Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

El problema del viajante de comercio online sobre espacios discretos

Tesis de Licenciatura en Ciencias de la Computación
de la Universidad de Buenos Aires

Mauro Aprea
Gustavo Sadovoy

Directores de Tesis: Dr. Esteban Feuerstein, Lic. Alejandro Strejilevich de Loma

Buenos Aires, diciembre de 2006

A mis abuelos,
que supieron comprenderme siempre.

M.A.

A mis abuelos,
que le hubieran dado un significado
muy especial a este momento.

G.S.

Resumen

En la versión online del clásico problema del viajante de comercio (OLTSP) las ciudades a visitar se van conociendo mientras el viajante ejecuta su recorrido. Este problema se modela con un servidor que se desplaza en un espacio métrico para satisfacer requerimientos, teniendo como objetivo terminar su servicio lo antes posible. Los requerimientos se van presentando en distintos puntos del espacio y su existencia sólo es conocida a partir del momento de presentación. Se distinguen dos variantes para OLTSP: en la variante Homing el servidor debe retornar al punto de partida para completar el servicio, mientras que en la variante Nomadic el servicio finaliza al atender todos los requerimientos.

Los trabajos anteriores que analizan OLTSP suelen considerar que el espacio es continuo y que el servidor puede cambiar de dirección en cualquier parte del recorrido. A diferencia de ese enfoque, presentamos una versión discreta (DOLTSP) en la que el espacio métrico se representa utilizando un grafo con longitudes asociadas a sus ejes y sólo se le permite al servidor modificar su recorrido cuando está sobre algún vértice. Esta limitación afecta directamente a la capacidad de reacción del servidor y potencia el riesgo asociado a cada decisión.

En esta Tesis proponemos diversos algoritmos para ambas variantes de DOLTSP cuando el grafo sobre el que se mueve el servidor es una cadena y medimos el desempeño de cada uno calculando su competitividad. Esta es la forma de análisis de peor caso más difundida para evaluar algoritmos online, y consiste en comparar el costo de las soluciones online contra el de una solución óptima offline, es decir, aquella que podría obtenerse si se conociera de antemano la secuencia completa de requerimientos. Mostramos que el problema discreto tiene asociadas competitividades más elevadas que el problema continuo y que algunos de los algoritmos que presentamos alcanzan la mejor competitividad posible.

Además, hacemos un análisis empírico seleccionando un conjunto significativo de instancias de DOLTSP y midiendo la calidad de las soluciones generadas por cada algoritmo. Mostramos con este análisis que los algoritmos que tienen la mejor competitividad no son los que mejor funcionan en la práctica.

Abstract

In the online traveling salesman problem (OLTSP), the cities to visit are known while the salesman is traveling. This problem is modeled with a server that moves in a metric space to satisfy requirements, aiming at completing the service as early as possible. Requirements are released at different points of the space and their existence becomes known only after the release time. Two variants are distinguished: in Homing OLTSP the server has to return to the starting point in order to complete its task. In Nomadic OLTSP the service is finished when all requirements are served.

Previous studies on OLTSP usually consider a continuous metric space where the server has no restrictions to change its direction. Unlike that approach, we introduce a discrete version (DOLTSP) and represent the metric space using a weighted graph, where the server is allowed to modify its route only if it is on a vertex. This limitation directly affects the capacity of the server to react and increases the risk related to each decision.

In this Thesis we propose algorithms for both variants of DOLTSP when the graph is a chain and we measure their performance using competitive analysis, the most widely accepted method for evaluating online algorithms. This worst-case analysis consists of comparing the cost of online solutions against the offline optimal cost, i.e., that of a solution that could be generated if the whole sequence of requirements were known beforehand. We show that the discrete problem has higher competitive ratios than the continuous one and that some of the algorithms we present achieve the best possible competitive ratio.

In addition, we perform an empirical analysis generating a significant set of instances for DOLTSP and measuring the quality of the solutions generated by each algorithm. We show that algorithms with the best competitive ratio do not necessarily have the best performance in practice.

Agradecimientos

El primer agradecimiento es para nuestros directores, Esteban y Alejandro, que confiaron en nosotros y se entusiasmaron con el proyecto. Además de guiarnos muy de cerca en las distintas etapas por las que atravesamos, ayudaron a que este trabajo alcanzara una calidad muy superior a la que nos creíamos capaces de lograr.

A nuestros padres les debemos, además de la existencia, un par de cualidades que nos fomentaron desde que éramos chicos, sin las cuales hubiera sido imposible llevar adelante un proyecto de estas dimensiones: la curiosidad y la pasión. Ellos estuvieron muy presentes en nuestra formación y si logramos llegar a este punto es también, sin duda, un logro de ellos.

Queremos destacar el generoso apoyo y especialmente la paciencia que nos brindaron Laura y Magalí durante todo el desarrollo de este proyecto. Su compañía fue esencial para nosotros y significó una motivación constante para progresar. También recibimos el apoyo de nuestros familiares y amigos, a quienes en reiteradas ocasiones tuvimos que postergar para poder dedicar algunas horas al avance de nuestra Tesis. No podemos dejar de agradecer a nuestros compañeros de trabajo y a las compañías donde trabajamos —Baufest y Telecom Personal— que generosamente nos ofrecieron valiosas horas y distintos recursos (por ejemplo, los que usamos para realizar esta impresión).

Finalmente queremos expresar que si logramos llegar con éxito a esta etapa, esto de ninguna manera se debió a un esfuerzo exclusivamente individual. Por el contrario, si hay algo que aprendimos —y de hecho pusimos en práctica al elegir desarrollar juntos nuestra Tesis— es que trabajando en equipo nuestras capacidades se potencian. En nuestro paso por la Facultad, tuvimos el honor de compartir enriquecedoras experiencias con personas que aún hoy siguen siendo para nosotros ejemplos a seguir. A riesgo de cometer involuntarias omisiones, vamos a nombrar a Gerardo, Diego, Juan Pablo, Mariano, Federico, Germán, Alan, Rubén, Matías y Alejandro, quienes compartieron con nosotros prácticamente toda la carrera y nos aportaron valores fundamentales que trascienden lo académico.

A todos ellos, muchas gracias.

Tabla de contenidos

Resumen **iii**

Abstract **v**

Agradecimientos **vii**

Tabla de contenidos **ix**

1 Introducción **1**

1.1 Problemas de ruteo de vehículos 2

1.2 Optimización online 3

1.3 Análisis de competitividad 4

1.4 La noción de tiempo real 5

1.5 Espacios continuos y espacios discretos 6

1.6 Sinopsis de la Tesis 7

2 Definición de DOLTSP **9**

2.1 El problema 9

2.2 Estrategias 12

2.3 Competitividad de un algoritmo online 13

2.4 Adversarios 13

2.5 Espacios métricos 15

2.6 Relación con la versión tradicional de TSP 17

3 Cotas inferiores generales **19**

3.1 La variante Homing 20

3.2 La variante Nomadic 23

4 Estrategias **29**

4.1 Algoritmos entusiastas 30

4.1.1 Zig-Zag y Replan 31

4.2 Algoritmos cautelosos 39

4.3 Demorar el regreso al origen 42

4.3.1 Cálculo de la espera 43

4.3.2 Correctitud de WBR 47

4.4 Demorar el próximo movimiento 52

4.4.1 Cuánto esperar y hacia dónde partir 54

4.4.2 Correctitud de WBB 59

4.5 Otros algoritmos online 64

4.5.1 WBB en cadenas completas 64

4.5.2 WBB para la variante Homing 65

4.5.3 Statistic WBB 66

4.5.4 Statistic Replan 68

5 Modelo de simulación **71**

5.1 Atributos de una instancia 71

5.2 Espacio muestral 75

5.2.1 Cantidad de vértices 75

5.2.2 Tipo de cadena 76

5.2.3 Distribución de vértices 77

5.2.4 Cantidad de requerimientos . . . 78
5.2.5 Distribución espacial de requerimientos . . . 79
5.2.6 Rango temporal . . . 80
5.2.7 Distribución temporal de requerimientos . . . 80
5.3 Calidad de las soluciones . . . 81
5.3.1 Algoritmo óptimo de ida y vuelta . . . 82
5.3.2 Algoritmo óptimo recursivo . . . 82
5.4 Ambiente de simulación . . . 85
5.4.1 Alcance . . . 85
5.4.2 Modelo . . . 85
5.4.3 Diseño conceptual . . . 86
5.4.4 Definición de instancias . . . 87
5.5 Proceso de simulación . . . 88

6 Análisis de resultados **91**
6.1 Comportamiento general de las estrategias . . . 92
6.2 Instancias difíciles . . . 95
6.3 Cautela vs. entusiasmo . . . 97
6.4 Impacto de cada atributo . . . 99

7 Conclusiones **111**
7.1 Resumen de las estrategias presentadas . . . 112
7.2 Principales logros . . . 113
7.3 Resumen de resultados teóricos . . . 114
7.4 Trabajo futuro . . . 115

Bibliografía **116**

.

Capítulo 1

Introducción

Tomaremos una situación muy cotidiana como punto de partida. Supongamos que nos piden hacer las compras para la casa y nos entregan una lista con los productos que debemos conseguir: pan, leche y un shampoo. Nuestro objetivo es llegar a casa cuanto antes con todos los productos que nos pidieron. Con este fin, consideramos la ubicación geográfica de la panadería, el almacén y la perfumería y determinamos el recorrido más corto que, partiendo de casa, nos permita visitar dichos negocios y regresar.

Sin saberlo, nos estamos enfrentando a lo que se conoce en la literatura como un *problema de optimización*. En este tipo de problemas, el desafío es encontrar una solución que tenga el menor costo posible (o que maximice el beneficio). Existe una gran variedad de problemas de optimización, cada uno de ellos caracterizado por la descripción de lo que se llama el *conjunto de soluciones factibles* para una *instancia* del problema.

En el ejemplo de las compras, que podríamos denominar *el problema de los mandados*, la instancia está determinada por el conjunto de negocios que debemos visitar y sus posiciones geográficas respecto a la casa. El conjunto de soluciones está formado por todos los recorridos que parten de la casa y regresan a ella luego de pasar por los distintos locales o, lo que es equivalente, todas las permutaciones entre los negocios que nos indican en qué orden debemos visitarlos. El costo de una solución específica está dado por la distancia total recorrida. Si asumimos que caminamos a una cierta velocidad constante, también podemos plantear el costo en función del tiempo que nos lleva completar la tarea. El objetivo es minimizar el costo, es decir, regresar a casa lo antes posible.

