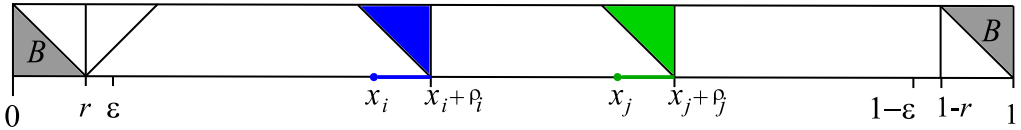


Figura 3.9: $P(A_i = 1 \text{ y } A_j = 1)$.

Por otro lado $E(A)^2 \rightarrow n^{2-c}$, así pues $Var(A) = E(A^2) - E(A)^2 \rightarrow n^{1-c/2} - n^{1-c}$, por lo que $Var(A) \rightarrow 0$ ya que $c > 2$. Si se utiliza la desigualdad de Chebyshev se obtiene

$$P(|A - E(A)| \geq 1) \leq \text{Var}(A)$$

$$P(|A - E(A)| < 1) \geq 1 - \text{Var}(A).$$

Dado que A es una variable aleatoria discreta y que $E(A) \rightarrow 0$, el evento $|A - E(A)| < 1$ significa que $P(A < 1) \rightarrow 1$. Con esto concluye la demostración sobre el umbral de conexidad para EDLA. ■

El teorema 3.17 establece el umbral de conexidad para EDLA. El teorema 3.20 al final de esta sección implica que este umbral es también para la solubilidad de EDLA. Primero se verán dos lemas auxiliares.

Lema 3.18. Sean $\{x_i\}_{i=1}^n$ variables aleatorias independientes con distribución uniforme en el intervalo $[0, 1]$, $L = \min \{x_i\}_{i=1}^n$ y $R = \max \{x_i\}_{i=1}^n$. Entonces

$$F_L(x) = P(L \leq x) = 1 - (1 - x)^n \quad \text{y} \quad F_R(x) = P(R \leq x) = x^n.$$

Demostración. Para $F_R(x)$ la demostración es sencilla. Se debe calcular la probabilidad de que las n variables aleatorias x_i estén en $(0, x)$. Así que por independencia de las variables aleatorias se tiene que

$$F_R(x) = P(R \leq x) = \prod_{i=1}^n P(x_i \leq x)$$

$$= x^n.$$

Para $F_L(x)$ se debe calcular la probabilidad de que al menos un x_i se encuentre en $(0, x)$. Para calcular esta probabilidad se utiliza el complemento, es decir, la probabilidad de que las n variables aleatorias x_i estén en $(x, 1)$

$$F_L(x) = P(L \leq x) = 1 - P(L > x)$$

$$= 1 - \prod_{i=1}^n P(x_i > x)$$

$$= 1 - (P(x_i > x))^n$$

$$= 1 - (1 - P(x_i \leq x))^n$$

$$= 1 - (1 - x)^n.$$
■

Lema 3.19. Sean $\{x_i\}_{i=1}^n$ variables aleatorias independientes con distribución uniforme en el intervalo $[0, 1]$, $L = \min\{x_i\}_{i=1}^n$ y $R = \max\{x_i\}_{i=1}^n$. Entonces $S = R - L$ tiene una distribución beta con parámetros $(n-2, 1)$.

Demostración. Por el lema 3.18 y la definición 3.12 se tiene que $f_L(x) = \frac{d}{dx}F_L(x) = n(1-x)^{n-1}$. A continuación se calcula $P(S \leq d)$ para obtener $f_S(d)$. Sea $d \in \mathbb{R}$. Si se condiciona en $L = x$, se deben considerar dos casos mutuamente excluyentes, como lo ilustra la Figura 3.10. El primero es $d < 1 - x$. La probabilidad condicional de que los restantes $n-1$ puntos estén en el intervalo $[x, x+d]$ dado que $L = x$ es

$$\left(\frac{d}{1-x}\right)^{n-1}.$$

El segundo caso es $d \geq 1 - x$. En este caso, todos los n puntos están en el intervalo $[x, 1]$ y por lo tanto $P(S \leq d | L \geq 1 - d) = 1$. Entonces, por la ley de la probabilidad total

$$\begin{aligned} P(S \leq d) &= P(S \leq d | L < 1 - d)P(L < 1 - d) + P(S \leq d | L \geq 1 - d)P(L \geq 1 - d) \\ &= \int_0^{1-d} \left(\frac{d}{1-x}\right)^{n-1} f_L(x) dx + 1 \cdot P(L \geq 1 - d) \\ &= \int_0^{1-d} \left(\frac{d}{1-x}\right)^{n-1} n(1-x)^{n-1} dx + (1 - (1-d))^n \\ &= n(1-d)d^{n-1} + d^n. \end{aligned}$$

Al derivar la expresión anterior se obtiene $f_S(x) = n(n-1)x^{n-2}(1-x)$. Ahora bien, de acuerdo a la definición 3.17 se tiene que $\beta(n-1, 2) = \frac{1}{n(n-1)}$, por lo cual $f_S(x) = \frac{1}{\beta(n-1, 2)}x^{n-2}(1-x)$ es una función de densidad beta con parámetros $(n-2, 1)$. ■

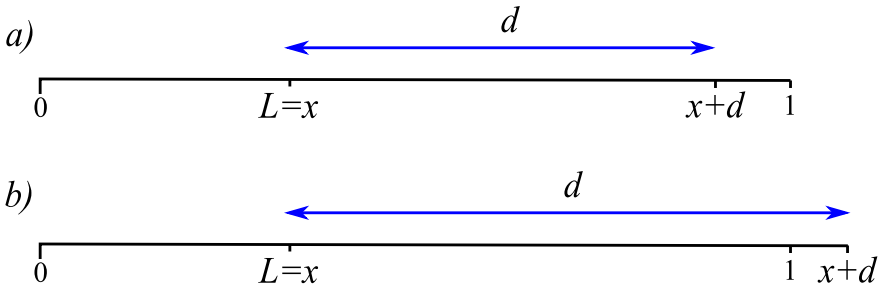


Figura 3.10: Los dos casos de la demostración del lema 3.19.

Teorema 3.20. Sean $\{x_i\}_{i=1}^n$ variables aleatorias independientes con distribución uniforme en el intervalo $[0, 1]$, $L = \min \{x_i\}_{i=1}^n$, $R = \max \{x_i\}_{i=1}^n$, $L' = L$ y $R' = \max \{x_i + \rho_i\}$. Entonces

$$P(R' - L' > 1 - \varepsilon) \rightarrow 1$$

cuando $n \rightarrow \infty$.

Demostración. Claramente $R \leq R'$ así que $R - L \leq R' - L'$, por lo cual $P(R - L \geq d) \leq P(R' - L' \geq d)$. Por último, sea $\varepsilon > 0$, entonces

$$\begin{aligned} P(R - L \geq 1 - \varepsilon) &= 1 - P(R - L < 1 - \varepsilon) \\ &= 1 - \left[n\varepsilon(1 - \varepsilon)^{n-1} + (1 - \varepsilon)^n \right]. \end{aligned}$$

Se ha utilizado el hecho de que $R - L$ tiene una distribución beta con parámetros $(n - 2, 1)$, de acuerdo con el lema 3.19. Ahora, al utilizar la regla de L'Hôpital y las reglas de productos infinitos, si se toma el límite cuando $n \rightarrow \infty$ se tiene que

$$\begin{aligned} \lim_{n \rightarrow \infty} n\varepsilon(1 - \varepsilon)^{n-1} + (1 - \varepsilon)^n &= \lim_{n \rightarrow \infty} n\varepsilon(1 - \varepsilon)^{n-1} + \lim_{n \rightarrow \infty} (1 - \varepsilon)^n \\ &= 0 + 0 \end{aligned}$$

y por lo tanto $\lim_{n \rightarrow \infty} P(R - L \geq 1 - \varepsilon) = 1$ lo cual implica que

$$\lim_{n \rightarrow \infty} P(R' - L' \geq 1 - \varepsilon) = 1.$$

■

La conjunción del teorema 3.12 y el teorema 3.20 significa que el umbral de transición de fase para EDLA no solo se refiere a la conexidad sino a la solubilidad también. Cuando se está por encima del umbral, la longitud de la componente conexa será eventualmente mayor a $1 - \varepsilon$. Así pues, esta unión de intervalos cubrirá a la salida $s = \varepsilon$ y a la terminal $t = 1 - \varepsilon$. Entonces será posible trasladar los datos desde ε hasta $1 - \varepsilon$. Por otro lado, cuando se está por debajo del umbral, la desconexidad de las gráficas de intervalos asociadas a instancias de EDLA equivale a que tales instancias no son solubles, de acuerdo al teorema 2.6 y al corolario 2.8.

3.8. Simulaciones

En esta sección se presentan dos grupos de simulaciones de los modelos probabilísticos presentados en este capítulo. Las instancias que se generaron son en realidad pseudoaleatorias, debido a que resulta imposible generar números verdaderamente aleatorios en una computadora. El primer grupo de simulaciones que se presentan se diseñó para exhibir el resultado del teorema 3.12. El experimento se diseñó de la siguiente manera:

1. Se fijan los valores $s = 0$, $t = 1$, r_{\min} , n_{\min} , r_{\max} y n_{\max} .
2. Se hace una partición uniforme P del intervalo $[r_{\min}, r_{\max}]$ en $n_{\max} - n_{\min}$ elementos.
3. Para cada pareja $[r, n]$, con $r \in P$ y $n = n_{\min}, n_{\min} + 1, \dots, n_{\max}$, se generan m instancias pseudoaleatorias con el modelo simétrico y se verifica su conexidad. La probabilidad empírica de conexidad se estima como el cociente de instancias conexas entre el número de instancias de prueba m . Se guarda este valor en una matriz de tamaño $r_{\max} - r_{\min} \times n_{\max} - n_{\min}$.
4. Se grafican en el plano los valores de la probabilidad empírica guardados en la matriz. Se asigna el color negro al 0 y el color blanco al 1.

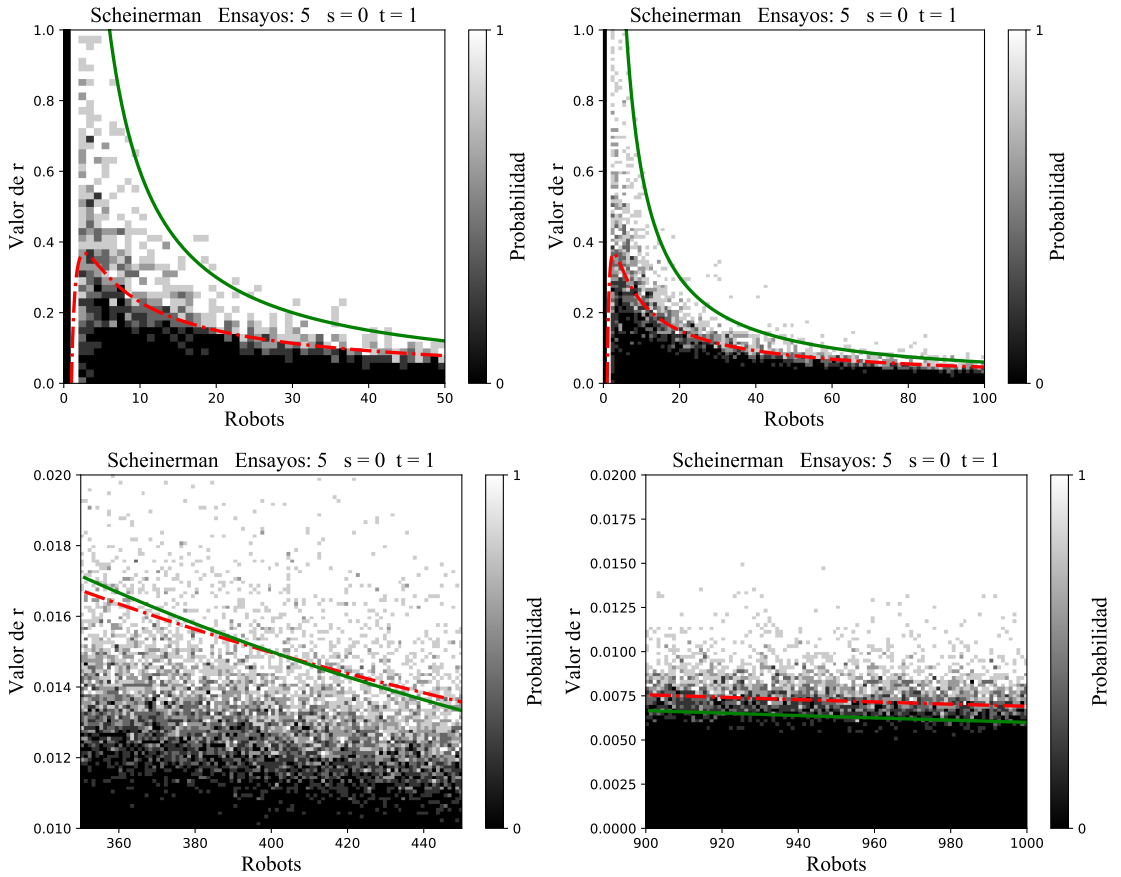


Figura 3.11: La curva roja (guión-punto) corresponde al umbral de Scheinerman $r = \frac{\log n}{n}$. La curva en verde (continua) corresponde a $r = \frac{6(r-s)}{n}$. Se aprecia cómo la segunda es incapaz de modelar el umbral de transición de fase con respecto a conexidad en las gráficas de intervalos aleatorios del modelo simétrico. Cuánto menos podrá modelar el umbral de transición de fase para el modelo asimétrico.

La libreta de Python Scheinerman **ipynb** para acceder al código y realizar las simulaciones: <https://colab.research.google.com/drive/10oNKfExd6yWUKTbu013hLMd0TgZDqBKusp=sharing>

Por último, la Figura 3.12 ilustra mediante simulaciones el Teorema 3.17, de manera análoga a la Figura 3.6 para ilustrar el Teorema 3.12 de la sección 3.6.

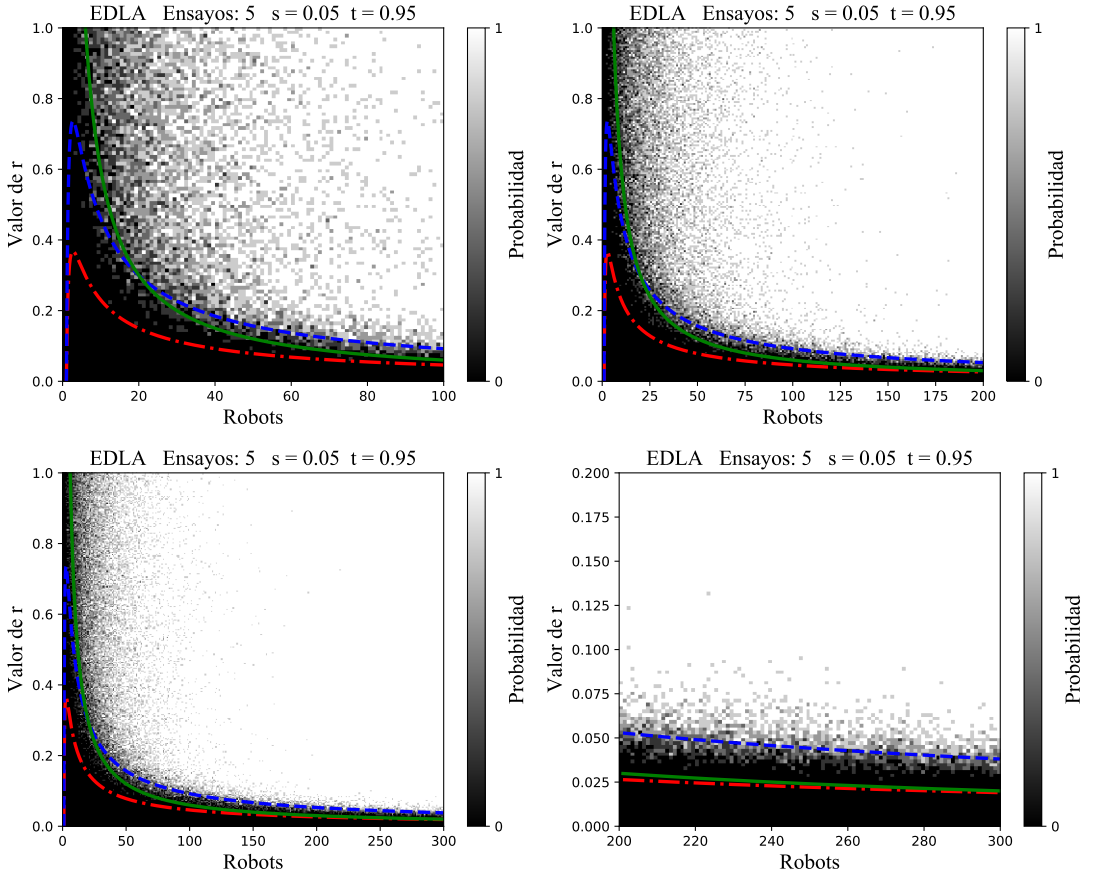


Figura 3.12: La curva en rojo (guión-punto) corresponde al umbral de Scheinerman $r = \frac{\log n}{n}$. La curva en verde (continua) corresponde a $r = \frac{6(t-s)}{n}$. La curva en azul (guión) corresponde al umbral de EDLA $r = \frac{2 \log n}{n}$. Se generaron instancias aleatorias para EDLA en el espacio de parámetros (n, r) y se resolvieron con el Algoritmo 2.2. En este caso, la probabilidad empírica se refiere a la probabilidad de que una instancia aleatoria sea EDLA-soluble.

La libreta de Python **EDLA_DensityPlot.ipynb** para acceder al código y realizar las simulaciones: https://colab.research.google.com/drive/17c6QgocmOW0AJb_g3IYm-B-pW0bW4Vii?usp=sharing

Capítulo 4

Búsqueda con retroceso para resolver EDL

4.1. Método exacto

La *búsqueda con retroceso* (conocido como "backtracking" en inglés) es un algoritmo de uso general para encontrar todas las soluciones, o una parte de ellas, de algunos problemas combinatorios. Las soluciones candidatas se construyen de manera incremental. Tan pronto como se determina que alguna no se puede completar para ser una solución válida, se abandona. En esto consiste el retroceso. Este método fue propuesto por R. J. Walker en la década de 1950. Básicamente consiste en construir una gráfica de árbol. Los nodos adyacentes a la raíz representan las opciones del primer elemento para construir el certificado, el cual será una secuencia de elementos. En el siguiente nivel se encuentran los nodos de los nodos del nivel anterior, es decir, las opciones para elegir el siguiente elemento en la sucesión dependiendo de la elección inmediata anterior. De esta manera se va construyendo la solución de manera incremental. Cuando se llega a un nodo donde se verifica que no se puede continuar construyendo una solución válida, se regresa un nivel anterior y se explora un nodo distinto. Para profundizar en el tema véase [46].

Para resolver las instancias del problema EDL de manera exacta se propone un algoritmo de búsqueda con retroceso (BR). Este algoritmo explora el árbol de todas las posibles subsucesiones de robots. Se inicia en el valor $d = s$ y se consideran los robots en $A(d, I)$ que no están marcados. Se selecciona cada uno por turnos, de manera arbitraria, y se marca como utilizado. Se actualiza d , se verifica si $d \geq t$ (en cuyo caso la búsqueda termina) y se prosigue del mismo modo. Cuando se llega a una d para la cual $A(d, I)$ no cuenta con robots disponibles, si $d < t$ entonces se regresa un nivel en el árbol de búsqueda. En el Algoritmo 4.1 se describe el pseudocódigo.

A continuación, un ejemplo de cómo funciona este algoritmo. Sea

$$I = \{a : (1, 3), b : (1, 2), c : (4, 2), d : (5, 2), e : (8, 3), f : (9, 4), g : (8, 2), s = 0, t = 11\}$$

Algoritmo 4.1 Algoritmo BR para resolver EDL. En la libreta de Python: `BR/recursiveDDL`.

```

1: BR( $I$ , solución,  $t$ , robots_disponibles):
2:    $d = \text{posición actual de los datos en } I$ ;
3:   si  $d \geq t$ : devolver solución;
4:   en otro caso:
5:     calcular  $A(d, I)$ ;
6:     para robot en  $A(d) \cap \text{robots\_disponibles}$ :
7:       copia_robots_disponibles = copia (robots_disponibles);
8:       mover datos con robot, se actualiza posición en  $I$ ;
9:       insertar robot a solución;
10:      eliminar robot de copia_robots_disponibles;
11:      solución = BR( $I$ , solución,  $t$ , copia_robots_disponibles);
12:      si  $d \geq t$ : devolver solución;
13:      en otro caso: deshacer movimiento de robot y remover robot de solución;
14:   devolver solución;

```

una instancia de EDL. En la Figura 4.1 se muestra cómo se construye el árbol de BR. Este ejemplo se encuentra en la libreta de Python correspondiente.

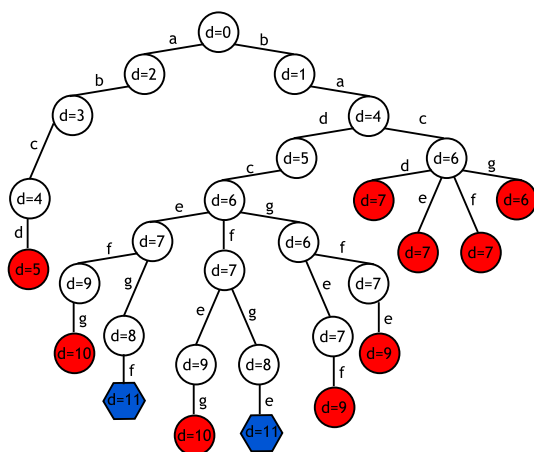


Figura 4.1: En este ejemplo, las subsecuencias de robots se construyen al explorar el árbol desde la raíz hasta alguna de las hojas. Las hojas circulares (rojas) representan subsecuencias que no llegan a la terminal $t = 11$. Las hojas hexagonales (azules) representan subsecuencias que sí llegan a la meta, en este caso, $\{b, a, d, c, e, g, f\}$ y $\{b, a, d, c, f, g, e\}$.