Ahora que conocemos el problema, probablemente queramos pensar en una estrategia general para abordarlo, en lugar de guiarnos por decisiones intuitivas. Una posible estrategia sería visitar primero el negocio más cercano a la casa, luego el más cercano al primero que visitamos y finalmente ir al tercer negocio, antes de regresar. A este tipo de estrategias, que nos proveen un procedimiento para llegar a una solución factible a partir de una instancia, las conocemos con el nombre de *algoritmos*. Un algoritmo debería también poder determinar si para cierta instancia no existe ninguna solución factible. Un algoritmo que ofrece siempre una solución con el menor costo posible

es denominado *algoritmo óptimo*.

Volvamos ahora al ejemplo original. Supongamos que hemos decidido pasar primero por el almacén, luego por la perfumería y finalmente por la panadería. Luego de pasar por el almacén, cuando estamos a punto de llegar a la perfumería, nos suena el teléfono celular. Es una llamada de casa, pidiéndonos que traigamos además un kilo de cebollas. ¡Qué lástima, la verdulería está justo al lado del almacén! De haber sabido antes que nos pedirían las cebollas, habríamos pasado por la verdulería al salir del almacén... Entonces ¿qué debemos hacer? ¿damos media vuelta y vamos a la verdulería o dejamos las cebollas para lo último? Supongamos que elegimos la primera opción porque, después de todo, no estamos tan lejos. Un poco resignados, volvemos sobre nuestros pasos camino a la verdulería y, ya con el kilo de cebollas en nuestra bolsa, continuamos el recorrido que habíamos planificado. Cuando estamos pagando en la panadería, la etapa final de nuestro cometido, suena nuevamente el celular: *"¿podrás traer dos kilos de cebollas, en lugar de uno?"*.

Es claro que el problema de los mandados cambia radicalmente si no contamos desde un comienzo con la lista completa de los negocios que debemos visitar. A este nuevo problema lo podemos llamar "el problema *online* de los mandados". La mayoría de los problemas de optimización tienen su versión *offline* y su versión *online*. La diferencia entre estas dos variantes está en la cantidad de información disponible al diseñar la solución. En la versión *offline* de los problemas, los algoritmos generan soluciones a partir de una instancia completa, mientras que en la versión *online* la información necesaria se va completando a medida que la solución va siendo generada e incluso ejecutada.

1.1 Problemas de ruteo de vehículos

Informalmente, en un *problema de ruteo de vehículos*, VRP [Ste78, Sav85, dP02, DB84], se requiere transportar objetos de un lugar a otro, usando cierto número de vehículos o *servidores*. Pueden formularse distintas variantes de VRP alterando la función objetivo y combinando una serie de restricciones relacionadas, por ejemplo, con la capacidad de los servidores, las posiciones donde comienzan o finalizan los recorridos o los momentos a partir de los cuales (o hasta los cuales) los objetos pueden ser transportados. Esta clase de problemas forma parte de una clase más grande que se conoce como *problemas de optimización combinatoria*, donde la cantidad de soluciones factibles suele ser exponencial respecto al tamaño de la instancia. Existe una gran variedad de problemas que no incluyen explícitamente vehículos ni transportes pero que, de todos modos, pueden ser interpretados como problemas de ruteo.

Uno de los problemas de ruteo más estudiados es *Dial-a-Ride*, DaR [Psa80, JOPW86]. Aquí, un servidor con capacidad limitada debe transportar objetos de un punto a otro de cierto espacio métrico. El *problema de los mandados* que describimos al comienzo es un caso particular de DaR en el que el destino de todos los traslados es siempre nuestra casa y la capacidad del vehículo está determinada por la bolsa que cargamos caminando. Si los productos que tenemos que comprar no entran todos juntos en la bolsa, cada vez que ésta se llene tendremos que volver a casa y descargarla

antes de continuar realizando compras.

Si nos olvidamos de la compra en sí misma y nos concentramos en los negocios que debemos visitar, podemos ver al *problema de los mandados* como un problema de ruteo donde el destino de los objetos a transportar coincide siempre con el origen (es decir que no hay que transportar nada) y finalmente el vehículo debe regresar al punto de donde partió. Acabamos de describir un clásico problema de optimización combinatoria, llamado *Traveling Salesman Problem*, TSP [Kru56, Lin65].

Finalmente, si cambiamos los roles y suponemos ahora que son los negocios quienes esperan ser visitados cuanto antes para que les suministremos mercadería, probablemente sea razonable intentar minimizar la suma de los tiempos de atención de los requerimientos. Esta función de costos se conoce con el nombre de *latencia* y da origen a otro problema de ruteo que se llama *Traveling Repairman Problem* [ACP+86].

Desde luego, cada uno de estos problemas puede ser estudiado también en su versión *online*, donde los requerimientos no están disponibles desde el comienzo sino que se van presentando en forma progresiva.

1.2 Optimización online

Muchas veces, al analizar un problema de optimización, se suele asumir un conocimiento completo de las instancias. Sin embargo, como vimos en el ejemplo de los mandados, en la realidad no siempre está disponible de antemano toda la información necesaria para generar la mejor solución. A veces se hace necesario tomar algunas decisiones basadas en información parcial, para luego ir adaptando la solución a medida que se vaya contando con más datos.

Esta observación ha motivado la investigación de lo que se conoce actualmente como *optimización online* [FW98]. Un algoritmo de optimización se llama *online* si toma decisiones (computa una solución parcial) con cada nueva porción de información que se le suministra. En un problema de optimización online, la entrada se modela como una secuencia usualmente finita de requerimientos $r_1, r_2, \ldots, r_k$ que es revelada paso a paso a un algoritmo online. La forma concreta en que este proceso se realiza depende del paradigma online que se esté utilizando. Los dos modelos más comunes son el *modelo de secuencia* y el *modelo de tiempo real* [Kru01].

En el modelo de secuencia los requerimientos deben ser atendidos en el orden en que se presentan. Más precisamente, cuando el servidor está ocupado atendiendo un requerimiento, no tiene ningún conocimiento sobre la existencia de otros requerimientos posteriores en la secuencia. Tampoco es conocida la cantidad total de requerimientos. Cada vez que uno de ellos es atendido, puede presentarse uno nuevo o, eventualmente, se le notifica al algoritmo online que la secuencia fue servida completamente. En este modelo, las decisiones que se toman acerca de cómo servir cada requerimiento son irrevocables.

Podemos encontrar un ejemplo de este modelo en un clásico problema de optimización denominado *paging problem* [Feu95]. En este caso, se considera un sistema con dos niveles de memoria: una *memoria caché* pequeña y rápida que cuenta con k páginas y otra *memoria principal* más grande pero más lenta que cuenta con un total de N páginas. Cada requerimiento es una solicitud de una página $p \in \{1, \ldots, N\}$. Si tal página es una de las k que están en la memoria caché, el costo de atender el requerimiento es 0. En caso contrario, es necesario remover alguna de las páginas de la caché y reemplazarla por la página que fue solicitada, con un costo de 1. Un algoritmo de paginación determina cuál es la página que debe ser removida en este caso. El objetivo del problema es minimizar el número de *page faults*, es decir, la cantidad de veces que se solicita una página que no está en la memoria caché.

En el modelo de tiempo real, cada requerimiento tiene asociado el *tiempo de presentación* a partir del cual se hace visible para el algoritmo y pasa a estar disponible para ser servido. Los tiempos de presentación son números reales no negativos. La secuencia de requerimientos $r_1, r_2, \ldots, r_k$ debe estar ordenada en forma no decreciente según los tiempos de presentación. A diferencia del *modelo de secuencia*, el algoritmo puede, en general, esperar y servir los requerimientos en el orden que prefiera. Incluso puede revocar cualquiera de las decisiones que haya tomado, siempre que aún no las haya ejecutado. Pero este proceso no puede extenderse demasiado: bajo el modelo de tiempo real, el tiempo que transcurre es una de las variables que forman parte del costo que el algoritmo online deberá minimizar. Todos los problemas que estudiaremos en esta Tesis utilizan este último modelo.

1.3 Análisis de competitividad

Los problemas de optimización online ya estaban siendo estudiados en las décadas de los 70 y 80. Sin embargo, la investigación en esta área recién pudo difundirse considerablemente cuando Sleator y Tarjan [ST85] sugirieron comparar el costo de los algoritmos online con el costo óptimo offline, dando lugar a lo que luego se conoció con el nombre de *análisis de competitividad*. Un algoritmo online se dice ρ-competitivo si para cualquier secuencia de entrada el costo de la solución que produce es menor o igual que ρ veces el costo de la solución producida por un *algoritmo óptimo offline* para esa misma secuencia. Este algoritmo offline genera una solución óptima teniendo conocimiento previo de la secuencia completa de requerimientos.

Notemos que en esta definición no se hace ninguna restricción sobre los recursos computacionales que pueden utilizar los algoritmos. El único recurso escaso en el análisis de competitividad es la información. Este análisis suele explicarse como un juego entre un *jugador online* y un *adversario offline*. El jugador online usa un algoritmo online para procesar una secuencia de entrada generada por el adversario. El adversario conoce la estrategia del jugador online y, por lo tanto, puede construir una secuencia de requerimientos que maximice la relación entre el costo del jugador y el costo óptimo.

El análisis de competitividad es una forma de *análisis de peor caso*, muchas veces criticado por

su excesivo pesimismo. En ciertos contextos, el adversario resulta tan poderoso que se dificulta el análisis de las estrategias online. Con el fin de atenuar esta situación, se han desarrollado distintas alternativas que amplían el concepto original de análisis de competitividad.