Algoritmo 4.2 Algoritmo de barrido para detectar componentes conexas en una gráfica de intervalos. En la libreta de Python: Scheinerman/**IntervalGraph.graphInit()**.

```

1: ConexidadEDL( $G$  (intervalos, aristas)):
2:   Inicializar: eventos = [], contador = 0, intersecciones = [];
3:   para ( $u_i, v_i$ ) en intervalos: //  $u$  es el inicio,  $v$  es el final
4:     añadir  $u_i, v_i$  a eventos;
5:   ordenar lexicográficamente eventos; // primero alfabéticamente, después por posición.
6:   para evento en eventos:
7:     si evento es inicio:
8:       para  $i$  en intersecciones: // si intersecciones es vacío no entra al bucle
9:         insertar ( $i$ , índice(evento)) en aristas;
10:        insertar (índice(evento),  $i$ ) en aristas;
11:      insertar índice(evento) en intersecciones;
12:      marcar componente(índice(evento)) = contador;
13:     si evento es final:
14:       quitar índice(evento) de intersecciones;
15:       si intersecciones es vacío:
16:         contador = contador + 1;
17:   terminar;

```

La libreta de Python **BR.ipynb** para acceder al código: <https://colab.research.google.com/drive/14Tb7igg1I5stijr2V1hvIaCz5rkC30wS?usp=sharing>

4.2. BR mejorado

En esta sección se examina una versión mejorada del algoritmo de búsqueda con retroceso (BRM). Se utiliza el teorema 2.6 para este fin. La idea es que dada una instancia I de EDL, se construye su gráfica de intervalos asociada con el modelo asimétrico $G_A(Q', a)$. En principio, no se conoce el vector a que garantiza la conexidad de $G_A(Q', a)$, pero debe de ser claro que para $a_i = 0$ el intervalo $[x_i - a_i p_i, x_i + (1 - 2a_i) p_i]$ se maximiza. En la sección 3.6 se demostró cómo la probabilidad de que dos intervalos aleatorios se intersecten se maximiza cuando $a = 0$. Así que el primer candidato es el vector $a = (0, 0, \dots, 0)$. La conexidad de $G_A(Q', a)$ se verifica al utilizar el Algoritmo 4.2, véase ejemplo de la Figura 4.2.

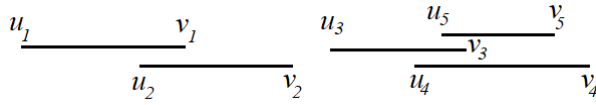


Figura 4.2: En este ejemplo, el algoritmo de barrido procesa los eventos en el orden: $u_1, u_2, v_1, v_2, u_3, u_4, u_5, v_3, v_5, v_4$, y al terminar, se encuentran dos componentes conexas.

En caso de no ser conexa, se observa la componente conexa que contiene a $(s, 0)$. Todos los intervalos en esta componente conexa tienen asociado el valor $a_i = 0$. La atención se fija en el extremo derecho v_j del último intervalo de esta componente conexa. Esta es la última posición hasta donde los datos serían llevados por los robots que se encuentran en esta componente conexa. Ahora bien, debe resultar claro que $A(v_j, I)^- = \emptyset$, por lo que en este punto entra el algoritmo BR descrito en la sección anterior. Se elige un robot k de manera aleatoria en $A(v_j, I)^+$ y se mueven los datos hasta una nueva posición d_k . En esta nueva posición, se examina si existe una componente conexa de intervalos en $G_A(Q', a)$ (excepto por el robot k) y se procede como se ha descrito anteriormente. La justificación para elegir y fijar los valores $a_i = 0$ para los intervalos en las componentes conexas se sigue del lema 2.2, es decir, se logra un alcance máximo al utilizar los robots de esta manera. Este procedimiento está descrito en el Algoritmo 4.3.

Esta variante del algoritmo BR resulta más eficiente porque todos los robots en una componente conexa se "contraen a un solo nodo" en la gráfica de intervalos, al fijar sus valores $a_i = 0$. Cuando el algoritmo BRM explora el espacio de las posibles subsucesiones de robots, está explorando también el espacio de los vectores a . Si se fija en principio ciertos valores a_i , la búsqueda en el árbol generado por BRM se reduce.

La libreta de Python **BRM.ipynb** para acceder al código:

<https://colab.research.google.com/drive/1TN-7G6IMInmZwYVVApIGNC6MUNXn0gvo?usp=sharing>

4.3. Simulaciones

En esta sección se ilustra experimentalmente lo descrito en las secciones 4.1 y 4.2. Se diseñó un experimento de la siguiente manera:

1. Se fijan los valores $s = 0, t = 1, r_{\min}, n_{\min}, r_{\max}$ y n_{\max} .
2. Se hace una partición uniforme P del intervalo $[r_{\min}, r_{\max}]$ en $n_{\max} - n_{\min}$ elementos.

Algoritmo 4.3 Algoritmo BRM. En la libreta de Python: BRM/**recursiveDDL**P.

```

1: Componente_conexa_a=0 ( $I$ ):
2:    $ConexidadEDL(G_A(Q', a = (0, 0, \dots, 0)))$ ;
3:   devolver robots en componente_conexa ( $s$ );
4:
5: BRM( $I$ , solución,  $t$ , robots_disponibles):
6:    $d = \text{posición actual de los datos}$ ;
7:   si  $d \geq t$ : devolver solución;
8:   en otro caso:
9:      $I = \{\text{robots\_disponibles}, d, t\}$ ;
10:     $construir G_A(Q', a = (0, 0, \dots, 0))$ ;
11:    robots_fijos = Componente_conexa_a=0 ( $I$ );
12:     $d_{max} = EDLA(\{\text{robots\_fijos}, d, t\})$ ; // Algoritmo 2.2
13:    robots_disponibles = robots_disponibles - robots_fijos;
14:     $I = \{\text{robots\_disponibles}, d_{max}, t\}$ ;
15:    para robot en  $A(d, I)^+ \cap \text{robots\_disponibles}$ :
16:      copia_robots_disponibles = copia(robots_disponibles);
17:      mover datos con robot, se actualiza posición en  $I$ ;
18:      insertar robot a solución;
19:      eliminar robot de copia_robots_disponibles;
20:      solución = BRM( $I$ , solución,  $t$ , copia_robots_disponibles);
21:      si  $d \geq t$ : devolver solución;
22:      en otro caso: deshacer movimiento de robot y remover robot de solución;
23:      si  $d \geq t$ : devolver solución;
24:      en otro caso: deshacer cambios en las líneas 9-14;
25:      devolver solución;

```

3. Para cada pareja $[r, n]$, con $r \in P$ y $n = n_{\min}, n_{\min} + 1, \dots, n_{\max}$, se generan m instancias aleatorias y se utiliza el algoritmo BRM para resolverlas. La probabilidad empírica de solubilidad se estima como el cociente de instancias resueltas afirmativamente entre el número de instancias de prueba m . Se guarda este valor en una matriz de tamaño $r_{\max} - r_{\min} \times n_{\max} - n_{\min}$.
4. Se grafican en el plano los valores de la probabilidad empírica guardados en la matriz. Se asigna el color negro al 0 y el color blanco al 1.

En la Figura 4.3 se ilustran dos simulaciones para distintos valores de r_{\min} , n_{\min} , r_{\max} y n_{\max} y m . En estos experimentos se aprecia la transición de fase. La zona negra representa la porción del espacio de parámetros (r, n) con una gran probabilidad de que se generen instancias aleatorias no solubles. Por otro lado, la zona blanca representa la porción del espacio de parámetros (r, n) con una gran probabilidad de que se generen instancias

aleatorias solubles.

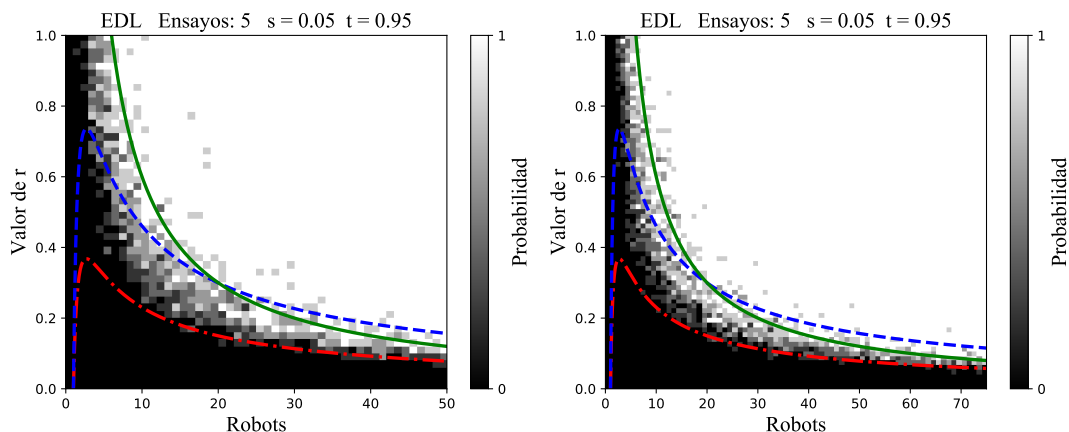


Figura 4.3: La curva en rojo (guión-punto) corresponde al umbral de Scheinerman $r = \frac{\log n}{n}$. La curva en verde corresponde a $r = \frac{6(t-s)}{n}$. La curva en azul (guión) corresponde al umbral de EDLA $r = \frac{2 \log n}{n}$. La naturaleza del algoritmo BR, que puede dar un tamaño de búsqueda exponencial, limitó estos experimentos para valores muy moderados de n .

La libreta de Python **EDL_DensityPlot.ipynb** para acceder al código y realizar las simulaciones: <https://colab.research.google.com/drive/12IsW5G0NcCRSNTt3E2dJAThD5cL?usp=sharing>

4.4. Indicador de dificultad para instancias de EDL

Durante las simulaciones con el algoritmo BR se observó un fenómeno interesante. En el umbral de transición de fase del espacio de parámetros de EDL la cantidad de nodos explorados podía crecer significativamente. Por encima y por debajo del umbral de transición de fase se observó que el número de nodos explorados se mantenía relativamente constante para cada pareja de valores (n, r) . En la literatura relativa a la búsqueda con retroceso se sugiere contabilizar el número de nodos explorados como un indicador de la complejidad asociada para resolver un problema dado mediante esta técnica [70]. Por supuesto que este indicador ignora ciertos detalles computacionales acerca de las operaciones necesarias para construir la estructura del árbol, así como el paradigma del

peor de los casos. No obstante, en este trabajo se adoptó esta idea para hacer una distinción entre instancias fáciles y difíciles. Una *instancia fácil* es aquella que requiere que se exploren pocos nodos para resolverla. Una *instancia difícil* es aquella que requiere que se explore una gran cantidad de nodos para resolverla.

Para explicar el fenómeno observado se propone lo siguiente. Cuando se genera una instancia aleatoria EDL por encima del umbral de EDLA, casi cualquier elección que se haga en los nodos del árbol dará como resultado un certificado afirmativo. Dicho de otra manera, el árbol generado por el algoritmo BR tiene una gran cantidad de *hojas afirmativas* (azules). Un muestreo aleatorio para elegir una trayectoria desde la raíz hasta alguna de las hojas del árbol dará como resultado una respuesta afirmativa con alta probabilidad. Por debajo del umbral de Scheinerman, el árbol de BR tendrá prácticamente en su totalidad *hojas negativas* (rojas). Es alrededor de la zona de transición de fase de EDL, que necesariamente estaría entre los dos umbrales anteriores, donde se conjetura que comienzan a aparecer las primeras hojas afirmativas. Encontrarlas requiere explorar una gran porción del árbol de BR y la probabilidad de muestrear aleatoriamente una trayectoria que termine en una hoja afirmativa es baja. En la sección 6.4 se amplía más sobre este punto al utilizar un modelo probabilístico (véase apéndice A.7) para explicar el muestreo aleatorio en el árbol generado por el algoritmo BR.

Para ilustrar lo anterior, se realizaron también simulaciones que se ilustran en la Figura 4.4. Esta vez el interés radica en el número de nodos explorados. Para obtener una imagen aproximada de dónde se encuentran las instancias *difíciles*, se realizó el siguiente experimento

1. Se fijan los valores $s = 0$, $t = 1$, r_{\min} , n_{\min} , r_{\max} y n_{\max} .
2. Se hace una partición uniforme P del intervalo $[r_{\min}, r_{\max}]$ en $n_{\max} - n_{\min}$ elementos.
3. Para cada pareja $[r, n]$, con $r \in P$ y $n = n_{\min}, n_{\min} + 1, \dots, n_{\max}$, se generan m instancias aleatorias y se utiliza el algoritmo BRM para resolverlas. Cuando el número de nodos explorados rebasa cierto valor μ_c , se interrumpe BRM y dicha instancia se clasifica como difícil (1). Cuando BRM resuelve la instancia y el número de nodos se mantuvo por debajo de μ_c dicha instancia se clasifica como fácil (0). La probabilidad empírica de dificultad se estima como el cociente de instancias clasificadas como difíciles entre el número de instancias de prueba m . Se guarda este valor en una matriz de tamaño $r_{\max} - r_{\min} \times n_{\max} - n_{\min}$.
4. Se grafican en el plano los valores de la probabilidad empírica guardados en la matriz. Se asigna el color blanco al 0 y el color negro al 1. Todos los valores intermedios se les asigna el color correspondiente del gradiente que va de blanco a negro. En la Figura 4.4 se presentan algunas simulaciones.

Pero, ¿cómo se determinó el valor μ_c ? En primer lugar, se conjeturó que la variable aleatoria X que representa el número de nodos explorados por BR se podría modelar con una distribución exponencial. La elección no es del todo arbitraria. En primer lugar, se realizaron una serie de experimentos para muestrear la media de X y se observó que la media y la desviación estándar eran parecidas para distintas configuraciones de valores. En la distribución exponencial con parámetro λ , la media $\frac{1}{\lambda}$ y la desviación estándar coinciden. En segundo lugar, la función exponencial se utiliza para modelar los tiempos de espera en una fila (p.ej. banco o supermercado). Estos tiempos suelen ser cercanos a la media con alta probabilidad, pero también es posible valores muy altos con baja probabilidad. Aunque estos argumentos son razonables, no representan ninguna clase de prueba. Queda pendiente investigar la distribución de la variable aleatoria que representa el número de nodos explorados.

En la sección 3.5 se determinó que el número esperado de robots en una instancia aleatoria resuelta afirmativamente es $\mu = \frac{3(t-s)}{r}$. Ahora bien, resulta ser que en muchos casos el número de nodos coincide con el número de robots en el certificado (cuando una instancia aleatoria se resuelve al muestrear una única trayectoria en el árbol de BR). De modo que, por las observaciones experimentales y esta última observación, empíricamente se obtuvo que $\mu \approx \frac{1}{\lambda}$. Entonces, ¿cuál es el valor x que acumula el 0.99 de probabilidad de que una observación de X sea menor que dicho valor? A continuación el cálculo correspondiente.

$$P(X \leq x) = 1 - e^{-\lambda x}$$

$$0.99 = 1 - e^{-\lambda x}$$

$$\frac{1}{100} = \frac{1}{e^{\lambda x}}$$

$$100 = e^{\lambda x}$$

$$\log 100 = \lambda x$$

$$\frac{1}{\lambda} \log 100 = x$$

$$\mu \log 100 = x$$

Entonces, al aplicar BRM, si el número de nodos rebasa el valor $\mu_c = \mu \log 100$, se interrumpe la búsqueda y se cataloga esa instancia como “difícil”. Presumiblemente con un número muy alto de nodos. Esto se hace en el espacio de parámetros (n, r) . Entre más negro se colorea una coordenada, significa que la mayoría de los muestreos en esa coordenada generaron instancias “presumiblemente difíciles”.

La libreta de Python **Nodes_EDL.ipynb** para acceder al código y realizar las simulaciones: https://colab.research.google.com/drive/12tnJ00D1G8R_zrNYV0Kg1dqX1Ta080Iusp=sharing

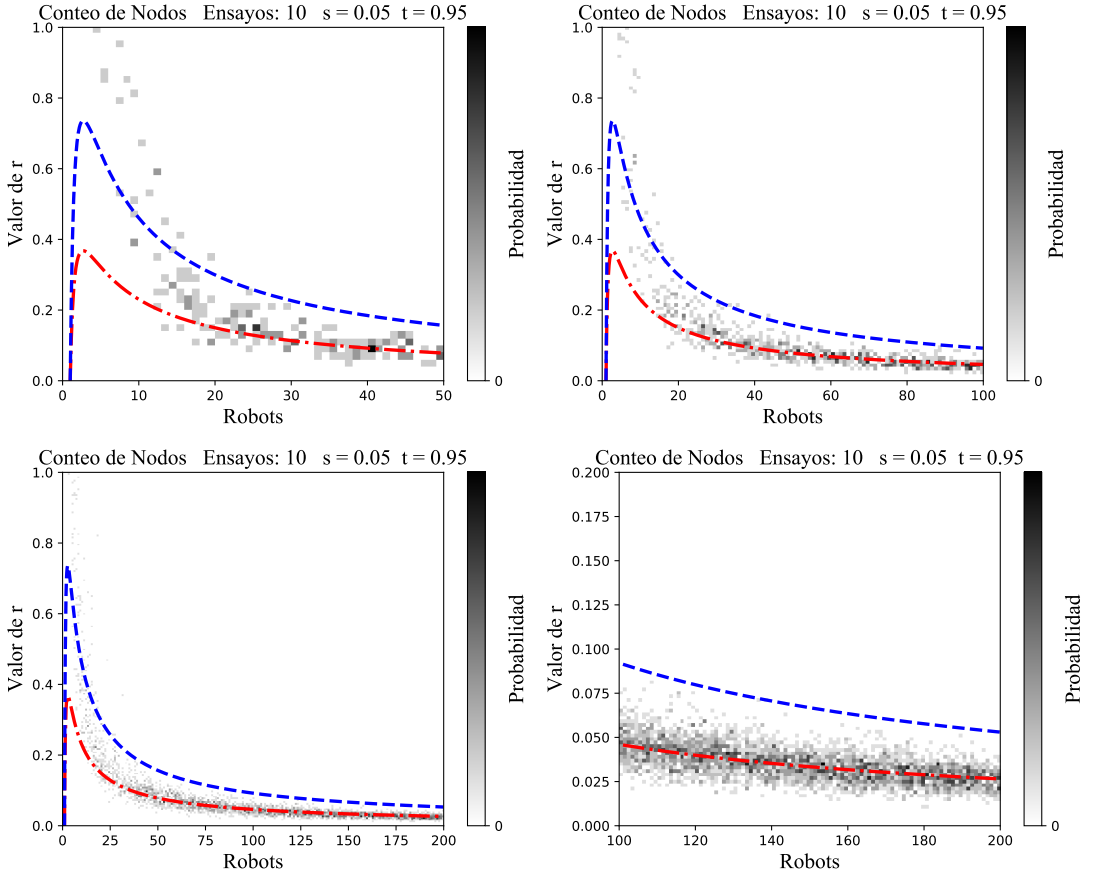


Figura 4.4: La curva en rojo (guión-punto) corresponde al umbral de Scheinerman $r = \frac{\log n}{n}$. La curva en azul (guión) corresponde al umbral de EDLA $r = \frac{2 \log n}{n}$. Los píxeles oscuros representan instancias que presumiblemente requieren que el algoritmo BRM explore una gran cantidad de nodos para resolver dichas instancias. En estas simulaciones $s = 0$, $t = 1$, $r_{\min} = 0$, $n_{\min} = 0, 100$, $r_{\max} = 1$, $n_{\max} = 50, 100, 200$ y $m = 10$.

Capítulo 5

Algoritmos parciales

En este capítulo se presentan algunos algoritmos parciales para resolver EDL cuyo tiempo de ejecución es $O(n \log_2 n)$. La justificación del tiempo es que los robots primero deben ser ordenados de acuerdo a $x_i - \rho_i$. Después de eso, cada robot es procesado a lo más un número constante de veces. La denominación *algoritmo parcial* se refiere a que dichos algoritmos siempre terminan, pero no siempre con la respuesta correcta. Cuando una instancia es calificada como negativa, existe la posibilidad de que se trate en realidad de un *falso negativo*, es decir, que en realidad sí sea soluble aunque el algoritmo no la haya resuelto.

La ventaja de estos algoritmos es que su tiempo de ejecución es $O(n \log_2 n)$ y cuando resuelven instancias aleatorias por encima del umbral de EDLA y por debajo del umbral de Scheinerman, lo hacen de manera correcta casi siempre. Esto da un paralelo a los algoritmos lineales propuestos en [54] para resolver instancias aleatorias de SAT con alta probabilidad, cuando estas instancias están «lejos» del umbral.

5.1. Algoritmo glotón

Como se ha visto en el capítulo 2, la mejor manera de utilizar un robot es cuando se encuentra detrás de los datos. De este modo, debe llegar a ellos, recogerlos y proseguir hacia adelante. Así, se tiene que $a_i = 0$. No se utiliza energía para regresar por los datos, y por lo tanto, se logra un máximo alcance. Esta idea se encuentra en el lema 2.2 así como en el algoritmo polinomial para resolver EDLA. Esta idea se utiliza en los algoritmos parciales que se presentan en esta sección. Cada vez que haya un robot en $A(d, I)^-$ éste se utiliza. Cuando $A(d, I)^- = \emptyset$ entonces se debe escoger algún robot en $A(d, I)^+$. El criterio para elegir un robot será mediante cinco enfoques glotones que a continuación se describen. En cada caso se menciona también un contraejemplo para ejemplificar la posibilidad de fallo de estos algoritmos.

Robot más cercano

Este algoritmo consiste en seleccionar el robot en $A(d, I)^+$ más cercano a d . La intuición es utilizar un robot que gaste la menor cantidad de energía para regresar por los datos. El contraejemplo está en la Figura 5.1.

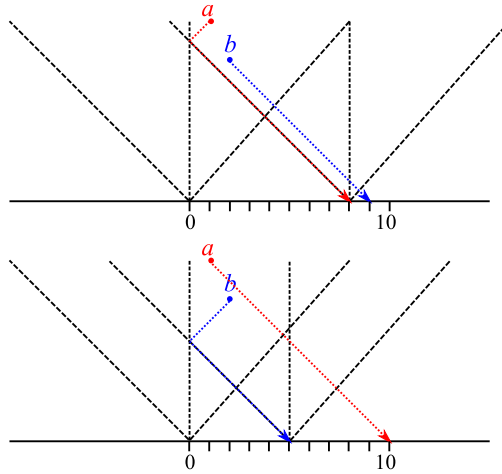


Figura 5.1: En esta instancia $I = \{a : (1, 9), b : (2, 7), s = 0, t = 10\}$. Resulta evidente que al utilizar primero el robot más cercano no se llega a la meta $t = 10$.

Menor alcance $x_i + \rho_i$

Este algoritmo consiste en seleccionar el robot en $A(d, I)^+$ cuyo alcance $x_i + \rho_i$ sea el menor. La intuición es utilizar primero el robot que no llega tan lejos, y utilizar después los que llegan más lejos. El contraejemplo está en la Figura 5.2.

Menor capacidad restante $x_i + \rho_i - d$

Este algoritmo consiste en seleccionar el robot en $A(d, I)^+$ cuya capacidad después de llegar a los datos $x_i + \rho_i - d$ sea la menor. La intuición es utilizar primero el robot que le quedaría poca energía después de llegar a los datos y utilizar después los que tendrían mayor energía restante. El contraejemplo es el mismo que el descrito en la Figura 5.2.

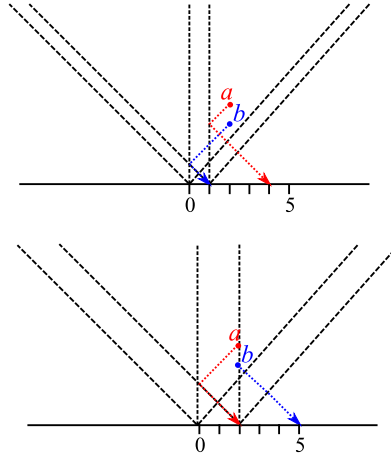


Figura 5.2: En esta instancia $I = \{a : (2, 4), b : (2, 3), s = 0, t = 5\}$. Resulta evidente que al utilizar primero el robot con menor alcance, no se llega a la meta $t = 5$.

Mayor capacidad restante $x_i + \rho_i - d$

Este algoritmo consiste en seleccionar el robot en $A(d)^+$ cuya capacidad después de llegar a los datos $x_i + \rho_i - d$ sea la mayor. Quizá parezca contra intuitivo, pero en realidad esta idea surge de la Figura 5.2, donde escoger el robot con mayor capacidad restante resultó mejor. El contraejemplo está en la Figura 5.3.

Menor a_i

Como se ha visto en el capítulo anterior, cuando $a_i = 0$ se maximiza la probabilidad de que dos intervalos asimétricos aleatorios se intersecten. A partir de este resultado surge este algoritmo, que consiste en seleccionar el robot en $A(d, I)^+$ cuya a_i para regresar por los datos sea la menor. Recuerdese que $a_i = (x_i - d)/\rho_i$. El contraejemplo está en la Figura 5.4.

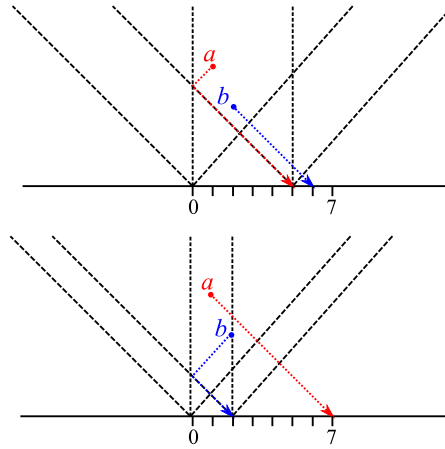


Figura 5.3: En esta instancia $I = \{a : (1,6), b : (2,4), s = 0, t = 7\}$. Resulta evidente que al utilizar primero el robot con mayor capacidad restante, no se llega a la meta $t = 7$.

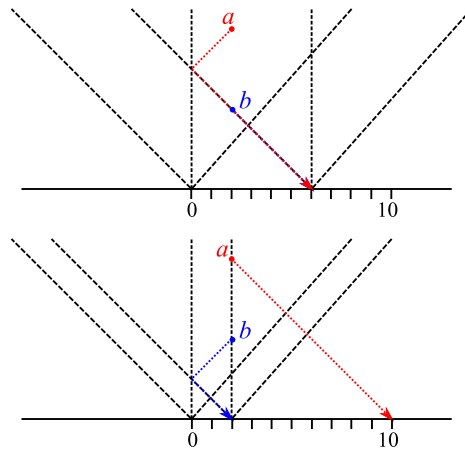


Figura 5.4: En esta instancia $I = \{a : (2,8), b : (2,4), s = 0, t = 10\}$. Para el robot a tenemos que $a_a = \frac{1}{4}$ mientras que para el robot b se tiene que $a_b = \frac{1}{2}$. Resulta evidente que al utilizar primero el robot con menor a_i , no se llega a la meta $t = 10$.

5.2. Algoritmos glotones inversos

En este capítulo se examina un enfoque para resolver EDL al procesar los robots *al revés*. Dada una instancia $I = \{(x_i, \rho_i), s, t\}_{i=1}^n$ se examina el conjunto de robots $A(t, I)$. En lugar de construir el certificado partiendo de s hasta llegar a t , se comienza de manera inversa. La idea es determinar el punto más a la izquierda de t que puede moverse un robot $(x_i, \rho_i) \in A(t, I)$ para recoger los datos en ese punto y traerlos hasta t . Este punto, que llamaremos t_1 , está determinado por la intersección de las rectas

$$\begin{aligned} x_i - \rho_i &= x - \rho \\ \rho &= -x + t \end{aligned}$$

Al resolver el sistema de ecuaciones se obtiene que $t_1 = \frac{1}{2}(t + x_i - \rho_i)$.

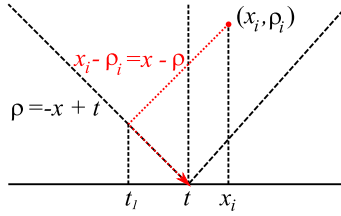


Figura 5.5: En este gráfico se aprecia el valor t_1 hasta donde el robot (x_i, ρ_i) puede regresar por los datos para llevarlos a t .

Ahora bien, ¿qué robot se debe escoger en $A(t)$? En este capítulo se examinan cinco criterios distintos con sus respectivos contraejemplos. Asimismo, se examina evidencia empírica y ciertos elementos teóricos que indican que elegir el robot que maximice $x_i - \rho_i$ es el mejor criterio. En la nueva terminal t_1 se elige un robot en $A(t_1, I)$ para determinar una nueva terminal t_2 como se ha descrito. En el momento en que alguna $t_k \leq s$ se detiene el algoritmo y se ha encontrado un certificado *al revés*. Si en algún momento $A(t_k, I) = \emptyset$ pero $t_k > s$, se detiene la búsqueda y no se ha podido encontrar una solución.

A continuación una definición necesaria y algunos lemas que ayudarán a escoger el mejor criterio.

Definición 5.1. Sea $I = \{(x_i, \rho_i), s, t\}_{i=1}^n$ una instancia de EDL. Se denota por

$$\alpha_i(d) = \begin{cases} \frac{1}{2}(d + x_i - \rho_i) & \text{si } (x_i, \rho_i) \in A(d) \\ d & \text{en otro caso} \end{cases}.$$

Ahora bien, si $B = \{i_m, i_{m-1}, \dots, i_1\}$ es una subsucesión de robots, se denota por

$$\alpha_B(d) = \alpha_{i_m, i_{m-1}, \dots, i_1}(d) = \alpha_{i_m}(\alpha_{i_{m-1}}(\dots \alpha_{i_1}(d))).$$

De acuerdo a lo discutido en el capítulo 2, ningún robot se utiliza dos veces sino únicamente una vez. Así que en este capítulo se mantiene la suposición de que en una subsucesión de robots $B = \{i_m, i_{m-1}, \dots, i_1\}$ no hay robots repetidos.

De la definición anterior, $\alpha_{j,i}(d)$ representa la posición después de haber utilizado en sentido inverso el robot i y el robot j . Es importante tener presente que se construye el certificado al revés.

Lema 5.1. Sea $\{(x_i, \rho_i), s, t\}_{i=1}^n$ una instancia de EDL, $d \in [s, t]$ y (x_i, ρ_i) un robot en $A(d, I)$. Entonces

$$x_i - \rho_i \leq \alpha_i(d) \leq d.$$

Demostración. Dado que $(x_i, \rho_i) \in A(d)$, eso significa que $x_i - \rho_i \leq d \leq x_i + \rho_i$, se tiene entonces que $d + x_i - \rho_i \leq 2d$, es decir, $\frac{1}{2}(d + x_i - \rho_i) \leq d$, lo cual demuestra que $\alpha_i(d) \leq d$. Por otra parte, $x_i - \rho_i \leq d$ implica que $2(x_i - \rho_i) \leq d + x_i - \rho_i$, es decir, $x_i - \rho_i \leq \frac{1}{2}(d + x_i - \rho_i) = \alpha_i(d)$. ■

Lema 5.2. Sea $I = \{(x_i, \rho_i), s, t\}_{i=1}^n$ una instancia de EDL y $a, b \in [s, t]$ tal que $a \leq b$. Entonces

$$\alpha_i(a) \leq \alpha_i(b)$$

para todo $(x_i, \rho_i) \in I$.

Demostración. En el primer caso, supóngase que $(x_i, \rho_i) \in A(a) \cap A(b)$, entonces $a \leq b \iff \frac{1}{2}(a + x_i - \rho_i) \leq \frac{1}{2}(b + x_i - \rho_i) \iff \alpha_i(a) \leq \alpha_i(b)$.

En el siguiente caso, supóngase que $(x_i, \rho_i) \notin A(a) \cup A(b)$. Se tiene por la definición 5.1 que $\alpha_i(a) = a \leq b = \alpha_i(b)$.

Supóngase ahora que $(x_i, \rho_i) \in A(a)$ pero $(x_i, \rho_i) \notin A(b)$. Por el lema 5.1 y la definición 5.1 se tiene que $\alpha_i(a) \leq a \leq b = \alpha_i(b)$.

En el último caso, $(x_i, \rho_i) \notin A(a)$ pero $(x_i, \rho_i) \in A(b)$. Esto es equivalente a decir que $a < x_i - \rho_i \leq b \leq x_i + \rho_i$, pero por la definición 5.1 y por el lema 5.1 se tiene que $\alpha_i(a) = a < x_i - \rho_i \leq \alpha_i(b)$. ■

Lema 5.3. Sea $I = \{(x_i, \rho_i), s, t\}_{i=1}^n$ una instancia de EDL y $a, b \in [s, t]$ tal que $a \leq b$. Sea $B = \{i_m, i_{m-1}, \dots, i_1\}$ una subsucesión de robots, entonces

$$\alpha_B(a) \leq \alpha_B(b).$$

Demostración. Se procede por inducción sobre m . Por el lema 5.2 $\alpha_{i_1}(a) \leq \alpha_{i_1}(b)$. Supóngase que es cierto $\alpha_{i_{m-1}, \dots, i_1}(a) \leq \alpha_{i_{m-1}, \dots, i_1}(b)$. Se aplica nuevamente el lema 5.2 y se obtiene que $\alpha_{i_m}(\alpha_{i_{m-1}, \dots, i_1}(a)) \leq \alpha_{i_m}(\alpha_{i_{m-1}, \dots, i_1}(b))$, es decir,

$$\alpha_{i_m, i_{m-1}, \dots, i_1}(a) \leq \alpha_{i_m, i_{m-1}, \dots, i_1}(b).$$

■

Se llega así al lema importante.

Lema 5.4. Sea $I = \{(x_i, \rho_i), s, t\}_{i=1}^n$ una instancia de EDL, $d \in [s, t]$ y sean i, j dos robots tal que $i \in A(d)$ y $x_j - \rho_j \leq x_i - \rho_i$. Entonces se tiene que

$$\alpha_{j,i}(d) \leq \alpha_{i,j}(d).$$

Demostración. En primer lugar, si $(x_j, \rho_j) \in A(d)$ se implica que $(x_j, \rho_j) \in A(\alpha_i(d))$ puesto que $x_j - \rho_j \leq x_i - \rho_i$. Ahora bien, se tienen cuatro casos resumidos en la siguiente tabla

	$A(d)$	$A(\alpha_i(d))$	$A(\alpha_j(d))$
caso 1	i, j	i, j	i, j
caso 2	i, j	i, j	j
caso 3	i	i, j	i
caso 4	i	i	i

En el primer caso, $(x_i, \rho_i) \in A(\alpha_j(d))$ y $(x_j, \rho_j) \in A(d)$ por lo cual

$$\alpha_{i,j}(d) = \frac{1}{2} \left(\frac{1}{2} (d + x_j - \rho_j) + x_i - \rho_i \right) = \frac{1}{4} (x_j - \rho_j) + \frac{1}{2} (x_i - \rho_i) + \frac{d}{4}.$$

Análogamente para $\alpha_{j,i}(d)$, por lo cual

$$\begin{aligned} \alpha_{i,j}(d) - \alpha_{j,i}(d) &= \frac{1}{4} [(x_j - \rho_j) - (x_i - \rho_i)] + \frac{1}{2} [(x_i - \rho_i) - (x_j - \rho_j)] \\ &= \frac{1}{4} (x_i - \rho_i) - \frac{1}{4} (x_j - \rho_j) \end{aligned}$$

Entonces $0 \leq \frac{1}{4} (x_i - \rho_i) - \frac{1}{4} (x_j - \rho_j) \iff 0 \leq (x_i - \rho_i) - (x_j - \rho_j) \iff x_j - \rho_j \leq x_i - \rho_i$, lo cual es cierto por hipótesis. Por lo tanto $0 \leq \alpha_{i,j}(d) - \alpha_{j,i}(d)$ y en consecuencia $\alpha_{j,i}(d) \leq \alpha_{i,j}(d)$.

Para el segundo caso, $(x_i, \rho_i) \notin A(\alpha_j(d))$ y $(x_j, \rho_j) \in A(d)$. Se tiene entonces que $\alpha_{i,j}(d) = \alpha_i(\alpha_j(d)) = \alpha_j(d)$ por la definición 5.1. Ahora bien, si se aplica el lema 5.1 se obtiene que $\alpha_i(d) \leq d$. Después se aplica el lema 5.2 y se obtiene $\alpha_{j,i}(d) \leq \alpha_j(d)$. Pero $\alpha_j(d) = \alpha_{i,j}(d)$ y en consecuencia $\alpha_{j,i}(d) \leq \alpha_{i,j}(d)$.

Para el tercer caso, $(x_j, \rho_j) \notin A(d)$ pero $(x_j, \rho_j) \in A(\alpha_i(d))$. Entonces $\alpha_{j,i}(d) = \alpha_j(\alpha_i(d)) \leq \alpha_i(d)$ por el lema 5.1. Por otro lado $\alpha_{i,j}(d) = \alpha_i(\alpha_j(d)) = \alpha_i(d)$ por la definición 5.1 y en consecuencia $\alpha_{j,i}(d) \leq \alpha_{i,j}(d)$.

Para el cuarto caso, $(x_j, \rho_j) \notin A(d) \cup A(\alpha_i(d))$, entonces

$$\alpha_{j,i}(d) = \alpha_j(\alpha_i(d)) = \alpha_i(d) \leq \alpha_i(d) = \alpha_i(\alpha_j(d)) = \alpha_{i,j}(d),$$

al utilizar la definición 5.1. ■

El siguiente lema conecta la solubilidad de una instancia EDL con la definición 5.1.

Lema 5.5 (Lema de equivalencia). *Sea $I = \{(x_i, \rho_i), s, t\}_{i=1}^n$ una instancia de EDL. I es soluble si y sólo si existe una subsucesión $B = \{i_1, \dots, i_{m-1}, i_m\}$ tal que $\alpha_B(t) \leq s$.*

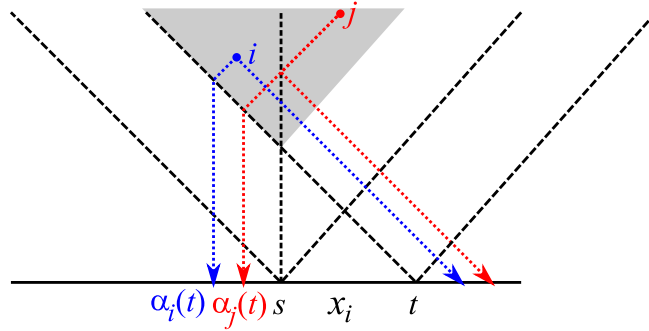


Figura 5.6: En este gráfico se observa una instancia $I = \{(x_i, \rho_i), (x_j, \rho_j), s, t\}$. La instancia se puede resolver con un sólo robot, i ó j . En ambos casos los robots se encuentran en el área sombreada delimitada por las rectas $p = -x + t$ y $p = x - (2s - t)$. Las flechas a la derecha indican hasta dónde pueden llevar los datos desde s . Las flechas a la izquierda indican desde dónde pueden recoger los datos para llevarlos hasta t .

Demostración. **A. I es soluble \Rightarrow existe una subsucesión $B = \{i_1, \dots, i_{m-1}, i_m\}$ tal que $\alpha_B(t) \leq s$.**

Se procede por inducción. Sea $B = \{i_1\}$ un certificado para I . Existen dos casos.

1. En el primero, $x_{i_1} \leq s$. Puesto que i puede llevar los datos al menos hasta t , entonces $t \leq x_{i_1} + \rho_{i_1}$. Es claro también que $x_{i_1} - \rho_{i_1} \leq t$, por lo tanto $i_1 \in A(t)$ y de acuerdo a la definición 5.1 $\alpha_{i_1}(t) = \frac{1}{2}(t + x_{i_1} - \rho_{i_1})$. Ahora, es claro que $\rho_{i_1} \geq t - s$ lo cual implica que $x_{i_1} - \rho_{i_1} \leq s - (t - s)$ de donde se sigue que $\frac{1}{2}(t + x_{i_1} - \rho_{i_1}) \leq s$, es decir $\alpha_{i_1}(t) \leq s$.
2. Ahora supóngase que $s < x_{i_1}$. Puesto que i_1 tiene capacidad ρ_{i_1} para regresar desde x_{i_1} a s y de ahí llegar hasta t se tiene que $\rho_{i_1} - (x_{i_1} - s) \geq t - s$ de donde se sigue que $\frac{1}{2}(t + x_{i_1} - \rho_{i_1}) \leq s$, es decir $\alpha_{i_1}(t) \leq s$.
3. Supóngase ahora que la hipótesis es cierta para $m - 1$ robots. Sea $B = \{i_1, \dots, i_{m-1}, i_m\}$ un certificado para I . Sea $t' \in [s, t]$ la posición donde el robot i_{m-1} deja los datos para que los recoja i_m . Por hipótesis de inducción se tiene que $\alpha_{i_1, \dots, i_{m-1}}(t') \leq s$. Por último, se aplican los argumentos 1. y 2. al robot i_m para obtener que $\alpha_{i_m}(t) \leq t'$. De este modo, si se aplica el lema 5.2 a la desigualdad anterior, se obtiene

$$\alpha_{i_1, \dots, i_{m-1}}(\alpha_{i_m}(t)) \leq \alpha_{i_1, \dots, i_{m-1}}(t') \leq s$$

$$\alpha_B(t) \leq s.$$

B. Existe una subsucesión $B = \{i_1, \dots, i_{m-1}, i_m\}$ tal que $\alpha_B(t) \leq s \Rightarrow I$ es soluble.

1. Se procede también por inducción. Sea $B = \{i_1\}$ una subsucesión de robots tal que $\alpha_{i_1}(t) \leq s$. Entonces $\frac{1}{2}(t + x_{i_1} - \rho_{i_1}) \leq s$ de donde se sigue que

$$x_{i_1} - \rho_{i_1} \leq s - (t - s).$$

Se tienen dos casos. En el primer caso $x_{i_1} \leq s$. Se sigue de la desigualdad anterior que $\rho_{i_1} \geq t - s$. En el segundo caso $x_{i_1} > s$. Se sigue que $\rho_{i_1} - (x_{i_1} - s) \geq t - s$. En ambos casos i_1 puede llevar los datos desde s a t , por lo que $B = \{i_1\}$ es un certificado para I .