En el *análisis comparativo* simplemente se restringen las capacidades del algoritmo óptimo offline. Este concepto fue introducido en el contexto del problema de paginación [KP94]. Los autores comparan la eficiencia de un algoritmo online contra otro que tiene cierta capacidad de predicción limitada. También es comúnmente utilizado en problemas online relacionados con las finanzas, como el de la selección de la cartera de inversiones [BEY98]. Nosotros usaremos una variante de análisis comparativo cuando presentemos el *adversario fair* en el siguiente capítulo y mostremos, en el Capítulo 3 y el Capítulo 4, que con este adversario la competitividad puede diferir con la calculada usando un *adversario standard*.

Otro medio utilizado para fortalecer la posición de las estrategias online, conocido como *resource augmentation* [PSTW97, PK95, ABF96, ST85], consiste en aumentar directamente sus capacidades respecto al adversario offline. En un problema de ruteo, por ejemplo, esto puede significar que al jugador online le asignemos vehículos más veloces que los que usa el adversario.

Por último, haremos una breve referencia al *adversario difuso*, presentado en [KP94]. En este caso, el adversario determina la secuencia de entrada en función de cierta distribución probabilística. Aunque tal distribución no es completamente conocida por el algoritmo online, la clase de distribuciones a la que pertenece está determinada de antemano y el algoritmo online puede usar esta información para diseñar la solución.

Todas estas extensiones al análisis de competitividad han sido útiles para ciertos problemas específicos y en algunos casos permitieron obtener resultados significativos. Sin embargo, aún no se conoce ninguna alternativa que permita reemplazar al análisis de competitividad como herramienta general para el análisis teórico de algoritmos online.

1.4 La noción de tiempo real

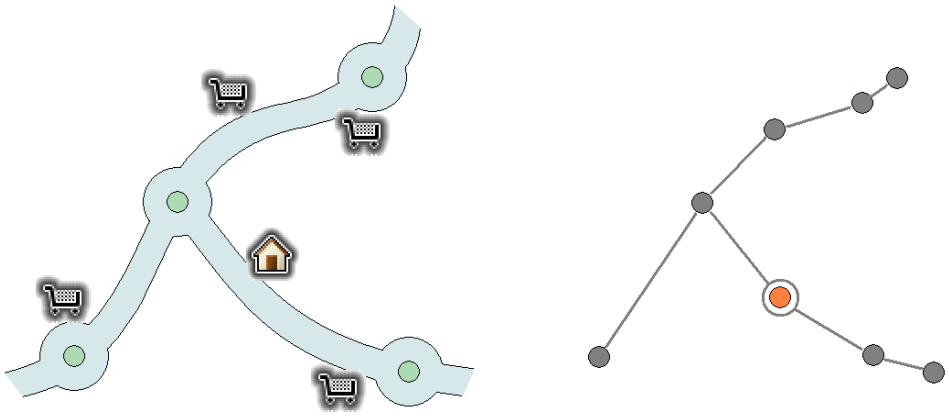
Si bien el análisis de competitividad no contempla los recursos computacionales utilizados por los algoritmos, en la práctica, para que un algoritmo pueda ser utilizado realmente "en tiempo real" es de vital importancia que pueda dar respuestas en un tiempo relativamente corto, dependiendo de las características particulares del problema. En este tipo de situaciones, el tiempo que demora el algoritmo en dar una respuesta suele afectar directamente a la calidad de la solución. Una solución entregada fuera de tiempo podría ser inútil y en algunos casos contraproducente, puesto que puede no ajustarse a los parámetros actuales del sistema, que van variando con el tiempo.

Más allá de estas cuestiones prácticas (y por razones de simplicidad), en nuestra Tesis trabajaremos sobre la hipótesis de que los tiempos computacionales empleados por los algoritmos son siempre ínfimos y despreciables. De hecho, todos los algoritmos online que proponemos tienen una complejidad de orden polinomial.

1.5 Espacios continuos y espacios discretos

Regresemos nuevamente al ejemplo del que partimos, el *problema de los mandados*. Hasta ahora hemos asumido que nuestra casa está ubicada medianamente cerca de los negocios que tenemos que visitar y, por lo tanto, es posible realizar el recorrido caminando. Eso significa que si surge un imprevisto en el camino de un negocio a otro, podemos dar media vuelta sin problemas, en cualquier punto del trayecto. Pero ¿qué sucedería si modificamos un poco estas condiciones?

Supongamos ahora que nuestra casa está ubicada en una zona rural y que para llegar a los negocios necesitamos utilizar un automóvil. Las rutas que transitamos son de doble mano, pero no se nos permite hacer giros en "U". Sólo podemos cambiar de dirección en una rotonda o entrando en un local.



En la situación que acabamos de describir, el espacio puede modelarse como un grafo donde los vértices representan negocios, rotondas o nuestra casa, y los ejes representan los segmentos de ruta que hay entre ellos, en los que nuestro vehículo no puede cambiar de dirección. Si bien la situación original también podía modelarse con un grafo, en aquélla no había ningún impedimento para alterar el recorrido en cualquier punto. Esta sutil diferencia es la principal motivación de nuestro trabajo.

Hasta ahora, en la literatura relacionada con los problemas de ruteo online, y particularmente en la versión online de TSP (OLTSP), se suelen considerar espacios métricos continuos, donde el servidor puede tomar decisiones en cualquier momento [AFL+01, dP02, Kru01]. A diferencia de este modelo, nosotros definimos el problema considerando que el tiempo es continuo y el espacio métrico es representable por un grafo con longitudes asociadas a los ejes. De esta forma, un requerimiento puede presentarse mientras el servidor está viajando de un vértice a otro, aunque éste no puede tomar ninguna acción hasta llegar al destino.

Ya estamos en condiciones de especificar el problema que estudiaremos en esta Tesis. Llamamos *Discrete Online Traveling Salesman Problem* (DOLTSP), al problema de ruteo online donde un

servidor tiene por objetivo atender distintos requerimientos que se van presentando a lo largo del tiempo, finalizando su trabajo tan pronto como le sea posible. Cada requerimiento es una solicitud de visita a un vértice de un grafo. Atenderlo significa desplazar al servidor a través de los ejes y hacerlo pasar por ese vértice en algún momento no anterior a su presentación.

Si bien definimos DOLTSP para cualquier tipo de grafo, las estrategias que estudiaremos en esta Tesis son válidas para grafos donde el servidor sólo puede moverse en dos direcciones, a las que nos referimos simplemente como "izquierda" y "derecha". Podríamos pensar en una autopista donde los cambios de dirección están permitidos sólo en determinados puntos; o en un ascensor en el que los pisos no necesariamente están dispuestos a distancias regulares; o en un sistema de distribución de piezas en una fábrica, donde un carro se desplaza sobre una guía entre los distintos puestos de trabajo. Encontramos otro ejemplo que puede asociarse con este modelo en el caso de un disco rígido, donde la cabeza de lectura/escritura se desplaza hacia el centro o hacia el exterior, sin conocerse de antemano qué cilindros le serán solicitados.

Si pensamos el modelo de un modo más abstracto, también podemos considerarlo apropiado para cualquier mecanismo donde sean necesarios sucesivos cambios de estado que no pueden ser planificados de antemano. En este caso, los vértices representan a los distintos estados y las distancias están determinadas por el costo de las transiciones. Un ejemplo en el que el diagrama de transiciones se puede representar con una cadena, es el de una versión simplificada de una caja de cambios automática. Este dispositivo decide cuándo es conveniente aumentar o disminuir la marcha en función de las revoluciones del motor. Los cambios de marcha son siempre en forma secuencial, es decir, no es posible pasar de *segunda* a *cuarta* sin antes pasar por *tercera*. Una estrategia online para este problema debería optimizar una función objetivo que contempla variables como el costo de las transiciones, el rendimiento del motor, etc.

1.6 Sinopsis de la Tesis

Esta Tesis tiene por objetivo estudiar profundamente un problema de ruteo online, utilizando para ello la herramienta más difundida en la actualidad: el *análisis de competitividad*. Una vez que definimos formalmente el alcance del problema y establecemos los mejores resultados a los que podemos aspirar, presentamos algunas estrategias válidas para distintas variantes del problema, que analizamos tanto teórica como empíricamente.

Hasta aquí hemos contextualizado el problema, explicando su origen y las circunstancias que motivan su estudio. El siguiente capítulo está destinado a formalizar los detalles de DOLTSP, introduciendo dos variantes: *Homing* y *Nomadic*. En la primera, el recorrido debe finalizar en el punto donde comenzó; mientras que en la segunda, esta restricción desaparece. Definimos también la forma en que evaluaremos la eficiencia de los algoritmos, por medio del análisis de competitividad, presentando dos modelos de adversario. Finalmente hacemos una breve diserción sobre los grafos que estudiaremos y detallamos los principales aspectos que caracterizan al problema TSP *online* en contraste con su homónimo *offline*.

En el Capítulo 3, para cada variante de DOLTSP y según el modelo de adversario, establecemos competitividades que ningún algoritmo online puede mejorar. En el Capítulo 4 distinguimos dos categorías de algoritmos online: los *entusiastas*, que mantienen ocupado al servidor mientras queden requerimientos sin servir, y los *cautelosos*, que esperan antes de tomar una decisión para reducir la posibilidad de cometer errores. Presentamos dos algoritmos entusiastas que alcanzan la mejor competitividad posible dentro de su categoría y dos algoritmos cautelosos, que mejoran la eficiencia en el peor caso y alcanzan las cotas inferiores mostradas en el Capítulo 3. Finalmente introducimos otras estrategias basadas en ideas intuitivas, aunque dejamos el análisis de competitividad para otros trabajos.

En el Capítulo 5 planteamos un modelo para hacer un análisis empírico de todas las estrategias que presentamos, seleccionando un conjunto representativo de instancias de DOLTSP. Para ello desarrollamos un entorno de simulación cuyas características también especificamos. Finalmente, en el Capítulo 6 analizamos los resultados obtenidos con la simulación y elaboramos algunas de las conclusiones que exponemos en el capítulo final, en el que planteamos, además, algunos desafíos que esperamos den origen a nuevas investigaciones.

Capítulo 2

Definición de DOLTSP

En este capítulo definimos formalmente una instancia del problema DOLTSP como una secuencia de requerimientos y a su solución como el recorrido que debe realizar un servidor para atender a todos los requerimientos presentados. Establecemos el costo que tiene cada solución y las características generales de los *algoritmos online*, que tienen por objetivo encontrar alguna solución con el menor costo posible. Para poder evaluar el desempeño de los algoritmos, planteamos los conceptos fundamentales del *análisis de competitividad* y distinguimos el adversario *fair* del *standard*. Las últimas dos secciones están dedicadas a analizar los factores que diferencian a DOLTSP de la versión continua del problema online (OLTSP) y de la versión offline tradicional (TSP).