2. Suponer ahora que la hipótesis es cierta para $m - 1$ robots. Sea $B = \{i_1, \dots, i_{m-1}, i_m\}$ una subsucesión tal que $\alpha_B(t) \leq s$. Por hipótesis de inducción se tiene que $B' = \{i_1, \dots, i_{m-1}\}$ es un certificado para la instancia $I' = \{(x_i, \rho_i), s, \alpha_{i_m}(t)\}_{i=1}^n$. Por la definición 5.1 el robot i_m puede llevar los datos desde $\alpha_{i_m}(t)$ hasta t . De este modo, $B = \{i_1, \dots, i_{m-1}, i_m\}$ es un certificado para la instancia original $I = \{(x_i, \rho_i), s, t\}_{i=1}^n$.

■

Menor x_i

El primer criterio es escoger el robot con la posición x_i más cercana al inicio s . De esta manera se minimiza su a_i , lo cual es deseable para efectos de tener un mayor intervalo

asimétrico asociado. Así, presumiblemente existiría mayor probabilidad de que la gráfica de intervalos con el modelo asimétrico sea conexa. En la Figura 5.7 se encuentra el contraejemplo.

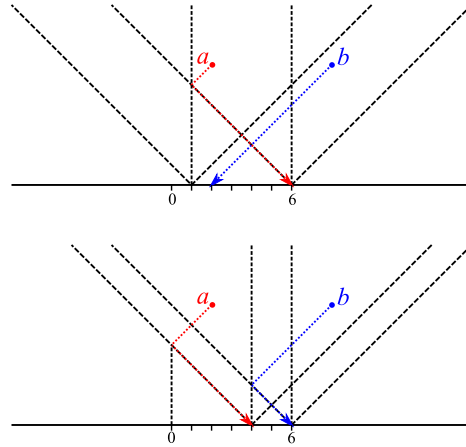


Figura 5.7: En esta instancia $I = \{a : (2, 6), b : (8, 6), s = 0, t = 6\}$. Es claro que $a, b \in A(d, I)$. Si se elige, en sentido inverso, primero el robot a y luego el robot b la subsucesión de robots es $\{b, a\}$. De este modo es imposible resolver la instancia. En cambio, al elegir en sentido inverso primero el robot b y luego el a se logra resolver la instancia y el certificado es $\{a, b\}$.

Mayor x_i

El segundo criterio es escoger el robot con la posición x_i más cercana a la terminal t . La intuición de esta elección es utilizar al último los robots más cercanos a t (recuérdese que el certificado se va construyendo en sentido inverso). En la Figura 5.8 se encuentra el contraejemplo.

Menor $\frac{x_i + \rho_i - d}{2}$

El tercer criterio es escoger el robot con el valor $\frac{x_i + \rho_i - d}{2}$ más pequeño, que se refiere a la distancia que el robot puede regresar por los datos para después llevarlos a d . La intuición de esta elección es realizar avances pequeños y utilizar la mayor cantidad de robots. El contraejemplo es el mismo que el de la Figura 5.7.

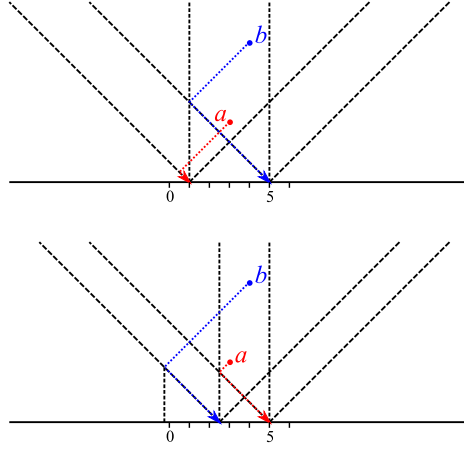


Figura 5.8: En esta instancia $I = \{a : (3,3), b : (4,7), s = 0, t = 5\}$. Es claro que $a, b \in A(d, I)$. Si se elige en sentido inverso primero el robot b , por estar más cerca de $t = 5$, el robot a no puede regresar por los datos más allá de $d = 0.5$. En cambio, al elegir en sentido inverso primero el robot a y luego el b se logra resolver la instancia y el certificado es $\{b, a\}$.

Mayor $\frac{x_i + p_i - d}{2}$

El cuarto criterio es escoger el robot con el valor $\frac{x_i + p_i - d}{2}$ más grande, que se refiere a la distancia que el robot puede regresar por los datos para después llevarlos a d . La motivación es porque algunas veces es mejor elegir de esta manera, como se vió en el contraejemplo de la Figura 5.7. En la Figura 5.9 se encuentra el contraejemplo.

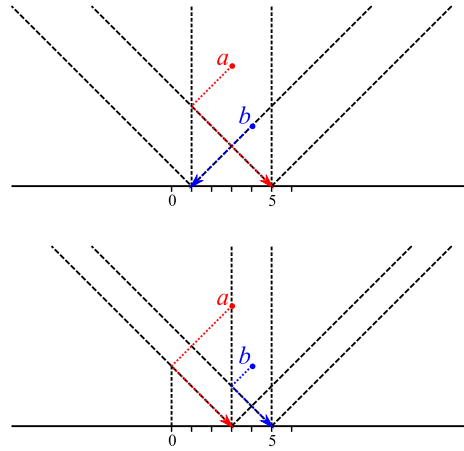


Figura 5.9: En esta instancia $I = \{a : (3, 6), b : (4, 3), s = 0, t = 5\}$. Cuando $d = t$ los robots $a, b \in A(d)$. Si elegimos en sentido inverso primero el robot a , por tener el valor $\frac{x_i + \rho_i - d}{2}$ más grande, el robot b no puede regresear por los datos más allá de $d = 1$. En cambio, al elegir en sentido inverso primero el robot b y luego el a se logra resolver la instancia y el certificado es $\{a, b\}$.

Menor $x_i - \rho_i$ (HLR)

El quinto criterio es escoger el robot con el máximo alcance inferior $x_i - \rho_i$ (HLR por sus siglas en inglés "highest lower reach"), que se refiere al punto más atrás que el robot puede alcanzar. Este criterio ha demostrado ser el más efectivo de todos y no sin una justa razón. Como se ha visto en los cuatro criterios anteriores, bastaron dos robots para construir contraejemplos. Esto es imposible con este último criterio, pues el lema 5.4 garantiza que al resolver instancias de tamaño $n = 2$ el algoritmo HLR es óptimo. No obstante esta ventaja sobre los demás criterios, sigue siendo un criterio *miope* pues no es la estrategia óptima para más de dos robots, como se aprecia en el contraejemplo de la Figura 5.10.

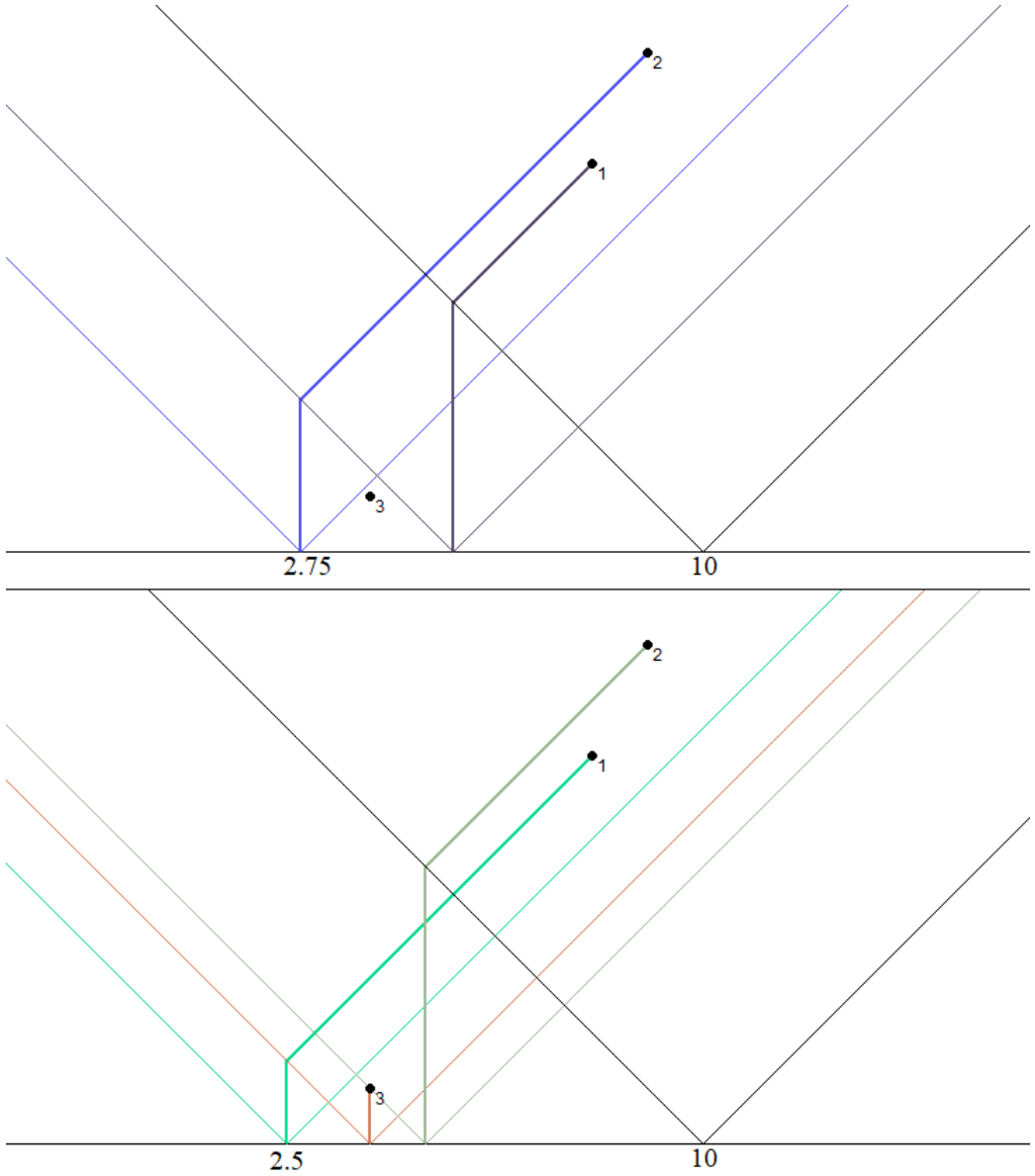


Figura 5.10: En esta instancia $I = \{i_1 : (8, 7), i_2 : (9, 9), i_3 : (4, 1), s = 2.5, t = 10\}$. Si se utiliza el algoritmo HLR, se obtiene la subsucesión $\{2, 1\}$ tal que $\alpha_{i_2 i_1}(10) = 2.75$ como lo muestra el gráfico superior. Por otro lado, $\alpha_{i_1 i_3 i_2}(10) = 2.5$ como lo muestra el gráfico inferior, lo cual indica que $\{i_1, i_3, i_2\}$ es un certificado para I . La razón por la que HLR falla en esta instancia es porque ignora al robot i_3 debido a su pequeña capacidad.

Capítulo 6

Estudio experimental de eficiencia

6.1. Algoritmos parciales y BRM

En esta sección se presenta el comparativo entre los algoritmos parciales y la búsqueda aleatoria simple con respecto al algoritmo BRM. Las simulaciones se realizaron de una manera similar a las presentadas en el capítulo 4. En el espacio de parámetros (n, r) se realizó una discretización, a manera de una rejilla, y se generó una instancia aleatoria para cada pareja de valores (n, r) . Se aplicaron nueve algoritmos parciales descritos en el capítulo anterior y se añadió la búsqueda aleatoria simple descrita en el capítulo 3. En cada simulación, se resolvió también cada instancia con el Algoritmo 4.3 (BRM). Dado que el Algoritmo 4.3 es exacto, se pueden contabilizar los falsos negativos que se obtuvieron con los algoritmos parciales. Un falso negativo es cuando una instancia se declara insoluble por un algoritmo parcial cuando en realidad es soluble. En los gráficos 6.1-6.4 se ilustra la zona en el espacio de parámetros donde aparecen los falsos negativos con mayor frecuencia. La figura 6.3 resume los resultados de la simulación. Para cada instancia, representada por una pareja (n, r) , se contabilizó el número de falsos negativos dados por los algoritmos parciales y la búsqueda aleatoria simple. El color negro representa una probabilidad empírica alta de obtener un falso negativo. El color blanco representa una probabilidad empírica baja de obtener un falso negativo.

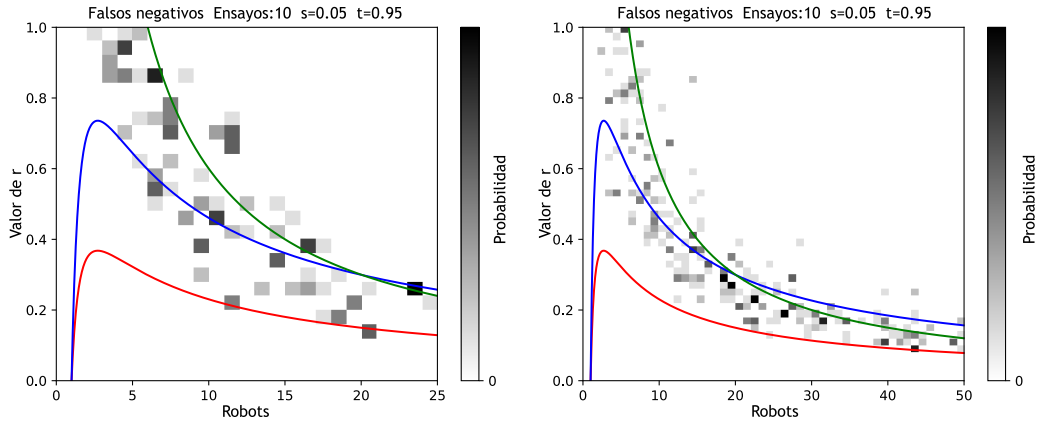


Figura 6.1: a) Simulación de 625 instancias aleatorias, cada una con parámetros $(n, r) \in [1, \dots, 25] \times [\frac{1}{25}, \dots, 1]$. Se determinó con BRM que 347 instancias son solubles. b) Simulación de 2500 instancias aleatorias, cada una con parámetros $(n, r) \in [1, \dots, 50] \times [\frac{1}{50}, \dots, 1]$. Se determinó con BRM que 1769 instancias son solubles.

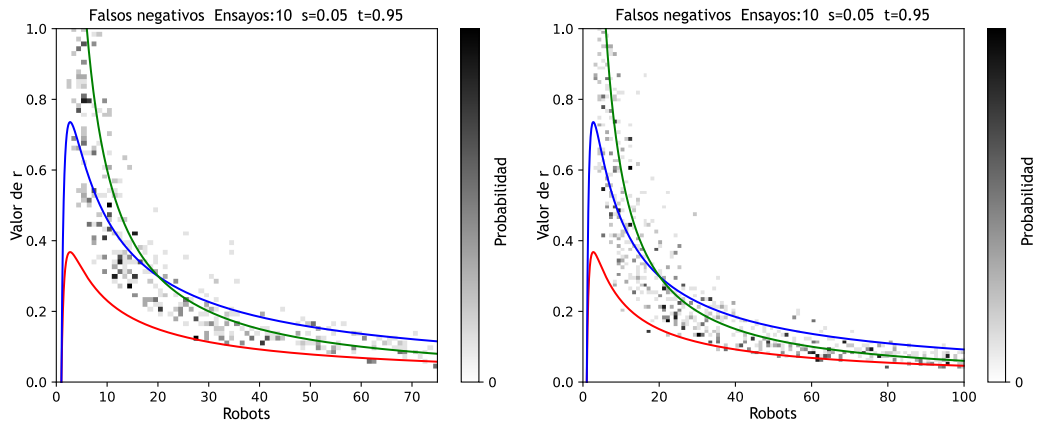


Figura 6.2: c) Simulación de 5625 instancias aleatorias, cada una con parámetros $(n, r) \in [1, \dots, 75] \times [\frac{1}{75}, \dots, 1]$. Se determinó con BRM que 4345 instancias son solubles. d) Simulación de 10,000 instancias aleatorias, cada una con parámetros $(n, r) \in [1, \dots, 100] \times [\frac{1}{100}, \dots, 1]$. Se determinó con BRM que 8130 instancias son solubles.

Los resultados ilustrados en las figuras 6.1 y 6.2 se resumen en la tabla 6.3. En esta tabla, cada renglón resume los resultados por algoritmo. Cada columna se refiere a cada una de las cuatro simulaciones. Las celdas contienen el número de instancias resueltas afirmativamente en términos absolutos y relativos al BRM. Al final de la tabla se colocaron los promedios y las desviaciones estándar de las observaciones.

	$n \leq 25$		$n \leq 50$		$n \leq 75$		$n \leq 100$	
$\min x_i + \rho_i$	346	99.7 %	1760	99.4 %	4334	99.7 %	8107	99.7 %
RS	307	88.4 %	1642	92.8 %	4137	95.2 %	7780	95.6 %
$\min x_i$	319	91.9 %	1684	95.2 %	4216	97.0 %	7893	97.0 %
$\max \frac{x_i + \rho_i - d}{2}$	332	95.6 %	1729	97.7 %	4288	98.6 %	8044	98.9 %
LC	317	91.3 %	1714	96.8 %	4262	98.0 %	7992	98.3 %
$\min x_i + d $	345	99.4 %	1764	99.7 %	4331	99.6 %	8101	99.6 %
HC	339	97.6 %	1753	99.1 %	4310	99.1 %	8071	99.2 %
HLR	347	100 %	1769	100 %	4345	100 %	8130	100 %
$\max x_i$	344	99.1 %	1759	99.4 %	4327	99.5 %	8090	99.5 %
$\min \frac{x_i + \rho_i - d}{2}$	326	93.9 %	1699	96.0 %	4250	97.8 %	7967	98.0 %
Promedio	332.2	95.7 %	1727.3	97.6 %	4280	98.5 %	8017.4	98.6 %
Des. Est.	14.3	4.1 %	42.1	2.4 %	65.6	1.5 %	111.5	1.4 %

Figura 6.3: En esta tabla se resumen los resultados de cuatro simulaciones. En cada celda se registra el número de instancias resueltas afirmativamente. Cada renglón corresponde a un algoritmo diferente. En cada columna, el valor n se refiere al número máximo de robots que se consideró en cada simulación.

Como se puede observar en la tabla 6.3, prácticamente todos los algoritmos tuvieron una aproximación al método exacto (Algoritmo 4.3) por encima del 95 %. Esto sugiere que la gran mayoría de las instancias en el espacio de parámetros se pueden resolver con facilidad.

De acuerdo al capítulo 3, por encima del umbral de EDLA $r = \frac{2 \log n}{n}$ las instancias aleatorias tienen una probabilidad muy alta de ser resueltas afirmativamente en tiempo polinomial. Por otro lado, las instancias aleatorias por debajo del umbral de Scheinerman $r = \frac{\log n}{n}$ tienen una alta probabilidad de ser desconexas con el modelo simétrico, lo cual se verifica en tiempo polinomial, de tal modo que son insolubles con alta probabilidad. Las instancias *difíciles* se encuentran con alta probabilidad entre los dos umbrales mencionados.

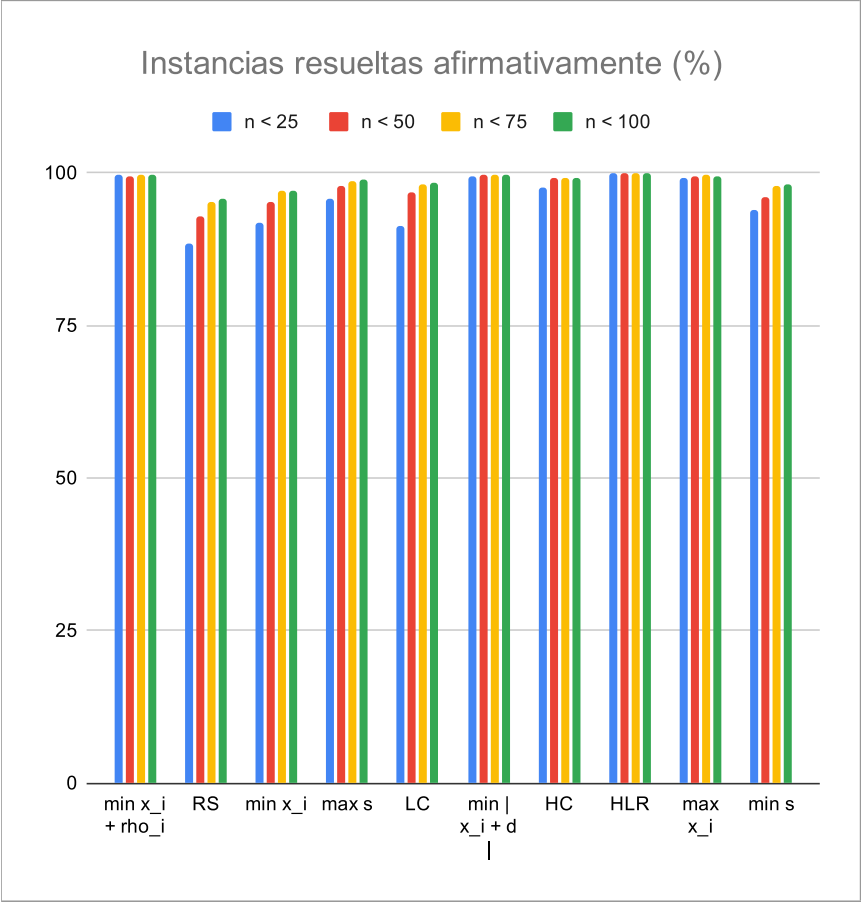


Figura 6.4: En esta gráfica de barras se muestran los resultados porcentuales de la tabla 6.3.

En el siguiente grupo de simulaciones se restringió a evaluar únicamente instancias aleatorias cuyos parámetros (n, r) se encuentran entre los dos umbrales ya mencionados. Gráficamente, las celdas que se encuentren entre la curva azul (umbral EDLA) y la curva roja (umbral Scheinerman) representan estas instancias aleatorias.