2.1 El problema

En DOLTSP, un servidor tiene por objetivo atender distintos requerimientos que se van presentando a lo largo del tiempo, finalizando su trabajo tan pronto como le sea posible. Cada requerimiento es una solicitud de visita a un vértice de un grafo. Atenderlo significa desplazar al servidor a través de los ejes y hacerlo pasar por ese vértice en algún momento no anterior a su presentación.

DOLTSP pertenece a la clase de problemas de ruteo online, que usualmente no se definen sobre grafos sino sobre espacios métricos continuos [AFL+01, dP02, Kru01]. En estos espacios, el servidor puede detenerse o cambiar de dirección en cualquier punto del recorrido. Por el contrario, en DOLTSP el servidor sólo puede modificar su recorrido en los vértices del grafo.

Definición 2.1. Un *espacio métrico* $M = (X, d)$ es un conjunto de puntos X con una función d que indica la distancia entre cada par de puntos de X como un número real no negativo ($d : X \times X \to \mathbb{R}_{\geq 0}$). Por ser d una función de distancia, vale que $d(x, x) = 0$, $d(x, y) = d(y, x)$ y $d(x, y) + d(y, z) \geq d(x, z)$, donde x, y y z son puntos arbitrarios del espacio métrico [Rud76].

Definición 2.2. Un espacio métrico $M = (X, d)$ se dice *continuo* si para cualquier par de puntos $x, y \in X$ el camino más corto que los une es continuo, está formado por puntos de X y su longitud es $d(x, y)$ [AFL+01].

Definición 2.3. Un espacio métrico $M = (X, d)$ se dice *discreto* si los puntos de X están separados entre sí. Es decir, para cada $x \in X$ existe un número real $e(x) > 0$ tal que $\forall y \neq x$, $d(x, y) > e(x)$.

A partir de cualquier espacio métrico discreto $M = (X, d)$ donde X es finito, puede construirse un grafo completo con un vértice por cada punto de X, en el que cada eje tiene asociada la distancia entre los puntos que une. En el caso en que X sea un conjunto infinito, este mismo concepto puede extenderse si consideramos que el grafo resultante puede contener una cantidad infinita de vértices.

La transformación inversa también es posible: en un grafo conexo cualquiera $G = (V, E)$, con longitudes positivas asociadas a sus ejes, podemos definir la distancia entre dos vértices como la longitud del camino más corto que los une. Usando esta distancia, podemos decir que G define un espacio métrico en el cual V es el conjunto de puntos.

En cierto sentido, los grafos son modelos más ricos que los espacios métricos discretos: los grafos no sólo permiten definir la distancia entre cada par de vértices, sino también cuáles son los diferentes caminos posibles entre ellos. En consecuencia, podemos considerar al problema OLTSP planteado sobre espacios métricos discretos como un caso particular del mismo problema planteado sobre grafos.

En el presente trabajo analizaremos el problema OLTSP sobre grafos, dando lugar a la variante que denominamos *Discrete Online Traveling Salesman Problem*, o simplemente *DOLTSP*.

En cada grafo $G = (V, E)$ distinguiremos un vértice especial $o \in V$ al que llamaremos *origen*. Para cualquier vértice $v \in V$, notaremos con $\bar{v} = d(v, o)$ a su distancia al origen. También asumiremos que el grafo tiene al menos un vértice distinto del origen (no es un grafo *trivial*) y que las longitudes asociadas a los ejes son positivas y cumplen la desigualdad triangular.

Definición 2.4. Dado un grafo G, una *instancia* de DOLTSP consiste en una secuencia finita y no vacía $\sigma = r_1, \ldots, r_k$ compuesta por *requerimientos* o *pedidos* de la forma $r_j = (t_j, x_j)$, donde $t_j = t(r_j) \in \mathbb{R}_{\geq 0}$ indica el momento en que se presenta el pedido y $x_j = x(r_j)$ es el vértice de G que debe visitarse para atenderlo. Asumimos que la secuencia de requerimientos está dada en un orden no decreciente de tiempos de presentación. Esto es, $0 \leq t(r_1) \leq t(r_2) \leq \cdots \leq t(r_k)$. Para cualquier $t \in \mathbb{R}_{\geq 0}$ denotamos por $\sigma_{\leq t}$ a la subsecuencia de requerimientos en σ que fueron presentados hasta el momento t inclusive.

Definición 2.5. Un *recorrido* es una secuencia finita (y eventualmente vacía) de movimientos $\mu = m_1, \ldots, m_n$. Cada movimiento $m_i = (\tau_i, v_i, w_i)$ establece que en el momento τ_i el servidor

comienza a moverse desde el vértice v_i hacia su vecino w_i (a través del eje que los une). La velocidad del servidor es constante y de una unidad de espacio por cada unidad de tiempo. Por lo tanto, el servidor alcanza el vértice destino de cada movimiento m_i en el momento $\tau_i + d(v_i, w_i)$. La secuencia de movimientos debe ser válida y, para ello, cumplir con las siguientes propiedades:

i. El primer movimiento comienza en el origen ($v_1 = o$).

ii. Cada uno de los siguientes movimientos comienza donde termina el movimiento anterior y no antes de que éste se haya completado. Formalmente, $v_i = w_{i-1}$ y $\tau_i \geq \tau_{i-1} + d(v_{i-1}, w_{i-1})$ para $i = 2, \ldots, n$.

Adicionalmente puede requerirse que

iii. el último movimiento termine en el origen ($w_n = o$).

Si se satisface esta última propiedad, decimos que el recorrido es *cerrado*. En caso contrario, se trata de un recorrido *abierto*.

Definición 2.6. Una vez que se presenta un requerimiento $r = (t, x)$, consideramos que fue *servido* o *atendido* si el servidor visitó el vértice x en algún momento no anterior a t y decimos que está *pendiente* en caso contrario. Más formalmente, decimos que r es atendido con el recorrido $\mu = m_1, \ldots, m_n$ si se satisface alguna de las siguientes condiciones:

i. El pedido r se presenta en el origen y μ es una secuencia vacía.

ii. Existe un movimiento $m = (\tau, v, w) \in \mu$ tal que $v = x$ y $\tau \geq t$.

iii. El último movimiento de μ tiene destino en x.

Definición 2.7. Decimos que un recorrido μ es *solución* de una secuencia de requerimientos σ si con μ son atendidos todos los requerimientos de σ.

Para determinar el costo de una solución utilizaremos una función conocida con el nombre de *makespan*.

Definición 2.8. Dado un recorrido μ que es solución de una secuencia de requerimientos σ, notamos con $makespan(\sigma, \mu)$ al momento en que el servidor termina de atender todos los requerimientos y, si corresponde, regresa al origen.

Existen algunas otras funciones de costo citadas en la literatura. Una de ellas es la *latencia*, que se define como la suma de los tiempos en que cada requerimiento es atendido. En este sentido, cabe aclarar que la definición del problema incluye a la elección de la función de costo: distintas funciones de costo tienen asociadas distintas cotas inferiores y distintas estrategias. En este trabajo definimos DOLTSP a partir de la función de costo *makespan*.

Definición 2.9. Dada una secuencia de requerimientos σ, el *problema DOLTSP* consiste en generar un recorrido μ que sea solución para σ, de forma tal que $makespan(\sigma, \mu)$ tenga el menor valor posible. Distinguimos dos variantes de DOLTSP: en la variante *Homing* sólo son válidos aquellos recorridos que sean cerrados, mientras que para la variante *Nomadic* son válidos tanto recorridos abiertos como cerrados.

2.2 Estrategias

Un algoritmo o estrategia *online* para DOLTSP va conociendo cada uno de los requerimientos de la secuencia a medida que se van presentando. No conoce la cantidad total de requerimientos ni el momento en que se presentarán. Además, por tratarse de un grafo y no de un espacio métrico continuo, el servidor sólo puede detenerse o cambiar su dirección cuando está sobre algún vértice.

Con estas limitaciones, la estrategia debe determinar el comportamiento del servidor según se vayan presentando los requerimientos. Esencialmente, esto consiste en planificar, para cada momento t, un recorrido que sea solución de la subsecuencia $\sigma_{\leq t}$ y sea consistente con los movimientos ya comenzados.

Para una definición más precisa, tomemos una secuencia de requerimientos $\sigma = r_1, \ldots, r_k$. Al presentarse cada requerimiento r_j, un algoritmo online deberá generar un recorrido que sea solución de la subsecuencia $r_1, \ldots, r_j$ y extienda a la secuencia de movimientos comenzados hasta el momento $t(r_j)$.

Definición 2.10. Dado un recorrido μ, para cualquier $t \in \mathbb{R}_{\geq 0}$ llamamos *prefijo* de μ en el momento t a la subsecuencia compuesta por los movimientos $m_i = (\tau_i, v_i, w_i) \in \mu$ tales que $\tau_i \leq t$.

Definición 2.11. Un *algoritmo online determinístico* `ALG` para DOLTSP es una secuencia de funciones $f_j : R^j \to S$, con $j \in \mathbb{N}$, donde R^j es el conjunto de las secuencias de j requerimientos y S es el conjunto de los recorridos. La salida de `ALG` para una secuencia de requerimientos $\sigma = r_1, \ldots, r_k$ es una secuencia de recorridos $\mu_1, \ldots, \mu_k$, donde $\mu_j := f_j(r_1, \ldots, r_j)$ es solución de la subsecuencia $r_1, \ldots, r_j$. No cualquier secuencia de salida es válida: para cada $r_j \in \sigma$ donde $2 \leq j \leq k$, deben coincidir los prefijos de μ_{j-1} y μ_j al momento $t(r_j)$. Esta restricción garantiza que en cada nueva decisión el servidor comandado por `ALG` sea consistente con los movimientos ya comenzados. El costo de `ALG` para completar el servicio de σ está definido como:

$$ALG(\sigma) := makespan(\sigma, \mu_k)$$

Si bien sólo hemos dado una definición para un algoritmo online *determinístico*, existe frondosa literatura sobre algoritmos online *no determinísticos* o *aleatorios*. Estos algoritmos pueden verse simplemente como una distribución aleatoria sobre algoritmos online determinísticos [Kru01], en los que el recorrido resultante y su costo asociado son variables aleatorias. De todas maneras, en este trabajo pondremos el foco exclusivamente sobre algoritmos determinísticos.

2.3 Competitividad de un algoritmo online

El *análisis de competitividad* determina la eficiencia de una estrategia online comparando, para cualquier secuencia de requerimientos posible, el costo de la solución producida contra el costo óptimo offline.

Definición 2.12. Dada una secuencia de requerimientos σ, el *costo óptimo offline* está definido como:

$$OPT(\sigma) := \min\{makespan(\sigma, \mu) : \mu \text{ es solución de } \sigma\}$$

Definición 2.13. Sea $\rho \geq 1$ un número real. Un algoritmo online determinístico `ALG` para DOLTSP se dice ρ-competitivo si para cualquier secuencia de requerimientos el costo de la solución producida por `ALG` no es mayor que ρ veces el *costo óptimo offline* para esa misma secuencia. Es decir, para toda secuencia σ se cumple

$$ALG(\sigma) \leq \rho\, OPT(\sigma)$$

El lector puede notar que en esta definición no se hace ninguna restricción sobre los recursos computacionales que pueden utilizar los algoritmos. El único recurso escaso en el análisis de competitividad es la información.

2.4 Adversarios

El análisis de competitividad puede entenderse como una competencia entre un *jugador online* y un *adversario offline*. El primero usa una estrategia (determinística, en nuestro caso) para generar la solución a una secuencia de requerimientos generada por el segundo. El objetivo del *jugador online* es terminar cuanto antes con todos los requerimientos de la secuencia y el objetivo del adversario,

que conoce la estrategia online, es maximizar el costo de la solución online en relación al costo óptimo offline.

El concepto de *adversario* no tiene otro fin más que facilitar el análisis de competitividad. Concretamente, en lugar de analizar *todas* las secuencias de requerimientos posibles, la idea de un adversario imaginario permite que nos concentremos en el *peor caso* y consideremos sólo aquella secuencia que maximiza la competitividad de cierta estrategia online determinística.

En el presente trabajo estudiamos dos modelos de adversario: uno *standard*, que no tiene ninguna limitación y uno *fair* (justo, que juega limpio), que sólo puede desplazar el servidor offline dentro de los límites determinados por los requerimientos ya presentados. La idea de un adversario "justo" surgió en [BKdPS00] luego de considerar que el modelo de adversario *standard* es poco realista, puesto que el servidor online tiene una visión muy limitada del espacio, relacionada con los vértices donde ya se han presentado requerimientos. Si el adversario se aleja demasiado de estos lugares y logra anticiparse a la presentación de un nuevo requerimiento, está obteniendo una ventaja adicional que, según este modelo, se considera injusta.

A continuación mostramos algunas definiciones inspiradas en [Mul90], que permiten formalizar la noción de adversario *fair*, usando el concepto de *cápsula convexa* aplicado a grafos.

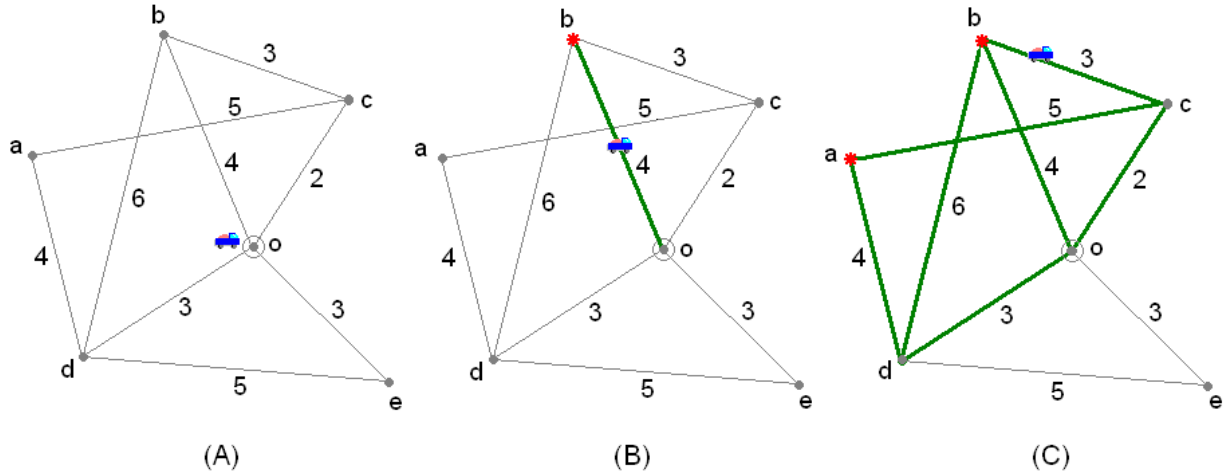
Definición 2.14. Sea $G = (V, E)$ un grafo conexo y con longitudes asociadas a sus ejes. Llamamos *segmento* entre un par de vértices v, w al conjunto de vértices formado por todos los caminos que unen v con w, cuya longitud coincide con la distancia de v a w. Un conjunto de vértices $A \subseteq V$ se dice *convexo* si contiene a todos los segmentos que conectan a cada par de vértices en A.

Definición 2.15. Sea A un conjunto de vértices en un grafo conexo y con longitudes asociadas a sus ejes. La *cápsula convexa* de A es el menor conjunto convexo de vértices que contiene a A.

Definición 2.16. Llamamos *región fair* a la cápsula convexa del conjunto formado por el origen y los vértices donde se hayan presentado requerimientos hasta un momento determinado.

Definición 2.17. Un adversario se dice *fair* si en todo momento el servidor offline está en un vértice de la *región fair* o atravesando un eje que une dos vértices de la *region fair*.

El siguiente ejemplo muestra cómo se va extendiendo la *región fair* a medida que se van presentando nuevos requerimientos.



(A) No se han presentado requerimientos aún y, por ende, la región fair está reducida a un único vértice: el origen. (B) Se presenta un primer requerimiento en b que extiende dicha región. (C) Se presenta otro requerimiento en a de modo que la región fair ahora contiene más vértices.

En general, un adversario *standard* podría sacar mejor provecho de su conocimiento sobre el futuro moviendo su servidor hacia un vértice donde se presentará un requerimiento, antes de que esto efectivamente suceda. De esta forma podría finalizar su servicio en menos tiempo que lo que le llevaría a un adversario *fair*. Debido a esto, suele haber algunas diferencias en la competitividad calculada contra los diferentes modelos de adversario.

2.5 Espacios métricos

Como dijimos en el Capítulo 1, los problemas de ruteo online suelen analizarse sobre espacios métricos continuos en lugar de grafos [AFL+01, dP02, Kru01]. Sobre este punto, muchos autores hacen referencia a [AFL+01], donde se consideran todos los espacios métricos *continuos* (ver Definición 2.2) y, bajo ciertas condiciones, aquellos espacios métricos discretos representables por grafos donde todos los ejes tienen una longitud unitaria. En estos últimos espacios métricos, se discretiza el tiempo de modo que, tanto los momentos en que se presentan requerimientos como los momentos en que el servidor comienza o termina un movimiento, coinciden con un número entero no negativo.

A diferencia de estos modelos, en nuestro trabajo definimos el problema considerando que el tiempo es continuo y el espacio métrico es representable por un grafo donde los ejes tienen longitudes arbitrarias. De esta forma, un requerimiento puede presentarse mientras el servidor está viajando de un vértice a otro sin que éste pueda tomar ninguna acción hasta llegar al destino.

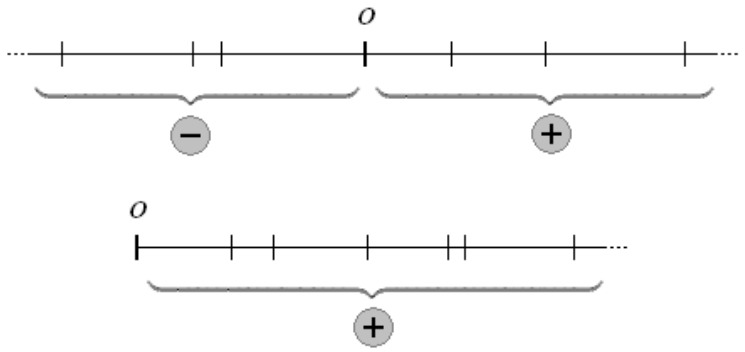
Si bien definimos DOLTSP para cualquier tipo de grafo, las estrategias que estudiaremos en esta Tesis son válidas para grafos donde el servidor sólo puede moverse en dos direcciones, a las que

nos referimos simplemente como "izquierda" y "derecha". En estos casos, podríamos describir a la topología del grafo como una *cadena*.

Definición 2.18. Una *cadena* es un un grafo conexo y sin circuitos donde ningún vértice tiene grado mayor que 2.

Definición 2.19. Llamamos *semicadena* a una cadena donde el vértice distinguido como origen tiene grado 1.

En una cadena donde el origen tiene grado 2, consideramos la existencia de dos semicadenas que comparten el origen. Denominamos *positiva* a una de ellas y *negativa* a la otra. Si el origen tiene grado 1, consideramos que la semicadena en cuestión es positiva.



Ejemplos de cadena y semicadena

Con el objeto de establecer un orden entre los vértices de una cadena, asignamos una *posición* a cada uno, cuyo valor absoluto está determinado por su distancia al origen y cuyo signo está determinado por la semicadena a la que pertenece. Esto puede representarse gráficamente en una recta horizontal, ubicando los vértices de izquierda a derecha, de acuerdo a cuál sea su posición. A lo largo del documento notaremos de la misma forma a un vértice y a su posición y usaremos los términos *derecha* o *izquierda* para indicar, respectivamente, posiciones mayores o menores que determinado punto de referencia.

Dado que cada vértice de una cadena tiene una posición que se corresponde con un número real, resulta natural asignarle también una posición al servidor, incluso cuando éste no se encuentre sobre un vértice.