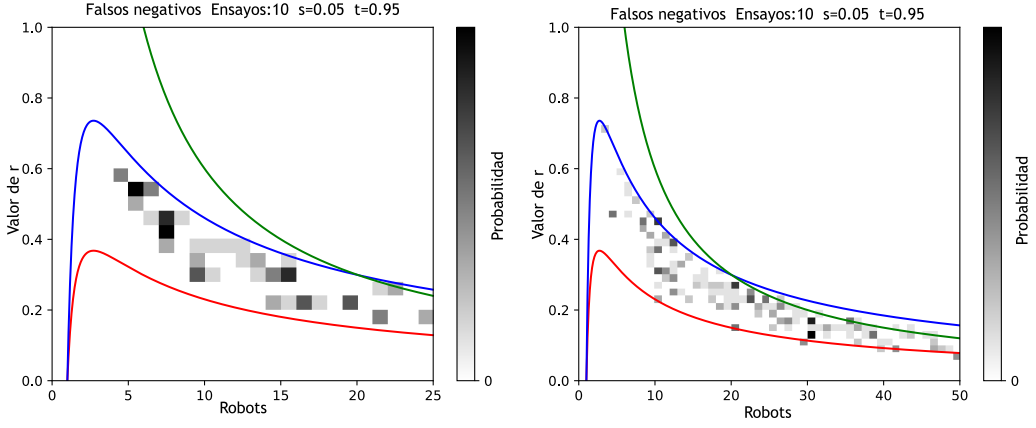


Figura 6.5: a) Simulación de 151 instancias aleatorias con $(n, r) \in [1, \dots, 25] \times [\frac{1}{25}, \dots, 1]$ y $\frac{\log n}{n} \leq r \leq \frac{2 \log n}{n}$. Se determinó con BRM que 46 instancias son solubles. b) Simulación de 425 instancias aleatorias con $(n, r) \in [1, \dots, 50] \times [\frac{1}{50}, \dots, 1]$ y $\frac{\log n}{n} \leq r \leq \frac{2 \log n}{n}$. Se determinó con BRM que 166 instancias son solubles.

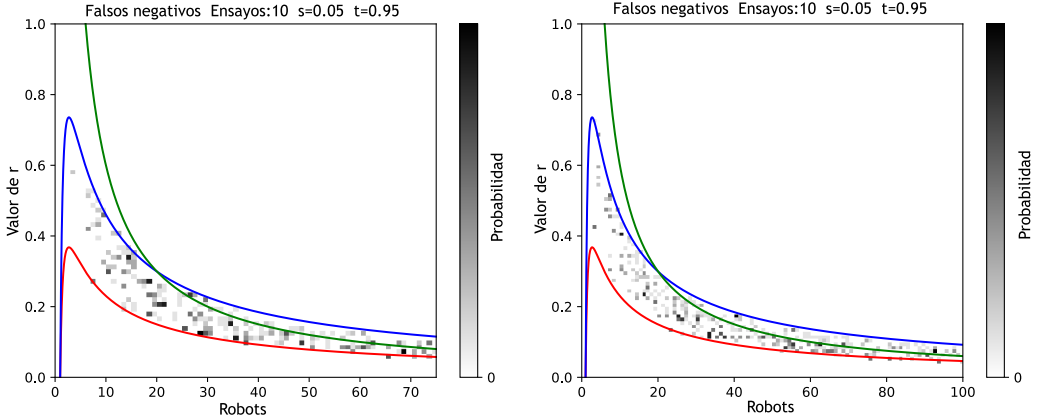


Figura 6.6: c) Simulación de 766 instancias aleatorias con $(n, r) \in [1, \dots, 75] \times [\frac{1}{75}, \dots, 1]$ y $\frac{\log n}{n} \leq r \leq \frac{2 \log n}{n}$. Se determinó con BRM que 311 instancias son solubles. d) Simulación de 1152 instancias aleatorias con $(n, r) \in [1, \dots, 100] \times [\frac{1}{100}, \dots, 1]$ y $\frac{\log n}{n} \leq r \leq \frac{2 \log n}{n}$. Se determinó con BRM que 495 instancias son solubles.

A continuación se resumen los resultados en la tabla 6.7 y en el gráfico 6.8.

	$n \leq 25$		$n \leq 50$		$n \leq 75$		$n = 100$	
$\min x_i + \rho_i$	45	97.8 %	163	98.1 %	298	95.8 %	484	97.7 %
RS	23	50.0 %	73	43.9 %	171	54.9 %	282	56.9 %
$\min x_i$	32	69.5 %	122	73.4 %	224	72.0 %	359	72.5 %
$\max \frac{x_i + \rho_i - d}{2}$	38	82.6 %	142	85.5 %	280	90.0 %	442	89.2 %
LC	35	76.0 %	130	78.3 %	243	78.1 %	415	83.8 %
$\min x_i + d $	46	100 %	164	98.8 %	303	97.4 %	484	97.7 %
HC	42	91.3 %	156	93.9 %	292	93.8 %	471	95.1 %
HLR	46	100 %	166	100 %	311	100 %	495	100 %
$\max x_i$	44	95.6 %	159	95.7 %	295	94.8 %	474	95.7 %
$\min \frac{x_i + \rho_i - d}{2}$	36	78.2 %	132	79.5 %	243	78.1 %	400	80.8 %
Promedio	38.7	84.1 %	140.7	84.8 %	266	85.5 %	430.6	87 %
Des. Est.	7.4	16.2 %	28.6	17.2 %	44.7	14.4 %	68.2	13.8 %

Figura 6.7: En esta tabla se resumen los resultados de cuatro simulaciones. En cada celda se registra el número de instancias resueltas afirmativamente. Cada renglón corresponde a un algoritmo diferente. En cada columna, el valor n se refiere al número máximo de robots que se consideró en cada simulación.

Resulta evidente en las simulaciones que el algoritmo HLR es el ganador indiscutible. Además, debe ser notorio que se resolvieron decenas de miles de instancias aleatorias con el algoritmo HLR sin encontrar un solo falso negativo. Esta observación sugirió inicialmente la posibilidad de que HLR fuera un algoritmo exacto, lo cual dio como resultado los lemas 5.1-5.5. Al tratar de demostrar un lema que hacía falta surgió el contraejemplo mostrado en la Figura 5.11. Debido a esto, se considera que HLR resolverá prácticamente cualquier instancia con una probabilidad muy baja de dar un falso negativo en el modelo probabilístico que se ha presentado. En este sentido, se concluye esta sección con el último grupo de simulaciones, donde se pretende ilustrar el umbral de EDL con mayor definición al utilizar HLR para aproximar el algoritmo exacto BR. Esta tarea resultaba impráctica para valores de $n \geq 100$ porque el tiempo de BR para resolver algunas instancias crece exponencialmente.

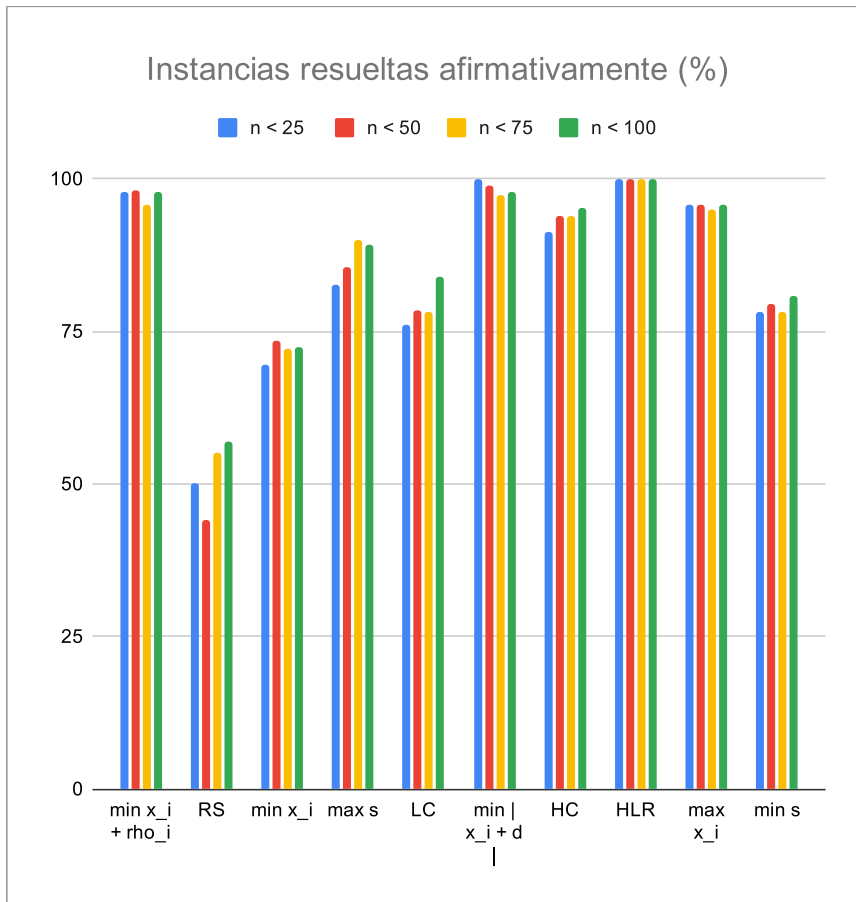


Figura 6.8: En esta gráfica de barras se muestran los resultados de la tabla 6.5.

La libreta de Python **Difficult_EDL.ipynb** para acceder al código y realizar las simulaciones: https://colab.research.google.com/drive/10A1vursHf-1DGyT0R41zAT_EckKG0KgC?usp=sharing

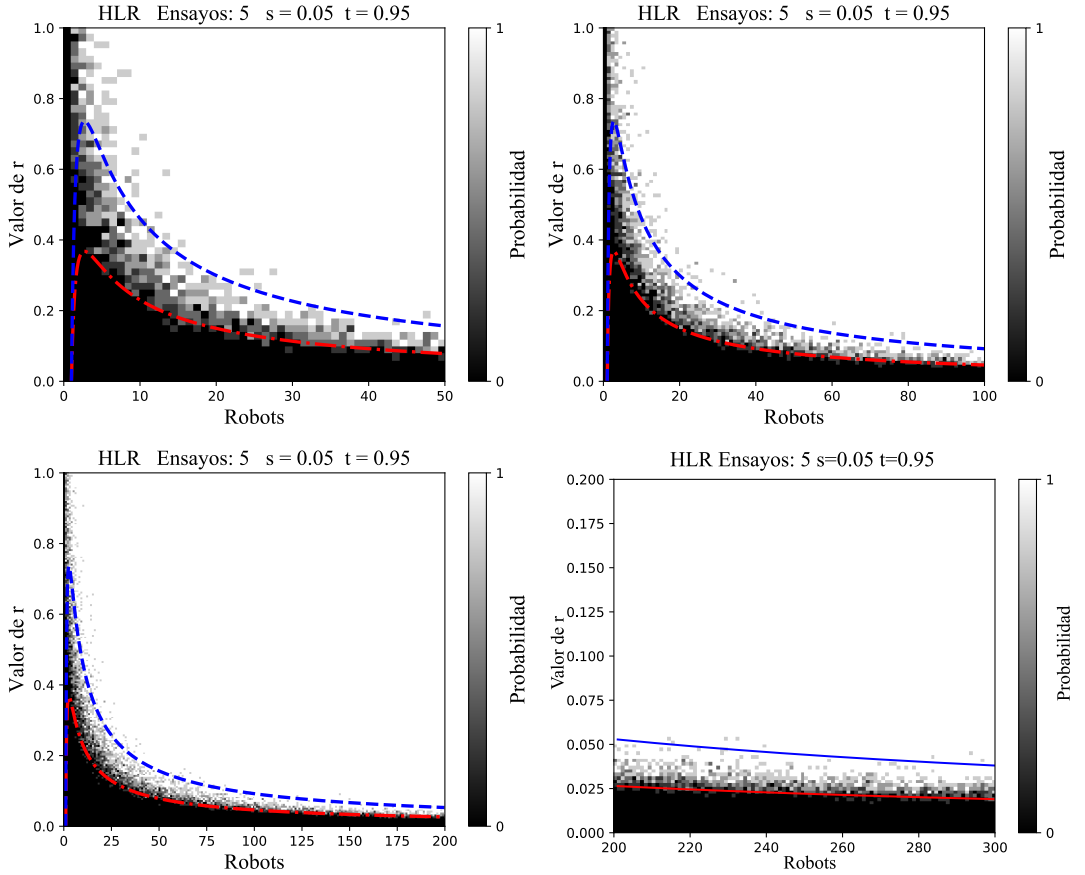


Figura 6.9: En este grupo de simulaciones se utilizó el algoritmo HLR para vislumbrar de manera aproximada el umbral de EDL con una mejor resolución que en la Figura 4.3.

La libreta de Python **HLR_DensityPlot.ipynb** para acceder al código y realizar las simulaciones: <https://colab.research.google.com/drive/1TBDraDFLsR7fT9auYK5dasNzu-vlusp=sharing>

La conjetura final es que existe $1 \leq \alpha_c \leq 2$ tal que el umbral de transición para EDL está dado por $r = \frac{\alpha_c \log n}{n}$. Dicho de otro modo, la función que describe el umbral de EDL se conjetura como $\Theta\left(\frac{\log n}{n}\right)$.

6.2. Optimización en EDL

Inicialmente, el problema EDL se abordó como uno de optimización. La pregunta que se plantea en este caso es: dado un conjunto de robots $\{(x_i, \rho_i)\}_{i=1}^n$ y una salida s ¿cuál es el punto más lejano a donde se pueden trasladar los datos? Para resolver esta pregunta de manera aproximada se puede utilizar el siguiente enfoque. Primero, se define la *cota superior* $cs = \max \{x_i + \rho_i\}_{i=1}^n$. Debe ser claro que los datos no pueden ser llevados más allá de este punto. Lo que en este trabajo se ha manejado como el certificado, es decir, una subsucesión de robots en el orden en que deben moverse, será el homólogo a una solución para la versión de optimización. El movimiento de los robots es como el que se definió en el capítulo 2. En segundo lugar, se generan instancias aleatorias como se definió en el capítulo 3. La pregunta ya no es en este caso ¿se pueden llevar los datos de s a t ?, sin embargo, para resolver aproximadamente la versión de optimización se resuelven de manera sucesiva a lo más n problemas de decisión. Esto se logra al particionar el intervalo $[0, cs]$ en n pedazos de la misma longitud, $p_j = \frac{j*cs}{n}$ para $j = 0, 1, \dots, n$. Así pues, se resuelven los problemas de decisión para el siguiente grupo de instancias

$$I_j = (\{(x_i, \rho_i)\}_{i=0}^n, s = 0, t = p_j)$$

Se comienza con p_n y se continua con p_{n-1} hasta llegar, si es necesario a p_0 . El primer índice j donde la respuesta es afirmativa define un intervalo $[p_j, p_{j+1}] \subset [0, cs]$ donde se encuentra el alcance óptimo opt . Si el índice es n entonces $opt = p_n$. En el siguiente grupo de simulaciones, realizadas de manera similar a aquéllas del capítulo 3 y 4, se grafica el valor $\frac{j}{n}$, donde j es el primer índice tal que I_j es una instancia afirmativa. Los gráficos de las simulaciones se encuentran en la Figura 6.10.

Debido a su eficacia empírica y a su eficiencia, se escogió el algoritmo HLR para estas simulaciones. El tiempo de ejecución es $O(n^2 \log n)$ para cada instancia. Aunque la probabilidad de que HLR dé un falso negativo es positiva, en la práctica es muy baja. Si al utilizar el algoritmo descrito en esta sección se sustituye HLR por el algoritmo BR entonces se tiene la garantía de que $opt - p_j < \frac{cs}{n} \leq \frac{1+r}{n}$. Evidentemente si se desea resolver de manera exacta la versión de optimización se puede recurrir a una versión modificada de BR. El objetivo ahora es explorar todo el árbol y guardar el registro del alcance en cada una de sus ramas. El valor de opt será el valor más alto de todas las hojas del árbol generado por BR.

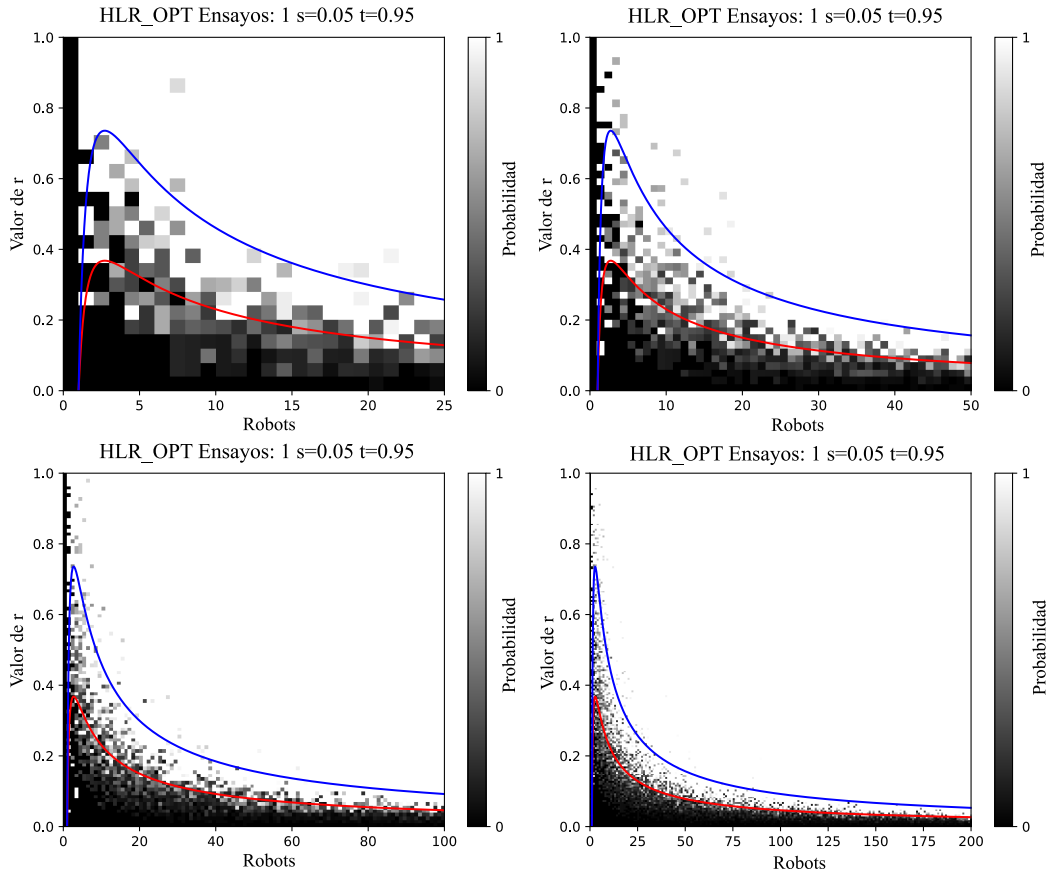


Figura 6.10: En cada uno de los gráficos, los pixeles de color blanco representan instancias que se resolvieron óptimamente al alcanzar su valor cs . Los pixeles de color negro representan instancias donde $opt \in [0, p_1]$. Para cualquier pixel en escala de gris $opt \in [p_j, p_{j+1}]$, para algún j . Solo en el primer caso se tiene la garantía de haber resuelto las instancias de manera óptima.

La libreta de Python **HLR_OPT.ipynb** para acceder al código y realizar las simulaciones: <https://colab.research.google.com/drive/1DEACVqQ5TNWpfd07-1T96zp3NVQVrQR?usp=sharing>

6.3. Metaheurísticas

Al inicio de este trabajo se contempló incluir dos metaheurísticas muy socorridas en el área de optimización discreta, a saber, el Recocido Simulado (RS) y el Algoritmo Genético (AG). La razón primordial se debe al hecho que las metaheurísticas son muy populares en la comunidad de optimización para atacar problemas que son NP-completos. De manera que se consideró oportuno tratar de entender la complejidad computacional de las metaheurísticas así como su aparente eficacia que las ha vuelto tan populares. En esta sección se pretende cuestionar ambos aspectos. En el apéndice A.6 se da una descripción somera de la implementación de RS y AG para resolver EDL.

La primera objeción es que las metaheurísticas no pueden garantizar resolver un problema de decisión u optimización de manera exacta en un tiempo finito. Algunas metaheurísticas, como RS o AG, poseen resultados analíticos que garantizan su convergencia *al óptimo* cuando el tiempo de cómputo tiende al infinito, véase [55, 33]. Pero existen metaheurísticas que no tienen ningún tipo de garantía de llegar a la solución óptima. Para una discusión detallada sobre la convergencia de las metaheurísticas véase [38]. En contraste, un algoritmo de fuerza bruta tiene la garantía de encontrar el óptimo en un tiempo de cómputo finito, si bien es cierto que puede ser exponencial. De este modo, si utilizamos el paradigma del peor de los casos, las metaheurísticas tienen una complejidad computacional que tiende al infinito.

En segundo lugar, una metaheurística engloba una búsqueda aleatoria. En [38] se plantea un algoritmo genérico que describe a todas las metaheurísticas. Dicho algoritmo tiene un paso fundamental donde la selección de soluciones es aleatoria. De hecho, la razón fundamental por la que algunas metaheurísticas tienen una garantía de convergencia asintótica al óptimo es porque son una *búsqueda aleatoria sofisticada*, por decirlo de algún modo. La búsqueda aleatoria simple tiene un resultado asintótico de encontrar la solución óptima [60]. Esta cualidad la heredan las metaheurísticas que poseen garantías de convergencia. Cabe señalar que Wolpert y Macready [69], demostraron en su teorema «No Free Lunch» que no existen algoritmos superiores a todos los demás en todos los problemas. De este modo, en un sentido amplio, una metaheurística no es mejor que una búsqueda aleatoria simple en promedio.