Definición 2.20. En una cadena, si en el momento t el servidor online está sobre el vértice v, su *posición* es $s_t = v$. En cambio, si está en movimiento luego de haber abandonado el vértice v en el momento τ, entonces $s_t = v + \lambda(t - \tau)$, donde λ es 1 si se dirige hacia la derecha y -1 si se dirige hacia la izquierda. Cuando t se deduzca del contexto, notaremos la posición del servidor online simplemente con s, y con $\bar{s}$ a su distancia al origen.

Inclusión entre grafos

En el próximo capítulo diremos que ciertas propiedades valen para cualquier grafo que *contenga* determinada cadena. A continuación ofrecemos algunas definiciones que hacen más preciso este concepto.

Definición 2.21. Dos grafos $G = (V, E)$ y $G' = (V', E')$ con longitudes asociadas a sus ejes se dicen *isomorfos* si existe una biyección $\varphi : V \to V'$ tal que en E hay un eje (u, v) si y sólo si en E' hay un eje $(\varphi(u), \varphi(v))$ con la misma longitud. Nos referimos a φ como un *isomorfismo* entre G y G'.

Definición 2.22. Sean G y G' grafos con longitudes asociadas a sus ejes, cuyos vértices distinguidos como origen son, respectivamente, o y o'. Decimos que G' *contiene* a G si existe un isomorfismo φ entre G y un subgrafo H' de G', tal que $\varphi(o) = o'$.

2.6 Relación con la versión tradicional de TSP

Veamos que OLTSP, y en particular DOLTSP, difiere en algunos aspectos de su famoso "progenitor", *Traveling Salesman Problem*. Para comenzar, en la versión tradicional de TSP todos los requerimientos son conocidos al momento de diseñar la solución, mientras que en la versión online del problema las estrategias no saben cuántos requerimientos tiene la secuencia ni en qué momento se presentarán. Pese a esta limitación, el servidor online debe comenzar en algún momento un recorrido para atender los requerimientos pendientes. Todo esto implica un riesgo al momento de tomar cada una de las decisiones.

En segundo término, el costo de una solución no depende de la distancia total recorrida sino del tiempo que necesita el servidor para satisfacer a todos los requerimientos. En realidad podría definirse OLTSP a partir de otras funciones de costo. Pero, dado que nos basamos en el análisis de competitividad como mecanismo para establecer la eficiencia de una estrategia, si la función de costo fuera la distancia total recorrida por el servidor, es posible mostrar que ninguna estrategia sería mejor que n-competitiva, con n arbitrariamente grande. Esto equivale a decir que ningún algoritmo sería competitivo.

Para mostrar esto, alcanza con construir una secuencia de pedidos de la siguiente forma. Se presenta un requerimiento en un vértice arbitrario x, diferente del origen. En algún momento el servidor online deberá atenderlo. En ese momento, se presenta otro requerimiento en el origen. Cuando el servidor llega al origen para servirlo, aparece un nuevo requerimiento en x, y así sucesivamente. Este procedimiento puede realizarse n veces de forma tal que la distancia total recorrida por el servidor online sea $2n\bar{x}$. Sin embargo, el recorrido offline de menor distancia no es más largo

que $2\bar{x}$, porque consiste en desplazar una única vez el servidor hasta x y, si se trata de la variante *Homing*, regresar al origen.

Finalmente, notemos que al versión tradicional de TSP suele definirse como "encontrar el circuito *hamiltoniano* más corto" en cierto grafo. Un circuito hamiltoniano es un camino que pasa una única vez por cada vértice del grafo y finaliza en el mismo vértice en el que comenzó. Pero en OLTSP, debido a su naturaleza online, podría ser inevitable que el servidor online visite ciertos vértices más de una vez.

Capítulo 3

Cotas inferiores generales

En el presente capítulo planteamos algunas cotas inferiores a la competitividad, válidas para cualquier algoritmo online determinístico diseñado para las variantes Homing o Nomadic de DOLTSP. Recordemos que en la variante Homing el servidor debe regresar al origen luego de servir todos los requerimientos, mientras que en la variante Nomadic el servicio finaliza cuando ya no quedan requerimientos por servir, sin importar la posición del servidor. Para cada una de estas dos variantes, estudiamos la competitividad contra un adversario *standard* –que no tiene ninguna restricción– y contra un adversario *fair*, que debe mantenerse siempre dentro de los límites determinados por los requerimientos que ya fueron presentados.

Las cotas inferiores que mostraremos contra un adversario *fair* son válidas para cualquier grafo no trivial. En cambio, cuando consideremos un adversario *standard*, veremos que nuestras cotas inferiores varían dependiendo de las distancias entre los vértices del grafo. Mostraremos algunos ejemplos donde la cota inferior es la más elevada para cada variante.

Si vemos al análisis de competitividad como un juego entre un participante online y un adversario offline, hallar cotas inferiores significa ponerse en el lugar del adversario y definir, para cada estrategia online posible, una secuencia de requerimientos que maximice la competitividad. Es decir, debemos lograr que el cociente entre el *makespan* del recorrido online y el *costo óptimo offline* sea el más alto posible.

En general, la técnica que utilizamos en forma recurrente para encontrar cotas inferiores consiste en presentar un requerimiento r un instante después de que el servidor online abandona cierto vértice. De esta forma, el servidor se ve obligado a terminar el movimiento que acaba de empezar para luego dirigirse a servir el requerimiento r que le queda pendiente. Si τ es el momento en que el servidor online abandona el vértice, debería ser $t(r) = \tau + \varepsilon$, con $\varepsilon > 0$. Cuanto menor sea ε, menor será el costo de la solución offline y, por ende, mayor será la diferencia entre los tiempos que necesitan el jugador online y su adversario para completar el servicio. Dado que el valor de ε puede ser tan cercano a 0 como se quiera y con el objetivo de no complicar innecesariamente las demostraciones, consideraremos este valor despreciable y de aquí en más no lo mencionaremos.

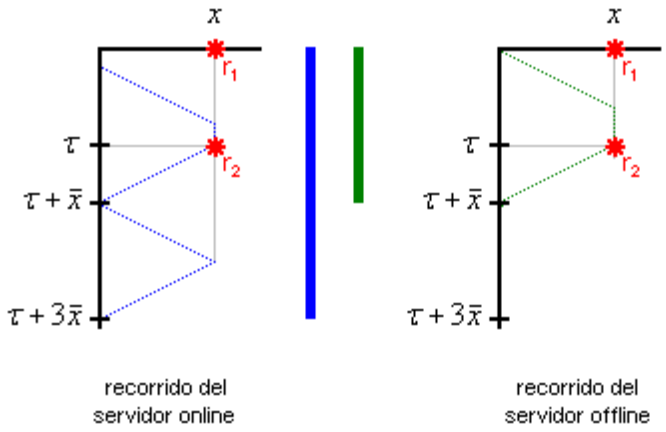
3.1 La variante Homing

A continuación mostramos una cota inferior para Homing DOLTSP contra un adversario *fair*, que vale para cualquier grafo no trivial.

Teorema 3.1. *Sea* `ALG` *un algoritmo online determinístico cualquiera para la variante Homing de DOLTSP. Entonces, la competitividad de* `ALG` *contra un adversario* fair *no es menor que* $\rho = \frac{1+\sqrt{5}}{2} \approx 1.618$.[1]

Demostración. Consideremos un grafo donde x es el vértice más cercano al origen. En el momento 0 se presenta el requerimiento r_1 en x. El servidor online deberá atenderlo y, por tratarse de la variante Homing, regresar en algún momento al origen. Sea $\tau \geq \bar{x}$ el momento en que el servidor operado por `ALG` comienza el movimiento hacia el origen, luego de haber servido r_1. El adversario podría servir r_1 y regresar al origen en tiempo $2\bar{x}$, mientras que `ALG` no podría terminar antes que $\tau + \bar{x}$. Si ocurriera que $\tau + \bar{x} \geq 2\bar{x}\rho$, no se presentaría ningún otro requerimiento y `ALG` no sería mejor que ρ-competitivo.

Asumiremos entonces que $\tau + \bar{x} < 2\bar{x}\rho$ [$\star$]. Ahora el adversario presenta un nuevo requerimiento $r_2 = (\tau, x)$, logrando concluir en tiempo $OPT(r_1, r_2) = \tau + \bar{x}$. El costo de la estrategia online es $ALG(r_1, r_2) \geq \tau + 3\bar{x}$, puesto que el servidor online llega al origen en el momento $\tau + \bar{x}$ y, una vez allí, debe ir nuevamente hasta x para servir r_2 y luego regresar al origen para completar el servicio.



En consecuencia tenemos

$$\frac{ALG(r_1, r_2)}{OPT(r_1, r_2)} \geq \frac{\tau + 3\bar{x}}{\tau + \bar{x}} = \frac{\tau + \bar{x}}{\tau + \bar{x}} + \frac{2\bar{x}}{\tau + \bar{x}} >^{\star} 1 + \frac{2\bar{x}}{2\bar{x}\rho} = 1 + \frac{1}{\rho} = \rho$$

□

[1] Puede resultarle interesante al lector observar que el valor de la cota inferior hallada coincide con la razón áurea.

Si analizamos la competitividad en la misma variante, pero contra un adversario *standard*, debemos considerar que ahora el servidor offline tiene la posibilidad de adelantarse hacia vértices donde todavía no se han presentado requerimientos y hacer así más provechoso su conocimiento sobre el futuro. Sin embargo, lo cierto es que, en un grafo, tal ventaja sólo se hace efectiva cuando existen los vértices necesarios para que el adversario pueda adelantarse. Es decir que la mejor competitividad alcanzable por una estrategia online podría variar dependiendo de la forma del grafo.

A continuación describimos una semicadena con una distribución particular de vértices que le permite a un adversario *standard* adelantarse a la presentación de requerimientos lo suficiente como para lograr que la cota inferior sea mayor que contra un adversario *fair*.[2] En esta semicadena, los vértices están ubicados en las posiciones determinadas por los valores de una sucesión que denominamos ω, y definimos como sigue:

$$\begin{aligned} \omega_0 &= 1 \\ \omega_n &= \sqrt{2} + (\sqrt{2} - 1)\ \omega_{n-1} \end{aligned} \tag{3.1.1}$$

Antes de continuar, veamos algunas propiedades de esta sucesión que nos serán útiles más adelante.

Lema 3.2. *La sucesión ω satisface las siguientes propiedades:*

i. Es estrictamente creciente.

ii. Es acotada superiormente.

iii. Su límite es $1 + \sqrt{2}$.

Demostración. Veamos que es válida cada una de las propiedades enunciadas.

i. Para comprobar que ω es estrictamente creciente, basta con verificar que $\forall n \in \mathbb{N}$ vale que $\omega_n - \omega_{n-1} > 0$. Si $n = 1$, tenemos que

$$\omega_1 - \omega_0 \;=\; \sqrt{2} + (\sqrt{2} - 1) - 1 = 2\ (\sqrt{2} - 1) > 0$$

Ahora, asumiendo como hipótesis inductiva que $\omega_{n-1} - \omega_{n-2} > 0$ para cualquier $n > 1$, podemos comprobar que

$$\begin{aligned} \omega_n - \omega_{n-1} &= \sqrt{2} + (\sqrt{2} - 1)\ \omega_{n-1} - \sqrt{2} - (\sqrt{2} - 1)\ \omega_{n-2} \\ &= (\sqrt{2} - 1)(\omega_{n-1} - \omega_{n-2}) > 0 \end{aligned}$$

[2] En el siguiente capítulo veremos que existen algoritmos online cuya competitividad contra un adversario standard coincide con esta nueva cota.

ii. Para ver que ω es acotada superiormente, veamos que $\forall n \in \mathbb{N}_0$ se cumple que $\omega_n < 1 + \sqrt{2}$. Es claro que $\omega_0 = 1 < 1 + \sqrt{2}$. Ahora, asumiendo como hipótesis inductiva que $\omega_{n-1} < 1 + \sqrt{2}$ para cualquier $n > 0$, podemos comprobar que

$$\begin{aligned} \omega_n &= \sqrt{2} + (\sqrt{2} - 1)\ \omega_{n-1} < \sqrt{2} + (\sqrt{2} - 1)(1 + \sqrt{2}) \\ &= \sqrt{2} + 2 - 1 = 1 + \sqrt{2} \end{aligned}$$

iii. Finalmente analicemos el límite de ω. Al ser ω creciente y acotada superiormente, tiene límite. Notamos su valor con c. Si aplicamos límite en la definición recursiva de ω_n (3.1.1) obtenemos

$$\begin{aligned} c &= \lim_{n\to\infty} \omega_n = \lim_{n\to\infty} \sqrt{2} + (\sqrt{2} - 1)\ \omega_{n-1} \\ &= \sqrt{2} + (\sqrt{2} - 1) \lim_{n\to\infty} \omega_{n-1} \\ &= \sqrt{2} + (\sqrt{2} - 1)\ c \end{aligned}$$

Es decir, $c = \sqrt{2} + (\sqrt{2} - 1)\ c \ \Rightarrow\ c = 1 + \sqrt{2}$.

□

Ahora que conocemos un poco más sobre la sucesión ω, estamos listos para demostrar la cota inferior contra un adversario *standard*.

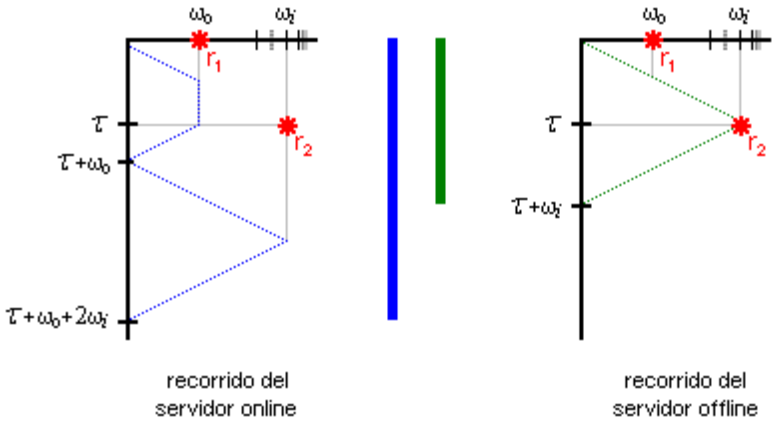
Teorema 3.3. *Existe una familia de grafos en los que la competitividad de cualquier algoritmo online determinístico* `ALG` *para la variante Homing de DOLTSP contra un adversario* standard *no es menor que* $\rho = \frac{2+\sqrt{2}}{2} \approx 1.707$.

Demostración. Consideremos una semicadena donde el vértice más cercano al origen está ubicado en la posición $\omega_0 = 1$ y además hay un vértice en cada una de las posiciones determinadas por los valores de la sucesión ω (expresión 3.1.1). Esto es posible puesto que según el Lema 3.2, la sucesión ω es estrictamente creciente. La semicadena puede contener además otros vértices, siempre que su distancia al origen no sea menor que ω_0.

En el instante 0 se presenta un primer requerimiento r_1 en la posición $\omega_0 = 1$. Sea $\tau \geq \omega_0$ el momento en que el servidor operado por `ALG` comienza el regreso hacia el origen, luego de haber servido r_1. `ALG` no puede completar el servicio antes que $\tau + \omega_0$, mientras que el adversario puede terminar en tiempo $2\omega_0 = 2$. Entonces, si $\tau + \omega_0 \geq 2\rho$ no se presenta ningún otro pedido y `ALG` no es mejor que ρ-competitivo.

En cambio, si $\omega_0 \leq \tau < 2\rho - \omega_0 = 1 + \sqrt{2}$, como el Lema 3.2 asegura que la sucesión ω es creciente y su límite es $1 + \sqrt{2}$, podemos deducir que existe $i \in \mathbb{N}$ tal que $\omega_i \leq \tau < \omega_{i+1}$ [★]. Se presenta un segundo requerimiento $r_2 = (\tau, \omega_i)$. De esta forma el servidor online no puede completar la secuencia de requerimientos antes de $\tau + \omega_0 + 2\omega_i$. Dado que $\omega_0 \leq \omega_i \leq \tau$, en el momento inicial

el adversario puede dirigir su servidor directamente a la posición ω_i (sirviendo r_1 al pasar por ω_0) y esperar hasta el momento τ a que se presente r_2. Así, el servidor offline estaría de regreso en el origen en el momento $\tau + \omega_i$.



Finalmente, la competitividad de `ALG` queda acotada por la siguiente expresión:

$$\begin{aligned}
\frac{ALG(r_1, r_2)}{OPT(r_1, r_2)} &\geq \frac{\tau + \omega_0 + 2\omega_i}{\tau + \omega_i} = 1 + \frac{\omega_0 + \omega_i}{\tau + \omega_i} >^{\star} \\
&> 1 + \frac{\omega_0 + \omega_i}{\omega_{i+1} + \omega_i} = 1 + \frac{\omega_0 + \omega_i}{\sqrt{2} + (\sqrt{2} - 1)\ \omega_i + \omega_i} = \\
&= 1 + \frac{\omega_0 + \omega_i}{\sqrt{2} + \omega_i\sqrt{2}} = 1 + \frac{1 + \omega_i}{(1 + \omega_i)\ \sqrt{2}} = 1 + \frac{1}{\sqrt{2}} = \frac{2 + \sqrt{2}}{2} = \rho
\end{aligned}$$

□

3.2 La variante Nomadic

En cualquier variante de DOLTSP, el algoritmo online desconoce cuándo y dónde se presentarán los próximos requerimientos; incluso desconoce si aparecerá o no algún otro pedido que no se haya presentado hasta el momento. Aún así, en la variante Homing las estrategias cuentan con un dato seguro sobre el futuro: saben que, independientemente de la secuencia de requerimientos, finalmente el servidor deberá regresar al origen. En contraste con esta pequeña certeza conocida desde el comienzo, en la variante Nomadic las estrategias no pueden dar por seguro ningún destino final para el servidor. Esta sutil diferencia hace que las cotas inferiores que mostramos en esta sección tengan un valor ligeramente más elevado que las correspondientes según el tipo de adversario, expuestas en la sección anterior.

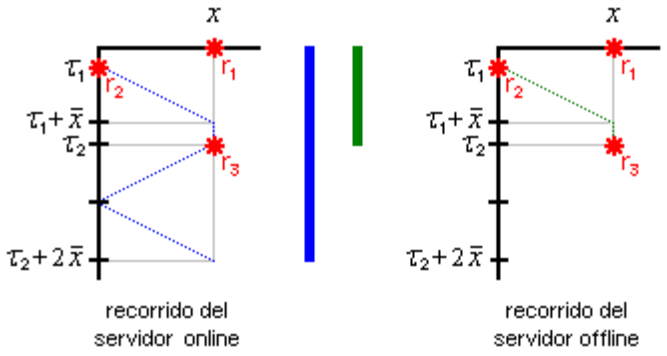
El siguiente resultado plantea una cota inferior para Nomadic DOLTSP contra un adversario *fair*. Al igual que en el Teorema 3.1, esta cota vale para cualquier grafo no trivial.

Teorema 3.4. *Sea* `ALG` *un algoritmo online determinístico cualquiera para la variante Nomadic de DOLTSP. Entonces, la competitividad de* `ALG` *contra un adversario* fair *no es menor que* $\rho = \frac{1+\sqrt[3]{19-3\sqrt{33}}+\sqrt[3]{19+3\sqrt{33}}}{3} \approx 1.839$.

Demostración. Se presenta $r_1 = (0, x)$, en un grafo donde x es el vértice más cercano al origen. Sea $\tau_1 \geq 0$ el momento en que el servidor operado por `ALG` sale del origen para servir r_1. Éste no podría terminar antes que $\tau_1 + \bar{x}$, mientras que el adversario podría servir r_1 en tiempo $\bar{x}$. Si ocurriera que $\tau_1 + \bar{x} \geq \rho\bar{x}$, entonces no se presentaría ningún otro requerimiento y `ALG` no sería mejor que ρ-competitivo.