En tercer lugar, es bien sabido que una dificultad inherente a las metaheurísticas es lo que se conoce como *afinación* de los parámetros. A priori no existe manera de determinar los valores de los parámetros que gobiernan una metaheurística. En la práctica, esta afinación se lleva a cabo por ensayo y error, lo cual presenta ciertos inconvenientes.

Por último, las implementaciones de RS y AG que se realizaron en este trabajo no toman en cuenta la estructura del problema. A diferencia de BR, BRM y los algoritmos parciales que toman en cuenta la región activa para la selección de robots, RS y AG no lo hacen así, sino que muestrean el espacio de las $n!$ permutaciones de robots. En los primeros ensayos que se realizaron con RS y AG se encontró que los tiempos de ejecución crecían demasiado con el tamaño de la instancia. Asimismo, se encontró una

familia de instancias que se resuelven trivialmente $I = \left(\left\{ \left(\frac{i-1}{n}, \frac{1}{n} \right) \right\}_{i=1}^n, s=0, t=1 \right)$ que ambas metaheurísticas fallaban en resolver. El que los certificados fuesen permutaciones de $\{1, \dots, n\}$ obedece a que todos tengan el mismo tamaño. Esto resulta muy natural para trabajar con RS y AG al momento de definir vecinos en RS y al momento de realizar el cruce de cromosomas en AG. En la Figura 6.11 se muestra un comparativo de los tiempos de cómputo entre algunos de los algoritmos parciales presentados en el capítulo 5 con RS y AG.

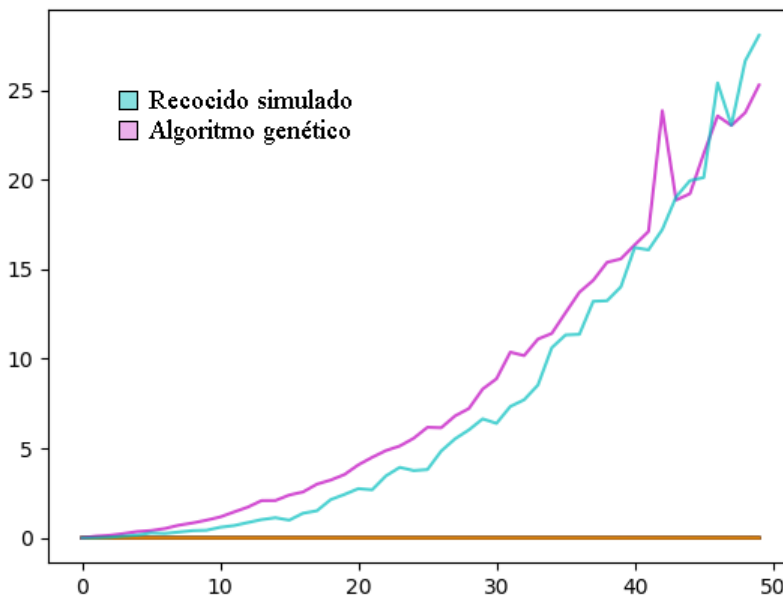


Figura 6.11: Comparativo de tiempos de cómputo entre algoritmos parciales y metaheurísticas. El eje horizontal representa el número de robots n . Verticalmente se encuentra el tiempo de cómputo en segundos. Para cada valor de n se generó una instancia aleatoria y se evaluó con los distintos algoritmos a comparar. Las gráficas de los tiempos de cómputo para los algoritmos parciales aparecen prácticamente como una línea horizontal en color ocre.

Debido a las objeciones presentadas anteriormente, se optó por no ampliar más sobre el tema de las metaheurísticas. A pesar de ello, se consideró la posibilidad de mejorar la

codificación de certificados tanto para RS como para AG. La idea es que los certificados correspondan a trayectorias de la raíz a una hoja en el árbol generado por BR. Dos vecinos se podrían definir como dos certificados de robots que compartan los primeros k robots para algún $k \in \mathbb{N}$. No queda claro como realizar el cruce de dos certificados de distinta longitud para AG, pero al parecer hay indicios en la literatura que apuntan a que esto es posible. Pero aún cuando esta mejora fuese realizable, se debe destacar que el algoritmo BR tendría todavía una ventaja sobre RS y AG. En principio, tanto RS como AG muestrean el espacio de posibles soluciones con reemplazo. En contraste, se puede visualizar a BR como una búsqueda aleatoria simple sin reemplazo. De este modo, el algoritmo BR muestrea el espacio de posibles soluciones por completo y una única vez. Esto se logra al explorar el árbol de posibles subsucesiones de robots a manera de búsqueda primero en profundidad. En cambio, RS y AG pueden mostrar soluciones de manera repetida con la garantía de explorar todo el árbol de BR cuando el tiempo de ejecución tiende al infinito. Así que aún en este escenario, el algoritmo BR sería el ganador indiscutible.

Pero, ¿cuál es la bondad del muestreo aleatorio que incorporan BR, BRM y las metaheurísticas? En el apéndice A.7 se presenta un modelo probabilístico para exhibir la bondad que presenta el muestreo aleatorio simple en términos del valor esperado de muestreos que se deben realizar para encontrar una solución de calidad aceptable. La siguiente sección aplica el modelo antes mencionado al caso específico de EDL. De este modo, se pretende dar un vistazo a la respuesta de ¿por qué las metaheurísticas parecen ser tan eficaces?

6.4. Muestreo aleatorio en el árbol generado por BR

En el apéndice A.7 se plantea un modelo para visualizar el muestreo aleatorio para un problema de optimización combinatoria o de decisión. Este modelo se puede adaptar y aplicar a EDL. La urna será el árbol BR. Las bolas blancas serán las hojas afirmativas. Las bolas negras serán las hojas negativas. Muestrear el árbol BR significa tomar una trayectoria aleatoria desde la raíz a alguna de sus hojas. Esto se logra con el algoritmo RS. Como se puede ver en el apéndice A.7, el número esperado de muestras antes de obtener una hoja afirmativa (en el muestreo con reemplazo) es $\frac{\# \text{ hojas negativas}}{\# \text{ hojas afirmativas}}$. De este modo, si se realizan k muestreos para encontrar una hoja afirmativa, tenemos que

$$k - 1 \approx \frac{h. n.}{h. a.} \implies h. a. (k - 1) \approx h. n.$$

Podemos además estimar el porcentaje aproximado de hojas afirmativas en todo el árbol BR

$$\%h.a. = \frac{h.a.}{h.n. + h.a.} \approx \frac{h.a.}{h. a. (k - 1) + h.a.} = \frac{1}{k - 1 + 1} = \frac{1}{k}.$$

La Figura 6.12 muestra un grupo de simulaciones que se realizaron en el espacio de parámetros (n, r) . En esta ocasión, se generó una instancia aleatoria para cada pareja (n, r) . Acto seguido, se aplicó el algoritmo RS para muestrear con reemplazo trayectorias en el árbol BR. Se fijó el número máximo de muestreos en 100, es decir, el algoritmo RS realiza hasta 100 intentos para tratar de resolver la instancia. El número de intentos en que se resuelve la instancia es k , así que se guarda el valor $\frac{1}{k}$ en una matriz de $n \times n$. Si no se resuelve la instancia en 100 intentos, se guarda el valor 0 en la matriz. Estos valores se grafican en una rejilla, donde al 0 le corresponde el color negro y al 1 el color blanco. Este número entre 0 y 1 representa la estimación porcentual de hojas afirmativas en el árbol BR de la instancia correspondiente.

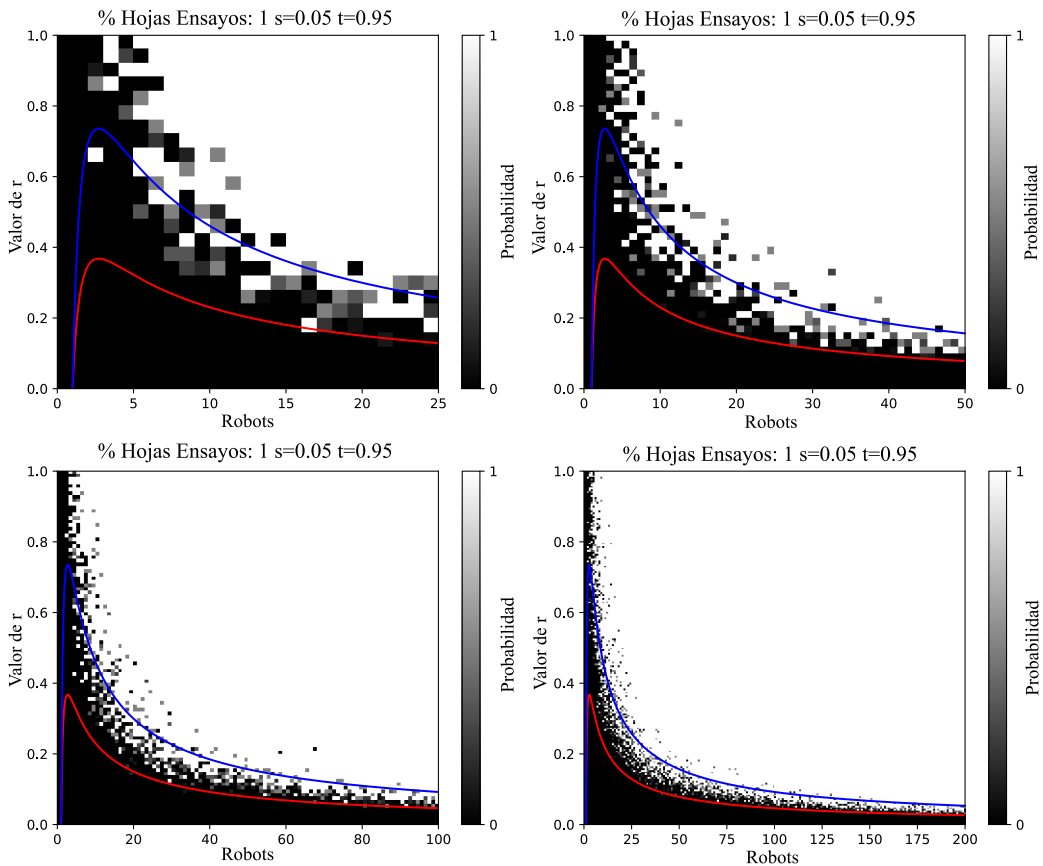


Figura 6.12: En estos gráficos se aprecia %h.a. al utilizar el algoritmo RS.

Las observaciones de esta sección conducen a la conclusión de que, tanto en la *zona blanca* como en la *zona negra*, cualquier algoritmo tendría un buen desempeño para clasificar las instancias. Como se mencionó en la sección anterior, esto arroja luz sobre un buen desempeño de las metaheurísticas en este contexto. Dado que las metaheurísticas efectúan una buena cantidad de muestreos aleatorios en una sola corrida, no es de extrañar que su comportamiento sea similar al del algoritmo RS en el espacio de parámetros (n, r) .

Las instancias difíciles, de acuerdo a el análisis presentado en este trabajo, consisten en aquéllas cuyo árbol BR tiene algunas pocas hojas afirmativas. Por un lado, el algoritmo BR realiza muestreos sin reemplazo en el árbol BR, mientras que el algoritmo RS o alguna de las metaheurísticas mencionadas realizan muestreo con reemplazo. En el primer caso, ante una instancia difícil, el algoritmo BR podría tardar un tiempo exponencial al recorrer casi todas las ramas en busca de alguna de las pocas hojas afirmativas. El muestreo aleatorio con reemplazo tardaría aún más tiempo. De manera que, el aprovechar alguna cualidad estructural del problema redundaría en un mejor desempeño, como es el caso del algoritmo BRM que simplifica la búsqueda al contraer un conjunto de nodos a un solo nodo, cuando dicho conjunto de nodos representa una componente conexa de la gráfica de intervalos con valores $a_i = 0$. O bien, el algoritmo HLR que debido a ser una estrategia óptima para dos robots, en la práctica resultó ser tan bueno como BR pero con un tiempo de ejecución $O(n \log n)$. De este modo, dado que las metaheurísticas son algoritmos de propósito general, no explotan ninguna cualidad estructural del problema, por lo que no debe esperarse un desempeño mejor que el algoritmo RS en promedio.

La libreta de Python **RandomSample_EDL.ipynb** para acceder al código y realizar las simulaciones: <https://colab.research.google.com/drive/1spDv0LJYiJfrbHpB8Yst1Vusp=sharing>

6.5. Estrategia definitiva

Para concluir con el estudio presentado en este trabajo, se presenta a continuación la mejor estrategia para resolver EDL que resume los hallazgos importantes. Sea

$I = (\{(x_i, \rho_i)\}_{i=0}^n, s, t)$ una instancia de EDL.

1. De acuerdo al corolario 2.5 si $G_S(Q')$ es desconexa entonces la instancia no es soluble. Debido a que verificar la conexidad de una gráfica de intervalos se logra en tiempo $O(n \log n)$ y a que el corolario 2.5 es una garantía que la desconexidad implica insolubilidad, este es el primer paso en la estrategia para resolver I . Si $G_S(Q')$ es desconexa, se termina la búsqueda. Si $G_S(Q')$ es conexa se procede al paso 2.
2. Si se verifica en el primer paso que $G_S(Q')$ es conexa se procede a utilizar el algoritmo HLR. La elección de HLR es por su eficiencia con respecto a BRM, y

eficacia con respecto a los demás algoritmos parciales. Si HLR resuelve afirmativamente I se termina la búsqueda. Si HLR clasifica a I como insoluble se procede al paso 3. Esto es debido a que HLR no es un algoritmo exacto y existe la posibilidad de que dé un falso negativo.

- 3. Si I fue clasificada como negativa por HLR, se utiliza BRM para descartar que se trate de un falso negativo.

Esta estrategia se encuentra en el diagrama de la Figura 6.13.

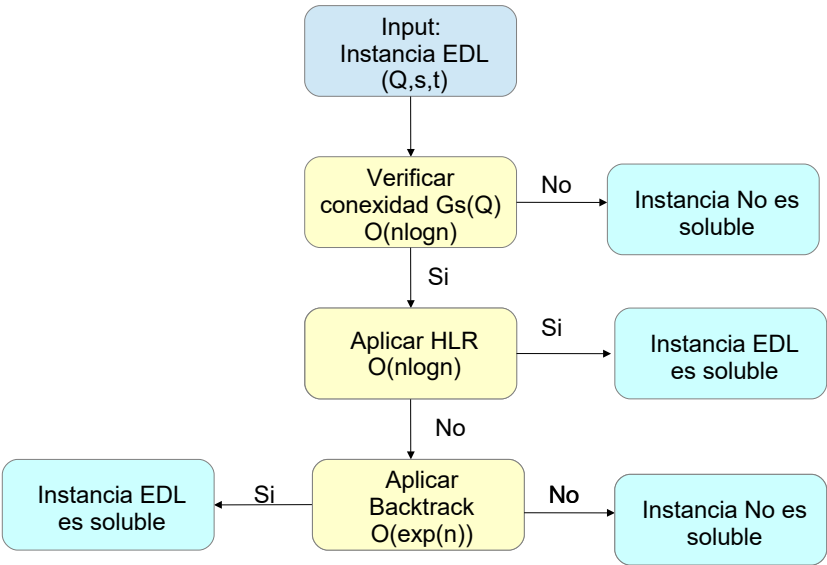


Figura 6.13: Estrategia definitiva para resolver EDL.

La estrategia anterior representa la suma de los hallazgos presentados en este trabajo en torno al problema EDL. La idea es utilizar algoritmos rápidos para resolver instancias que se encuentren en zonas en el espacio de parámetros donde se sabe a priori la

alta probabilidad de ser solubles o no. A esto se hizo referencia en la introducción. La zona por encima del umbral de EDLA representa el verano, con árboles de BR llenos de hojas afirmativas. La zona por debajo del umbral de Scheinerman representa el invierno, con árboles de BR con una alta probabilidad de no tener hojas afirmativas. Cerca del conjeturado umbral de transición de fase es difícil resolver las instancias, por lo que no queda más remedio que utilizar el algoritmo exacto BRM. Esta zona, acotada por los dos umbrales mencionados, representa el otoño y la primavera, donde se encuentran las instancias difíciles. Pero como ya se ha mencionado anteriormente, el algoritmo BRM incorpora algunas ventajas. En primer lugar, es equivalente a un muestreo aleatorio sin reemplazo. Esto lo pone en una mejor posición que las metaheurísticas o el muestreo aleatorio simple con reemplazo. En segundo lugar, al incorporar como subrutina el algoritmo polinomial para resolver EDLA, esto reduce el tamaño del árbol BR. Por último, la Figura 6.14 muestra una simulación, de manera similar a las simulaciones presentadas a lo largo de este trabajo, donde ahora lo que se grafica es el algoritmo utilizado para resolver la instancia representada por cada pareja (n, r) en la porción discretizada del espacio de parámetros. Aquellas instancias resueltas en el paso 1, se colorean de negro (disconexidad). Aquellas instancias resueltas en el paso 2 se colorean de blanco (HLR). Aquellas instancias resueltas en el paso 3 se colorean de gris (BRM).

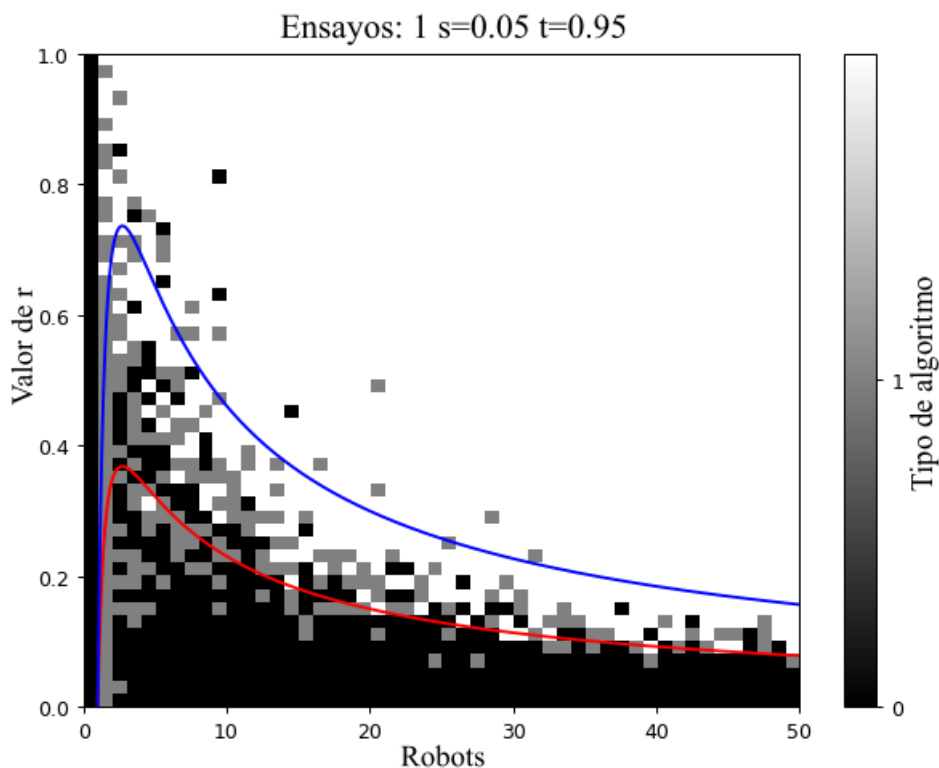


Figura 6.14: En esta simulación, los pixeles blancos representan instancias resueltas con HLR. Los pixeles grises representan instancias resueltas con BRM y los pixeles negros representan instancias resueltas de acuerdo al corolario 2.5, es decir, si $G_S(Q')$ es desconexa.

Como se puede apreciar, la mayoría de las instancias son resueltas eficientemente y de manera exacta, puesto que HLR no genera falsos positivos, ni verificar la desconexidad genera falsos negativos. Los casos restantes son resueltos por BRM.

La libreta de Python **Best_Strategy.ipynb** para acceder al código y realizar las simulaciones: <https://colab.research.google.com/drive/132sigrroCIEA0u0HuLu4RlBsgPV0LaLusp=sharing>

Parte I

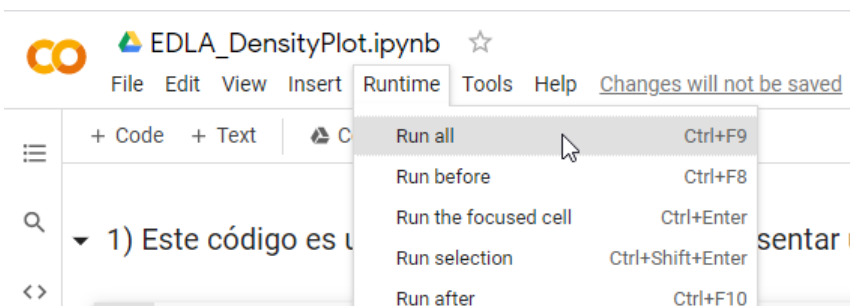
Apéndice

Apéndice A

Apéndice

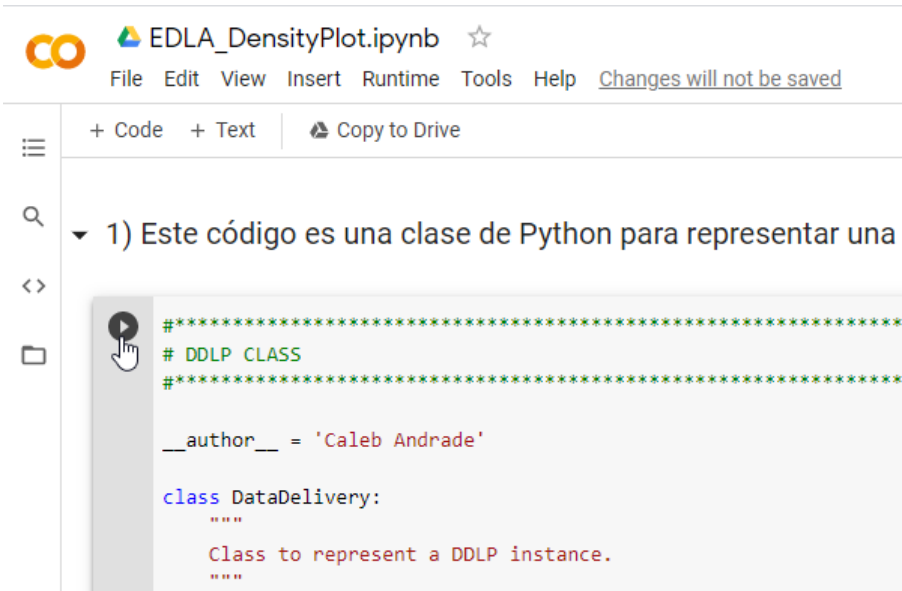
A.1. Google Colab

Es preciso contar con una cuenta de Google para tener acceso al código de la implementación de los algoritmos. La ventaja de esta plataforma es que no es necesario instalar ningún software. El código se compila y ejecuta en un servidor de Google, por lo que solo es necesario contar con acceso a internet y un explorador actualizado. Los enlaces a las carpetas de Python se encuentran al final de la sección donde se exponen los algoritmos. Hay dos maneras de ejecutar el código. La primera es seleccionar la opción *run all* con la cual todos los bloques de código se ejecutan automáticamente y el resultado se observa al final de los mismos. Esto se logra al apretar Ctrl + F9 o bien al seleccionar la opción del menú correspondiente como lo muestra la siguiente Figura.



La otra opción es ejecutar los bloques de código uno por uno. Esto tendría el objetivo de ver el funcionamiento de cada bloque de código y se logra al apretar el botón *play* a la izquierda de cada bloque de código. Es importante señalar que si se omite la ejecución de algún bloque de código, es muy probable que haya fallas en la simulación. Por otro lado, ya que es una plataforma que ofrece poder de cómputo gratuito, en ocasiones los

servidores se encuentran ocupados y puede ser que tome algo de tiempo hasta que se conecte a una máquina virtual disponible para la ejecución del código.



The screenshot shows a Jupyter Notebook titled "EDLA_DensityPlot.ipynb". The interface includes a top menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". Below the menu bar, there are tabs for "+ Code", "+ Text", and "Copy to Drive". The notebook content is displayed in a cell, showing a Python class definition. The code is as follows:

```
#####  
# DDLP CLASS  
#####  
  
__author__ = 'Caleb Andrade'  
  
class DataDelivery:  
    """  
    Class to represent a DDLP instance.  
    """
```

A.2. Breve reseña histórica de la computación

La palabra *computación* proviene del latín *computare*, que significa calcular. Desde tiempos antiguos el hombre se ha visto en la necesidad de realizar cálculos. Por ejemplo, cuando la civilización Egipcia llegó a un estatus de estabilidad bajo el mando de un solo gobernante, el faraón, entonces comenzó a evolucionar un extenso y poderoso sistema para la administración de los censos, la recolección de impuestos, la hacienda real y los recursos del ejército. Registros históricos muestran que la civilización Egipcia tenía un sistema numérico plenamente desarrollado hacia 3500 a.C. que les permitía contar y realizar operaciones aritméticas básicas para el manejo de su sistema administrativo [12]. Otro ejemplo se encuentra en la civilización Babilónica. El registro histórico, en forma de tablillas de barro con escritura cuneiforme, muestra que los babilonios tenían un conocimiento matemático importante hacia el 2100 a.C. Desarrollaron sus matemáticas en base al sistema sexagesimal, y demostraron un alto nivel de habilidad computacional para realizar distintas tareas como el cálculo del interés simple y compuesto, cálculo de áreas y volúmenes, resolución de ecuaciones cuadráticas y cúbicas, entre otros [29]. Conforme las demandas computacionales y la tecnología avanzaron a lo largo de la historia de la humanidad, se han desarrollado mecanismos cada vez más ingeniosos para realizar los cálculos, primordialmente a partir del siglo XVII, culminando con la era de la computadora moderna a partir de mediados del siglo XX. A continuación, una breve

cronología de algunos de los momentos importantes en el desarrollo de la computación, desde el siglo XVII y hasta la época actual. Como se ha mencionado en el capítulo 1, la unidad para medir el desempeño de los equipos de cómputo es el FLOPS (Floating point Operations Per Second).

- En 1617, John Napier desarrolló un sistema para realizar operaciones aritméticas mediante la manipulación de barras, que llamó *huesos*, descrita en su obra *Rabdo-logia* [52].
- En 1623, la primera calculadora mecánica fue diseñada por Wilhem Schickard en Alemania [37].
- En 1632, la primera regla deslizante fue inventada por William Oughtred. Esta herramienta tuvo una larga vida, hasta 1970 cuando las calculadoras portátiles se volvieron populares [37].
- En 1642, Blaise Pascal inventa la Pascalina, la primera calculadora que funcionaba a base de ruedas y engranajes [37].
- En 1673, Gottfried Leibniz inventa la primera calculadora de propósito general, que utilizaba un cilindro de dientes, en lugar de engranes [37].
- En 1822, Charles Babbage construyó el primer prototipo de su Máquina de Diferencias, diseñada para trabajar con polinomios de grado seis [24].
- En 1837, Charles Babbage diseñó su Máquina Analítica, considerado el primer computador moderno de propósito general [11].
- En 1843, Ada Augusta Lovelace escribió el primer programa de computadora para la Máquina Analítica de Babbage [31].
- En 1878, Ramón Vereá, inventó una calculadora capaz de sumar, restar, multiplicar y dividir números de nueve cifras [64].
- En 1890, Herman Hollerith utilizó su Máquina Tabuladora para el censo de los EE.UU., utilizando tarjetas perforadas para introducir a la máquina la información del censo [37].
- En 1893, Otto Steiger desarrolló la primera máquina exitosa de multiplicación automática [65].
- En 1920, Leonardo Torres Quevedo construye su aritmómetro electromecánico, primera calculadora automática [65].
- En 1936, Alan Turing describe la Máquina de Turing, la cual formaliza el concepto de algoritmo [63].

- En 1943, en Inglaterra se construyeron los ordenadores Colossus, con el objetivo de descifrar las comunicaciones de los alemanes durante la Segunda guerra mundial [20].
- En 1946, en la Universidad de Pensilvania se construye la ENIAC, la primera computadora electrónica de propósito general. ENIAC podía resolver 5000 sumas o 300 multiplicaciones en 1 segundo [37].
- En 1964, aparece la primera supercomputadora comercialmente disponible, el CDC 6600, capaz de operar a 1 MFLOPS [20, 66].
- En 1970, Texas Instruments introduce al mercado la primera calculadora electrónica portátil *pocketronic* [39].
- En 1973, Xerox PARC desarrolló la primera Computadora de Escritorio con interfaz gráfica [65].
- En 1976, la supercomputadora Cray-1 fue puesta en marcha en el Laboratorio Nacional de Los Álamos, capaz de operar a 133 MFLOPS [20, 53].
- En 1977, Apple presenta el primer computador personal que se vende a gran escala, el Apple II [65].
- En 1985, fue creada la supercomputadora Cray-2, capaz de operar a 1.9 GFLOPS [68].
- En 1996, la supercomputadora ASCI Red rompe la barrera del Tera-FLOPS con 1.34 TFLOPS [20, 67].
- En 2018, Summit de IBM se convirtió en la supercomputadora más rápida del planeta, con una capacidad actual de 200 PFLOPS [61].
- En 2020, Hewlett-Packard anunció que su nueva supercomputadora El Capitan alcanzará una capacidad pico de 2 EFLOPS en el 2023 [40].

A.3. Ejemplo de una MTD para resolver un problema de decisión.

En esta sección se examinará un ejemplo de Garey & Johnson de cómo una MTD resuelve un problema de decisión. El *valor de entrada* para la MTD es una cadena $x \in \Sigma^*$ con respecto a un alfabeto Σ . La cadena x se coloca en los cuadros del 1 al $|x|$, siempre un símbolo por cuadro. Todos los demás cuadros tienen por defecto el símbolo blanco. El programa comienza su ejecución en el estado q_0 , con la cabeza leyendo el cuadro 1. El cómputo procede ahora paso por paso. Si el estado actual es q_Y (sí) o q_N (no), el

cálculo ha terminado. De otro modo, el estado actual es $q \in Q - \{q_Y, q_N\}$, y el símbolo leído es algún s , con lo cual se define $\delta(q, s)$. Supóngase que $\delta(q, s) = (q', s', \Delta)$. La cabeza borra s , escribe s' en su lugar (si el programa requiere no cambiar s , se vuelve a escribir s), y se mueve un cuadro a la izquierda si $\Delta = -1$ o un cuadro hacia la derecha si $\Delta = +1$. A continuación, la definición del problema de divisibilidad por cuatro.

Criterio de Divisibilidad del Cuatro

- Parámetros: Un número positivo entero N .
- Pregunta: ¿Existe un número positivo entero m tal que $N = 4m$?

El entero N está dado en su representación binaria. Ahora bien, un número entero positivo es divisible por cuatro si el número formado por sus dos últimas cifras es divisible por cuatro. De lo anterior, se deduce que un número entero positivo es divisible por cuatro si y sólo si sus dos últimos dígitos en representación binaria son 0. El siguiente MTD, que llamaremos M , es un algoritmo para resolver el problema propuesto.

$$\Gamma = \{0, 1, b\}, \Sigma = \{0, 1\}$$

$$Q = \{q_0, q_1, q_2, q_Y, q_N\}$$

$$\delta(q, s) :$$

q	0	1	b
q_0	$(q_0, 0, +1)$	$(q_0, 1, +1)$	$(q_1, b, -1)$
q_1	$(q_2, b, -1)$	$(q_N, b, -1)$	$(q_N, b, -1)$
q_2	$(q_Y, b, -1)$	$(q_N, b, -1)$	$(q_N, b, -1)$

Figura A.1: Ejemplo de una MTD, $M = (\Gamma, Q, \delta)$.

En este ejemplo, la función de transición δ está dada en forma tabular. Los pasos del algoritmo al resolver una instancia $x = 10100$ se encuentran en la Figura A.1. Cada renglón representa una fracción de la cinta, y cada celda representa un cuadro. Los símbolos en negritas indican el símbolo que está leyendo la cabeza de la MTD.

El algoritmo se detiene después de ocho pasos, en el estado q_Y , lo cual significa que la respuesta para la instancia $x = 10100$ es «sí».

En este ejemplo, las soluciones afirmativas están dadas por el siguiente conjunto de cadenas

$$\{x \in \{0, 1\}^* : \text{los dos últimos dígitos de } x \text{ son ambos } 0\}.$$

$q_0 :$...	b	1	0	1	0	0	b	...
$q_0 :$...	b	1	0	1	0	0	b	...
$q_0 :$...	b	1	0	1	0	0	b	...
$q_0 :$...	b	1	0	1	0	0	b	...
$q_0 :$...	b	1	0	1	0	0	b	...
$q_0 :$...	b	1	0	1	0	0	b	...
$q_1 :$...	b	1	0	1	0	0	b	...
$q_2 :$...	b	1	0	1	0	b	b	...
$q_Y :$...	b	1	0	1	b	b	b	...

Figura A.2: Ejecución de una MTD para decidir si un número entero positivo es divisible por cuatro o no.

A.4. Ejemplo de una reducción polinomial

A continuación se ejemplifica cómo reducir polinomialmente Circuito Hamiltoniano a Agente Viajero.

Para una gráfica $G = (V, E)$ cuyo conjunto de vértices es V y conjunto de aristas E , un *circuito simple* en G es una secuencia de vértices distintos $\langle v_1, v_2, \dots, v_k \rangle$, tal que $(v_i, v_{i+1}) \in E$ para $1 \leq i \leq k$ y $(v_k, v_1) \in E$. Un *circuito Hamiltoniano* en G es un circuito simple que incluye todos los vértices de G . Se define a continuación el problema de Circuito Hamiltoniano:

Circuito Hamiltoniano

- Parámetros: Una gráfica $G = (V, E)$.
- Pregunta: ¿Contiene G un circuito Hamiltoniano?

Debería resultar clara la semejanza que existe entre este problema y el problema del Agente Viajero. A continuación se demuestra cómo reducir Circuito Hamiltoniano (CH) a Agente Viajero (AV), es decir, cada instancia de CH será transformada en una instancia correspondiente de AV mediante una función f que satisface las dos condiciones dadas en la definición 1.15.

Supóngase que $G = (V, E)$, con $|V| = n$, es una instancia de CH. La instancia correspondiente de AV tiene como conjunto de ciudades C el conjunto de vértices V de CH. Para cualesquiera dos ciudades $v_i, v_j \in C$ definimos $d(v_i, v_j) = 1$ si $(v_i, v_j) \in E$ y $d(v_i, v_j) = 2$ en otro caso. El valor de la cota B se define como n . Es claro ver que la transformación $f : CH \rightarrow AV$ es polinomial. Para cada una de las $\frac{n(n-1)}{2}$ distancias $d(v_i, v_j)$ que se deben especificar, es necesario verificar en G si $(v_i, v_j) \in E$. De esta manera se satisface la primer condición para f . Para demostrar la segunda condición, es

preciso demostrar que G tiene un circuito Hamiltoniano si y sólo si existe un recorrido en $f(G)$ que pasa por todas las ciudades una única vez, regresando a la ciudad de origen, y cuya longitud total es a lo más n . Supongamos primero que existe un recorrido en $f(G)$ que pasa por cada ciudad y regresa a la ciudad de origen, con longitud total no mayor a n . Claramente se tiene un circuito simple. Además, como contiene n aristas y su longitud total es a lo más n , necesariamente las n aristas tienen longitud 1, y por lo tanto dichas aristas pertenecen a E , por lo cual G tiene un circuito Hamiltoniano. Supongamos ahora que G tiene un circuito Hamiltoniano. Claramente dicho circuito es un recorrido en $f(G)$ y además, como las aristas del circuito Hamiltoniano pertenecen a E , las distancias entre cada ciudad son 1, por lo que la longitud del recorrido total es n .

De esta manera, $CH \propto AV$. La relevancia del lema 1.1 para problemas de decisión ahora puede ser ilustrada. Básicamente, se puede deducir que si AV se puede resolver mediante un algoritmo polinomial, entonces también CH se puede resolver eficientemente, y si CH es intratable, entonces AV también es intratable. Dicho de otro modo, si $\Pi_1 \propto \Pi_2$ entonces Π_2 es al menos tan difícil de resolver como Π_1 .

A.5. Dos problemas con buena caracterización

A continuación, dos problemas con buena caracterización. El primero está en P mientras que del segundo no se sabe.

Sistema de Ecuaciones Lineales (SEL)

- Parámetros: Una matriz $A \in \mathbb{Q}^{m \times n}$ y un vector $b \in \mathbb{Q}^m$ (con todos los números en representación binaria).
- Pregunta: ¿Existe un vector $x \in \mathbb{Q}^n$ tal que $Ax = b$?

Se puede demostrar que este problema se resuelve en tiempo polinomial con eliminación Gaussiana, al tomar en cuenta la representación binaria de los números para operaciones aritméticas. Ahora bien, se dará un bosquejo de la demostración $SEL \in NP \cap coNP$.

Es claro que $SEL \in NP$, pues dados valores A, b y x se puede verificar en tiempo polinomial (en el tamaño de la representación binaria y la matriz A) que $Ax = b$, al considerar que la suma y la multiplicación son operaciones de tiempo polinomial en el tamaño de las cadenas de bits. Ahora, para demostrar que $SEL \in coNP$, se utiliza el siguiente teorema del álgebra lineal:

Teorema A.1. *Para cada sistema de ecuaciones lineales $Ax = b$, con $A \in \mathbb{Q}^{m \times n}$ y $b \in \mathbb{Q}^m$, o bien existe $x \in \mathbb{Q}^n$ con $Ax = b$ o bien existe $y \in \mathbb{Q}^m$ tal que $y^\top A = 0$ y $y^\top b = 1$.*

Ahora bien, puesto que $y^\top (Ax - b) = (y^\top A)x - y^\top b$, si $y^\top A = 0$ y $y^\top b = 1$ se implica que $y^\top (Ax - b) = 1$, es decir, $Ax - b \neq 0$. En este caso tendríamos que el sistema $Ax = b$

no tiene solución. Se puede entonces resolver el sistema lineal $\{y^T A = 0, y^T b = 1\}$ en tiempo polinomial, por lo que $SEL \in coNP$, lo cual termina el bosquejo de la demostración.

El siguiente ejemplo es de un problema de decisión con buena caracterización del cual depende en gran medida la seguridad de los protocolos criptográficos actuales.

Factorización

- Parámetros: Tres números $x, a, b \in \mathbb{N}$ codificados en su representación binaria.
- Pregunta: ¿Existe un factor primo p en el intervalo $[a, b]$ tal que p divide a x ?

El problema está claramente en NP, pues dado un factor p , se puede verificar en tiempo polinomial que es primo con la prueba AKS. También se puede verificar en tiempo polinomial que p divide a x y que $p \in [a, b]$.

Este problema también está en coNP, pues la factorización en primos de x es un certificado para instancias negativas. Dados los números p_1, \dots, p_k , se puede verificar con el test AKS que en efecto son una factorización en primos de x , y que ninguno de estos primos esté en el intervalo $[a, b]$.

A.6. Metaheurísticas para resolver EDL

Algoritmo Genético

Una de las metaheurísticas más utilizadas basadas en poblaciones es el Algoritmo Genético (AG). Fue introducido por primera vez por John Holland en 1975 [42]. El Algoritmo Genético tiene como objetivo hacer que una computadora simule los supuestos de la teoría de la selección natural propuesta por Charles Darwin. La idea general es comenzar con una población de posibles certificados al problema en cuestión. Sigue una etapa de cruce, el análogo de la reproducción, que tiene lugar entre los cromosomas (certificados) más aptos. Después del cruce emerge una nueva población de descendientes de los más aptos. Además, con cierta probabilidad se produce una mutación aleatoria en los cromosomas. Esta mutación consiste en intercambiar el orden de dos robots adyacentes en el certificado. Este proceso se repite a lo largo de un número determinado de generaciones (bucles) o hasta que se cumple un criterio de paro. El esquema general del algoritmo se describe en el pseudocódigo del algoritmo A.1.

Un paso importante en AG es la representación del dominio de la variable del problema como un cromosoma. A esto se le conoce como codificación. La forma habitual de codificar certificados es como representaciones de cadenas de bits. Por simplicidad, una forma muy natural de codificar certificados de EDL es representar un cromosoma con

Algoritmo A.1 AG

-
- 1: *Inicializar* población con certificados aleatorios;
 - 2: *Evaluar* el valor objetivo de cada solución en la población;
 - 3: **mientras** $i < \#generaciones$ **hacer**:
 - 4: *Seleccionar* mejores certificados;
 - 5: *Cruce* entre pares de certificados;
 - 6: *Mutar* certificados descendientes;
 - 7: *Evaluar* valores objetivos de las certificados descendientes;
 - 8: *Seleccionar* mejores certificados para la próxima generación;
 - 9: $i++$
 - 10: **terminar**;
-

una cadena de números enteros. Dados n robots, un cromosoma será una permutación de los números $1, \dots, n$, que representa el orden en el que se moverán los robots.

Evaluar el valor objetivo se refiere a mover los robots del certificado, de acuerdo a lo estipulado en la sección 2.3, y registrar el valor d hasta donde se movieron los datos. Si un robot en el certificado es incapaz de mover los datos, se omite y se procede con el siguiente robot del certificado. Después se realiza una selección aleatoria tipo ruleta, en la que cada certificado tiene una rodaja de rueda proporcional a su valor objetivo d . La ruleta se gira N veces para seleccionar N certificados de forma independiente. Estos certificados participarán en la fase de cruce.

El operador de cruce más básico entre dos cadenas de símbolos suele consistir en seleccionar un punto de empalme, dividir ambos certificados en dicho punto e intercambiar las últimas mitades de cada certificado. Así se obtiene como resultado dos descendientes. En el caso de EDL, este operador de cruce podría generar certificados no válidos con repetición de robots. Para evitar este problema se adoptó el siguiente operador de cruce para dos certificados c_1 y c_2 :

1. Seleccione al azar dos puntos de empalme entre las posiciones 1 a $n - 1$, p_1 y p_2 , $p_1 < p_2$
2. De c_1 extraiga la subcadena $c_1[p_1 : p_2]$ (símbolos de c_1 de la posición p_1 a p_2)
3. Para cada símbolo en la subcadena $c_1[p_1 : p_2]$, elimine dicho símbolo en c_2 .
4. Inserte la subcadena $c_1[p_1 : p_2]$ en la cadena modificada c_2 en la posición p_1 .

Un par de certificados del conjunto seleccionado se cruzarán con probabilidad P_c , de lo contrario, la descendencia serán dos clones idénticos de los certificados originales.

Con el fin de determinar la mejor configuración de parámetros para AG, y debido a la ausencia de información previa que sugiera valores de parámetros, se llevaron a cabo una serie de experimentos sobre un conjunto de instancias con diferentes configuraciones del espacio de parámetros. No hay unanimidad completa en la selección de los parámetros de AG, pero hay algunas pautas para el rango de valores que estos parámetros satisfacen, véase [60]. Estos son $20 \leq N \leq 100$, $0.60 \leq P_c \leq 0.95$ y $0.001 \leq P_m \leq 0.01$. En este

caso, N representa el tamaño de la población, P_c es la probabilidad de cruce y P_m es la probabilidad de mutación.

Para un conjunto de 100 instancias fijas de EDL de diferentes tamaños, se muestreo uniformemente 50 vectores en el espacio de parámetros dado por las restricciones del párrafo anterior. Para cada configuración, se ejecutaron 100 generaciones y se contabilizó el número de instancias resueltas afirmativamente, así como el tiempo de ejecución. El criterio principal para seleccionar la mejor configuración fue el número de instancias resueltas afirmativamente. El tiempo de ejecución se utilizó para romper empates.

Recocido Simulado

En 1983 apareció el artículo "Optimización por recocido simulado" [44] en el que sus autores, Kirkpatrick, Gelatt y Vecchi, exponen una técnica de optimización heurística, denominada recocido simulado (RS). Esta técnica consiste en implementar una búsqueda estocástica en analogía a una proceso de la mecánica estadística llamado recocido. En este proceso, se busca que un sólido fundido alcance su estado de mínima energía. Esto se logra al permitir que sus moléculas se reorganicen mientras el material recibe un baño termal que mantiene su temperatura cerca del punto de solidificación por un período prolongado. Dicha circunstancia permite que las moléculas tengan cierta holgura para moverse. Luego, la temperatura disminuye lenta y progresivamente. En esta analogía, RS imita el proceso de recocido al considerar la función objetivo como el estado de energía que se debe minimizar. La temperatura es un parámetro controlable que disminuye gradualmente hasta cero.

El algoritmo comienza al generarse un certificado inicial c_1 y establecer los valores iniciales que gobiernan el comportamiento del algoritmo, como la temperatura t_k . Después de cada iteración, se selecciona aleatoriamente un nuevo certificado c_2 en la vecindad del certificado anterior. Un vecino del certificado se obtiene al permutar la posición de cualesquiera dos robots en el certificado original. Si el nuevo certificado c_2 tiene un mejor valor objetivo $d(c_2)$, se selecciona y reemplaza al certificado anterior. En caso contrario, se selecciona con cierta probabilidad que depende de la temperatura t_k y de la diferencia de los valores objetivo $d(c_1) - d(c_2)$. Lo anterior se resume en la siguiente regla

$$P(\text{aceptar } c_2) = \begin{cases} 1 & \text{si } d(c_2) \geq d(c_1) \\ \frac{1}{\exp\left(\frac{d(c_1) - d(c_2)}{t_k}\right)} & \text{si } d(c_2) < d(c_1) \end{cases}.$$

A medida que se ejecuta el algoritmo, la temperatura disminuye progresivamente mediante alguna regla. La regla más común es aplicar un factor de enfriamiento geométrico $\gamma \approx 0.9$ tal que $t_{k+1} = \gamma \cdot t_k$. El enfriamiento se aplica cuando se alcanza un estado de equilibrio dinámico. El equilibrio dinámico se refiere a que no existe mejora significativa en la búsqueda de nuevos certificados a lo largo de la cadena de Markov (de longitud

Algoritmo A.2 RS

```

1: Inicializar parámetros  $T, \lambda, \varepsilon, \gamma$ , mejor_certificado;
2: Seleccionar un certificado aleatorio  $s$ ;
3: mientras  $T > \varepsilon$  hacer:
4:   mientras no se alcance equilibrio dinámico en  $\lambda$  iteraciones hacer:
5:     Seleccionar aleatoriamente un certificado vecino  $s'$ ;
6:     si  $d(s') \geq d(s)$  entonces:
7:        $s \leftarrow s'$ 
8:     en otro caso:
9:        $s \leftarrow s'$  con probabilidad  $\exp\left(-\frac{d(s)-d(s')}{T}\right)$ ;
10:    si  $d(s) > d(\text{mejor\_certificado})$  entonces:
11:      mejor_certificado  $\leftarrow s$ 
12:   terminar;
13:    $T = \varphi \cdot T$ ;
14: terminar;
15: devolver mejor_certificado;

```

λ) que se genera con cada uno de los estados del sistema en cada iteración. Cuando la temperatura es menor a un $\varepsilon > 0$ el algoritmo se detiene.

La idea detrás de este método es proporcionar un mecanismo para que la búsqueda local escape de los mínimos locales. Siempre que la temperatura sea positiva, existe una probabilidad positiva de que se acepte un certificado que empeora el valor objetivo. El algoritmo se describe en el pseudocódigo del algoritmo A.2.

A.7. Modelo de las urnas

Dado un problema P de decisión (u optimización combinatoria), el espacio de soluciones se puede visualizar como una urna con bolas blancas y negras. Las bolas blancas representan el conjunto de soluciones afirmativas (u óptimas). Las bolas negras representan el resto de las soluciones. Si se muestrea aleatoriamente este espacio, ¿cuál es el número esperado de bolas negras que se deben muestrear antes de sacar una bola blanca? Resulta ser que si se muestrea el espacio de soluciones un número razonable de veces se pueden obtener buenos resultados. Por otro lado, si se tiene un conocimiento previo para restringir el espacio de búsqueda, esto sería como tener una urna con menos bolas. En consecuencia, el número de muestreos se reduce. El modelo se describe a continuación.

Se tienen b bolas blancas y n bolas negras en una urna y se realizan los siguientes experimentos

1. Extraer bolas sin reemplazo hasta sacar una bola blanca.

2. Extraer bolas con reemplazo hasta sacar una bola blanca.

Caso: $b = 1$ y n arbitraria

Sin reemplazo

Sea la variable aleatoria χ el número de bolas negras que son muestreadas antes de que salga la bola blanca. El muestreo es sin reemplazo. Entonces

$$E[\chi] = \sum_{k=0}^n kP(\chi = k).$$

Si se desea calcular $E[\chi]$, primeramente se calculan las probabilidades $P(\chi = k)$ considerando que el muestreo es sin reemplazo:

$$\begin{aligned} P(\chi = 0) &= \frac{1}{n+1} \\ P(\chi = 1) &= \frac{n}{n+1} \cdot \frac{1}{n} = \frac{1}{n+1} \\ P(\chi = 2) &= \frac{n}{n+1} \cdot \frac{n-1}{n} \cdot \frac{1}{n-1} = \frac{1}{n+1} \\ &\vdots \\ P(\chi = k) &= \frac{1}{n+1}. \end{aligned}$$

Así pues

$$E[\chi] = \sum_{k=0}^n \frac{k}{n+1} = \frac{1}{n+1} \sum_{k=0}^n k = \frac{n(n+1)}{2(n+1)} = \frac{n}{2}.$$

Con reemplazo

Sea la variable aleatoria χ el número de bolas negras que son muestreadas antes de que salga la bola blanca. El muestreo es con reemplazo. Entonces

$$E[\chi] = \sum_{k=0}^{\infty} kP(\chi = k).$$

Primeramente se calculan las probabilidades $P(\chi = k)$ considerando que el muestreo es con reemplazo:

$$\begin{aligned}
P(\chi = 0) &= \frac{1}{n+1} \\
P(\chi = 1) &= \frac{n}{n+1} \cdot \frac{1}{n+1} \\
P(\chi = 2) &= \left(\frac{n}{n+1}\right)^2 \frac{1}{n+1} \\
&\vdots \\
P(\chi = k) &= \left(\frac{n}{n+1}\right)^k \frac{1}{n+1}.
\end{aligned}$$

Así pues

$$\begin{aligned}
E[\chi] &= \sum_{k=0}^{\infty} k \left(\frac{n}{n+1}\right)^k \frac{1}{n+1} \\
&= \frac{n}{(n+1)^2} \sum_{k=0}^{\infty} k \left(\frac{n}{n+1}\right)^{k-1}.
\end{aligned}$$

El primer término de la serie es 0. Si lo extraemos y recorremos el índice k una posición se tiene que

$$\sum_{k=0}^{\infty} k \left(\frac{n}{n+1}\right)^{k-1} = 0 + \sum_{k=1}^{\infty} k \left(\frac{n}{n+1}\right)^{k-1} = \sum_{k=0}^{\infty} (k+1) \left(\frac{n}{n+1}\right)^k.$$

Ahora bien, sea $r = \frac{n}{n+1}$. Nótese que $\sum_{k=0}^{\infty} (k+1) r^k = \frac{d}{dr} \left(\sum_{k=0}^{\infty} r^{k+1} \right)$ y por la convergencia de la serie geométrica se tiene que

$$\sum_{k=0}^{\infty} r^{k+1} = r \sum_{k=0}^{\infty} r^k = \frac{r}{1-r}$$

ya que $r < 1$. Entonces se tiene que

$$\sum_{k=0}^{\infty} (k+1) r^k = \frac{d}{dr} \left(\sum_{k=0}^{\infty} r^{k+1} \right) = \frac{d}{dr} \left(\frac{r}{1-r} \right) = \frac{1}{(1-r)^2}$$

y por lo tanto

$$\begin{aligned}
 E[\chi] &= \frac{n}{(n+1)^2} \cdot \frac{1}{\left(1 - \frac{n}{n+1}\right)^2} \\
 &= \frac{n}{(n+1)^2} \cdot \frac{1}{\left(\frac{1}{n+1}\right)^2} \\
 &= n.
 \end{aligned}$$

Caso: b y n arbitrarias

Sin reemplazo

Sea la variable aleatoria χ el número de bolas negras que son muestreadas antes de que salga la primer bola blanca. El muestreo es sin reemplazo. Entonces

$$E[\chi] = \sum_{k=0}^n kP(\chi = k).$$

Primeramente se calculan las probabilidades $P(\chi = k)$ considerando que el muestreo es sin reemplazo:

$$\begin{aligned}
 P(\chi = 0) &= \frac{b}{n+b} \\
 P(\chi = 1) &= \frac{n}{n+b} \cdot \frac{b}{n+b-1} \\
 P(\chi = 2) &= \frac{n}{n+b} \cdot \frac{n-1}{n+b-1} \cdot \frac{b}{n+b-2} \\
 &\vdots \\
 P(\chi = k) &= \frac{b}{n+b} \prod_{i=0}^{k-1} \frac{n-i}{n+b-(i+1)}
 \end{aligned}$$

Así pues

$$\begin{aligned}
 E[\chi] &= \sum_{k=1}^n kP(\chi = k) \\
 &= \sum_{k=1}^n k \frac{b}{n+b} \prod_{i=0}^{k-1} \frac{n-i}{n+b-(i+1)} \\
 &= \frac{b}{n+b} \sum_{k=1}^n k \prod_{i=0}^{k-1} \frac{n-i}{n+b-(i+1)} \\
 &\approx \frac{b}{n+b} \sum_{k=1}^n k \left(\frac{n}{n+b} \right)^k.
 \end{aligned}$$

La última aproximación es útil para decir que el muestreo sin reemplazo tiene un valor esperado menor que el del muestreo con reemplazo. Esto se verá a continuación.

Con reemplazo

Sea la variable aleatoria χ el número de bolas negras que son muestreadas antes de que salga la primer bola blanca. El muestreo es con reemplazo. Entonces

$$E[\chi] = \sum_{k=0}^{\infty} kP(\chi = k).$$

Primeramente se calculan las probabilidades $P(\chi = k)$ considerando que el muestreo es con reemplazo:

$$\begin{aligned}
 P(\chi = 0) &= \frac{b}{n+b} \\
 P(\chi = 1) &= \frac{n}{n+b} \cdot \frac{b}{n+b} \\
 P(\chi = 2) &= \left(\frac{n}{n+b} \right)^2 \frac{b}{n+b} \\
 &\vdots \\
 P(\chi = k) &= \left(\frac{n}{n+b} \right)^k \frac{b}{n+b}
 \end{aligned}$$

Así pues, al utilizar los mismos argumentos que en el caso n arbitraria y $b = 1$ se obtiene

$$\begin{aligned} E[\chi] &= \sum_{k=0}^{\infty} k \left(\frac{n}{n+b} \right)^k \frac{b}{n+b} \\ &= \frac{bn}{(n+b)^2} \sum_{k=0}^{\infty} k \left(\frac{n}{n+b} \right)^{k-1} \\ &= \frac{bn}{(n+b)^2} \cdot \frac{1}{\left(1 - \frac{n}{n+b}\right)^2} \\ &= \frac{bn}{(n+b)^2} \cdot \frac{1}{\left(\frac{b}{n+b}\right)^2} \\ &= \frac{n}{b}. \end{aligned}$$

A.8. Modelo de computación en la práctica

Si bien es cierto que en la sección 1.4 se introdujo el modelo de computación más estándar, la máquina de Turing, en la práctica una computadora actual no es equivalente a una máquina de Turing. Un modelo de computación que se asemeja a las computadoras actuales, que fue propuesto por Michael Fredman y Dan Willard en 1990 [30], es el conocido como "Word RAM model", el cual se propuso para simular lenguajes de programación como C. En este modelo la memoria está compuesta por palabras de w bits. Las computadoras modernas utilizan palabras de 64 bits. Este modelo tiene la ventaja de que las operaciones aritméticas básicas se realizan en tiempo constante. En este trabajo utilizamos este modelo para calcular las complejidades computacionales de los distintos algoritmos expuestos en pseudocódigo.

Bibliografía

- [1] A. Aho, J. Hopcroft, and J. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley Publishing Company, 1974.
- [2] J. Anaya, J. Chalopin, J. Czyzowics, A. Labourel, A. Pelc, and Y. Vaxes. Collecting Information by Power-Aware Mobile Agents. In M. K. Aguilera, editor, *Distributed Computing*, volume 7611 of *Lecture Notes in Computer Science*, Salvador, Brazil, 2012. International Symposium on Distributed Computing, Springer.
- [3] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. *The Traveling Salesman Problem*. Princeton University Press, 2006.
- [4] S. Arora and B. Barak. *Computational Complexity*. Cambridge University Press, 2009.
- [5] B. Aspvall, M. Plass, and E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3), 1978.
- [6] E. Bampas, S. Das, D. Derenjowski, and C. Karousatou. Collaborative Delivery by Energy-Sharing Low-Power Mobile Robots. In A. Fernández, T. Jurdzinski, M. Mosteiro, and Y. Zhang, editors, *Algorithms for Sensor Systems*, volume 10718 of *Lecture Notes in Computer Science*. ALGOSENSORS, Springer, Cham, 2017.
- [7] A. Bärtschi, D. Graf, and M. Mihalák. Collective fast delivery by energy-efficient agents. In I. Potapov, P. Spirakis, and J. Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science*, volume 117, pages 56:1–56:16. MFCS, LIPICS, 2018.
- [8] B. Bollobás. *Random Graphs*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2001.
- [9] B. Bollobas, C. Borgs, J. Chayes, J. Han, and D. Wilson. The scaling window of the 2-SAT transition. Technical report, Microsoft Research and University of Memphis and Trinity College, 2001.

- [10] J. A. Bondy and U. S. R. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2008.
- [11] A. G. Bromley. Charles babbage’s analytical engine, 1838. *Annals of the History of Computing*, 4(3), 1982.
- [12] D. M. Burton. *The History of Mathematics: An introduction*. McGraw-Hill, 2011.
- [13] J. Chalopin, S. Das, M. Mihalák, P. Penna, and P. Widmayer. Data delivery by energy-constrained mobile agents. In *9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 111–122. ALGOSENSORS, 2013.
- [14] J. Chalopin, R. Jacob, M. Mihalák, and P. Widmayer. Data delivery by energy-constrained mobile agents on a line. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *Automata, Languages, and Programming*, volume 8573 of *Lecture Notes in Computer Science*, pages 423–434, Berlin, Heidelberg, 2014. ICALP, Springer.
- [15] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):354–363, 1936.
- [16] A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *International Congress for Logic Methodology and Philosophy of Science*, pages 24–30, 1964.
- [17] J. E. Cohen. The asymptotic probability that a random graph is a unit interval graph, indifference graph, or proper interval graph. *Discrete Mathematics*, 40:21–24, 1982.
- [18] S. Cook. The complexity of theorem-proving procedures. *Proceedings Third Annual ACM Symposium*, 1971.
- [19] W. J. Cook. *In Pursuit of the Traveling Salesman*. Princeton University Press, 2012.
- [20] N. R. Council. *Getting Up to Speed: The Future of Supercomputing*. The National Academy Press, 2005.
- [21] J. Crawford and L. Auton. Experimental results on the crossover point in satisfiability problems. *Proceedings 11th National Conference on Artificial Intelligence*, pages 21–27, 1993.
- [22] J. Czyzowics, D. Krzysztof, J. Moussi, and R. Wojciech. Algorithms for Communication Problems for Mobile Agents Exchanging Energy. ArXiv:1511.05987v1 [cs.DS], Nov. 2015.

-
- [23] G. B. Dantzig. The simplex method, 1956.
- [24] S. Dasgupta. *It Began with Babbage: The Genesis of Computer Science*. Oxford University Press, 2014.
- [25] J. Edmonds. Maximum partition of a matroid into independent subsets. *Journal of Research of the National Bureau of Standards*, 69B(1 and 2), 1964.
- [26] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, pages 449–467, 1965.
- [27] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B(4), 1966.
- [28] Erdős, P. and Rényi, A. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5, 1960.
- [29] H. Eves. *An introduction to the history of Mathematics*. Saunders College Publishing, sixth edition, 1990.
- [30] M. Fredman and D. Willard. Blasting through the information theoretic barrier with fusion trees. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of Computing*, 1990.
- [31] J. Fuegi and J. Francis. Lovelace & babbage and the creation of the 1843 notes. *IEEE Annals of the History of Computing*, 2003.
- [32] M. Garey and D. Johnson. *Computers and Intractability*. Bell Telephone Laboratories incorporated, 1979.
- [33] S. B. Gelfand and S. K. Mitter. Analysis of simulated annealing for optimization. In *24th IEEE Conference on Decision and Control*, pages 779–786, 1985.
- [34] E. Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, Dec. 1959.
- [35] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. *Monatshefte für Mathematik und Physik*, 38:1, 1931.
- [36] A. Goerdt. A threshold for unsatisfiability. *Journal of Computer and System Sciences*, 53:469–486, 1996.
- [37] H. H. Goldstine. *The Computer: from Pascal to von Neumann*. Princeton University Press, 1972.

- [38] W. Gutjahr. Convergence analysis of metaheuristics. In V. Maniezzo and T. Stutzle, editors, *Hybridizing Metaheuristics and Mathematical Programming*, volume 10: Matheuristics. Springer, 2009.
- [39] K. B. Hamrick. The history of the hand-held electronic calculator. *The American Mathematical Monthly*, 103(8):633–639, 1996.
- [40] Hewlett-Packard. Hpe and amd power complex scientific discovery in worlds fastest supercomputer for u.s. department of energys (doe) national nuclear security administration (nnsa), 2020.
- [41] D. Hilbert and W. Ackermann. *Grundzüge der theoretischen Logik*. Springer, 1928.
- [42] J. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.
- [43] R. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 40:85–103, 1972.
- [44] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:6, 1983.
- [45] V. Klee and G. J. Minty. How good is the simplex algorithm? In *Inequalities*, volume III, pages 159–175. Academic Press, 1972.
- [46] D. Knuth. *The Art of Computer Programming*. Addison-Wesley, 1968.
- [47] E. D. Knuth. Big omicron and big omega and big theta. *SIGACT News*, 1976.
- [48] L. Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.
- [49] Y. Matiyasevich. Enumerable sets are diophantine. *Dokl. Akad. Nauk SSSR*, 191(2):279–282, 1970.
- [50] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. *Proceedings 10th National Conference on Artificial Intelligence*, pages 459–465, 1992.
- [51] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic 'phase transitions'. *NATURE*, 400:133–137, 1999.
- [52] J. Napier. *Rabdology*. MIT Press, 1990.

-
- [53] T. H. of Computer Project. Cray 1, 2013.
 - [54] B. Reed and V. Chvátal. Mick gets some (the odds are on his side) (satisfiability). In *SFCS '92: Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 620–627. IEEE Computer Society, 1992.
 - [55] G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE transactions on neural networks*, 5:96–101, Feb. 1994.
 - [56] E. Scheinerman. An evolution of interval graphs. *Discrete Mathematics*, 82:287–302, 1990.
 - [57] E. Scheinerman. *Approaching Asymptotics*. Edward R. Scheinerman, 2019.
 - [58] S. Smale. On the average number of steps of the simplex method of linear programming. *Mathematical Programming*, (27):241–262, 1983.
 - [59] K. Sörensen. Metaheuristics - the metaphor exposed. *International Transactions in Operational Research*, Jan. 2013.
 - [60] J. Spall. *Introduction to stochastic search and optimization*. John Wiley & Sons, 2003.
 - [61] M. Technologies. Reaching the summit with infiniband, 2018.
 - [62] B. Trajtenbrot. *Los algoritmos y la resolucion automatica de problemas*. 1977.
 - [63] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42:230–265, 1937.
 - [64] R. Vereá. Improvement in calculating machines, 1878.
 - [65] Wikipedia. Anexo:historia de la computación.
 - [66] Wikipedia. Cdc 6600, 2020.
 - [67] Wikipedia. Computer performance by orders of magnitude, 2020.
 - [68] Wikipedia. Cray-2, 2020.
 - [69] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, pages 67–82, 1997.
 - [70] N. Yugami. Theoretical analysis of davis-putnam procedure and propositional satisfiability. *Automated Reasoning*, pages 282–288, 1994.
 - [71] M. Zukerman, L. Jia, T. Neame, and G. Woeginger. A polynomially solvable special case of the unbounded knapsack problem. *Operations Research Letters*, 29:13–16, 2001.