Asumiremos entonces que $\tau_1 + \bar{x} < \rho\bar{x}$ [★]. En este caso, se presenta un nuevo requerimiento $r_2 = (\tau_1, o)$. Sea $\tau_2 \geq \tau_1 + \bar{x}$ el momento en que el servidor operado por `ALG` parte hacia el origen para servir r_2 luego de haber servido r_1. Nuevamente, si sucediera que $\tau_2 + \bar{x} \geq \rho(\tau_1 + \bar{x})$, entonces no se presentaría ningún otro requerimiento y `ALG` no sería mejor que ρ-competitivo, puesto que el adversario en este caso podría servir primero r_2 en el origen y luego dirigirse a x para servir r_1, terminando en tiempo $\tau_1 + \bar{x}$ lo que al servidor online le llevaría al menos $\tau_2 + \bar{x}$.

Supondremos, por lo tanto, que $\tau_2 + \bar{x} < \rho(\tau_1 + \bar{x})$ [♣]. Dadas estas condiciones, se presenta un tercer requerimiento $r_3 = (\tau_2, x)$. El servidor operado por `ALG` no podrá terminar su recorrido antes de $\tau_2 + 2\bar{x}$, mientras que el adversario puede esperar en el origen hasta el momento τ_1 para servir primero r_2 y luego dirigirse hasta x para servir r_1, finalizando en tiempo τ_2, cuando se presenta r_3.



Es decir que podemos acotar la competitividad de `ALG` de la siguiente forma:

$$\begin{aligned}\frac{ALG(r_1,r_2,r_3)}{OPT(r_1,r_2,r_3)} &\geq \frac{\tau_2+2\bar{x}}{\tau_2} = 1+\frac{2\bar{x}}{\tau_2} >^{\clubsuit} 1+\frac{2\bar{x}}{\rho(\tau_1+\bar{x})-\bar{x}} >^{\bigstar} 1+\frac{2\bar{x}}{\rho^2\bar{x}-\bar{x}} = \\ &= 1+\frac{2}{\rho^2-1} = \rho\end{aligned}$$

□

Finalmente mostraremos que en la variante Nomadic, ninguna estrategia online que compita contra un adversario *standard* puede ser mejor que 2-competitiva. Por las mismas razones expuestas en la sección anterior, al tratarse de un adversario que puede sacar mejor provecho de su conocimiento, es natural que esta cota sea algo superior a la que mostramos para la misma variante, contra un adversario *fair* (≈ 1.839). Pero aquí también, tal beneficio depende de la existencia de ciertos vértices para que el servidor offline pueda adelantarse al online.

Expondremos dos ejemplos donde esta ventaja se hace efectiva. El primero de ellos sucede en una cadena con al menos dos vértices en posiciones simétricas respecto del origen. Pero, por la forma en que está construido, no es aplicable en una semicadena.

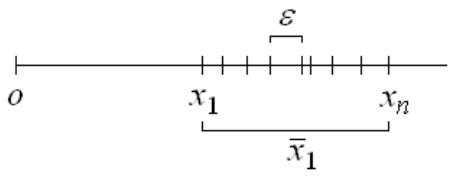
Teorema 3.5. *Existe una familia de grafos en los que la competitividad de cualquier algoritmo online determinístico* `ALG` *para la variante Nomadic de DOLTSP contra un adversario* standard *no es menor que* $\rho = 2$.

Demostración. Consideremos una cadena con dos vértices x e y además del origen, donde $x = -y$. Podemos asumir, sin pérdida de generalidad, que en el momento $\bar{x}$ el servidor comandado por `ALG` se encuentra entre el origen y el vértice y. Entonces se presenta un único requerimiento $r = (\bar{x}, x)$, que el servidor offline puede atender en tiempo $\bar{x}$, mientras que el costo del servidor online no sería menor que $2\bar{x}$. □

El segundo ejemplo es posible en una semicadena, pero su demostración es bastante más compleja.

Teorema 3.6. *Existe una familia de grafos en los que la competitividad de cualquier algoritmo online determinístico* `ALG` *para la variante Nomadic de DOLTSP contra un adversario* standard *no es menor que* $\rho = 2 - \xi$, *donde* ξ *es un número real positivo arbitrariamente pequeño.*

Demostración. Consideremos una semicadena que incluye a los vértices $x_1 < \ldots < x_n$, donde x_1 es el vértice más cercano al origen y la distancia entre x_1 y x_n es exactamente $\bar{x}_1$. Asumamos además que n es un número suficientemente grande y que existe una constante ε tal que para cualquier $i \in \{2, \ldots, n\}$ sucede que $d(x_{i-1}, x_i) \leq \varepsilon$.



En tiempo 0 se presenta un requerimiento r_1 en x_1. Sea $\tau_1 \geq 0$ el momento en que el servidor online abandona el origen para servirlo. Si $\tau_1 \geq \bar{x}_1$, no se presentan más requerimientos y `ALG` no es

mejor que 2-competitivo, puesto que no alcanza a servir r_1 antes de $\tau_1 + \bar{x}_1 \geq 2\bar{x}_1$, mientras que el adversario puede servir ese requerimiento en tiempo $\bar{x}_1$. Asumiremos por lo tanto que $\tau_1 < \bar{x}_1$ [★].

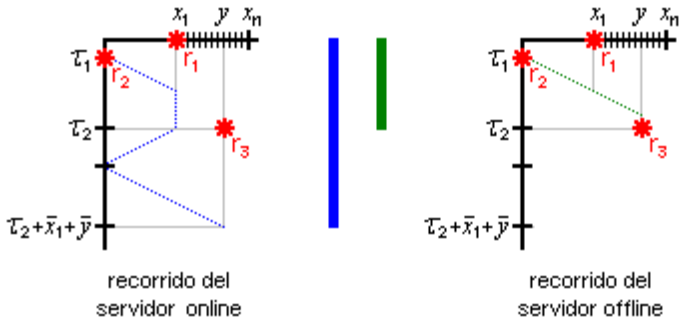
El adversario presenta un nuevo requerimiento $r_2 = (\tau_1, o)$. Llamemos $\tau_2 \geq \tau_1 + \bar{x}_1$ al momento en que el servidor operado por `ALG` comienza el trayecto hacia el origen para servir r_2 luego de haber servido r_1. El servidor online no logra servir r_2 antes de $\tau_2 + \bar{x}_1$, mientras que el adversario offline puede servir primero r_2 y luego r_1, completando el servicio en tiempo $\tau_1 + \bar{x}_1$. Entonces, si ocurriera que $\tau_2 + \bar{x}_1 \geq 2(\tau_1 + \bar{x}_1)$, no se presentaría ningún otro requerimiento, dado que los ya presentados son suficientes para mostrar que `ALG` no puede mejorar una competitividad de 2.

Queda por analizar el caso en que $\tau_2 + \bar{x}_1 < 2(\tau_1 + \bar{x}_1)$, lo que equivale a $\tau_2 - \tau_1 < \tau_1 + \bar{x}_1$. Con este fin, notemos que de la definición de τ_2 surge que si en el momento τ_1 el servidor comandado por el adversario atiende r_2 y comienza a alejarse del origen, en el momento τ_2 puede estar en x_1 o aún más a la derecha. Llamemos y al vértice más alejado del origen al que puede llegar el servidor offline en el momento τ_2, si parte del origen en el momento τ_1. Dado que el servidor offline se desplaza a velocidad unitaria, se cumple que $\bar{y} \leq \tau_2 - \tau_1$, y por lo tanto

$$\bar{x}_1 \leq \bar{y} \leq \tau_2 - \tau_1 < \tau_1 + \bar{x}_1 <^{\bigstar} 2\bar{x}_1 = \bar{x}_n$$

Como y está entre x_1 y x_n, si fuera $\tau_2 - \tau_1 \geq \bar{y} + \varepsilon$, entonces existiría un vértice a la derecha de y al que podría llegar el servidor offline en el momento τ_2, cosa que contradice la definición de y. Entonces, podemos concluir que $\tau_2 - \tau_1 < \bar{y} + \varepsilon$ [♣].

En el momento τ_2 se presenta un tercer requerimiento $r_3 = (\tau_2, y)$. Para servir los tres requerimientos presentados, el adversario podría permanecer en el origen hasta el momento τ_1, luego de servir r_2 partir hacia x_1 para servir r_1 y sin detenerse continuar hasta y. Allí esperaría hasta el momento τ_2 para servir r_3, completando el servicio. El servidor online, en cambio, está obligado a continuar el trayecto de x_1 al origen comenzado en el momento τ_2 para luego dirigirse a servir r_3, no pudiendo completar el servicio antes que $\tau_2 + \bar{x}_1 + \bar{y}$.



En consecuencia, podemos acotar la competitividad de ALG con la siguiente expresión:

$$\begin{aligned}\frac{ALG(r_1,r_2,r_3)}{OPT(r_1,r_2,r_3)} &\geq \frac{\tau_2+\bar{x}_1+\bar{y}}{\tau_2} = 1+\frac{\bar{x}_1+\bar{y}}{\tau_2} >^{\clubsuit} 1+\frac{\bar{x}_1+\bar{y}}{\tau_1+\bar{y}+\varepsilon} >^{\bigstar} 1+\frac{\bar{x}_1+\bar{y}}{\bar{x}_1+\bar{y}+\varepsilon} = \\ &= 2-\frac{\varepsilon}{\bar{x}_1+\bar{y}+\varepsilon} > 2-\frac{\varepsilon}{2\bar{x}_1}\end{aligned}$$

Si el valor de ε es suficientemente pequeño en relación a $\bar{x}_1$, la expresión anterior puede estar tan cerca de 2 como se desee. □

A lo largo de este capítulo hemos encontrado cuatro cotas inferiores para las variantes Homing y Nomadic de DOLTSP, contra adversarios *fair* y *standard*. En el siguiente cuadro mostramos estos resultados y los comparamos con cotas inferiores para otros dos problemas de ruteo online relacionados: OLTSP y OLDARP en espacios métricos continuos [Lip03]. Recordemos que OLTSP es el mismo problema que el que analizamos en este trabajo, pero permitiéndole al servidor cambiar de dirección en cualquier momento. En OLDARP los requerimientos no son solicitudes de visita, sino solicitudes de traslado de objetos de un punto a otro del espacio.

	Homing		Nomadic	
	fair	*standard*	*fair*	*standard*
OLTSP	≈ 1.281	1.5	≈ 1.60	≈ 1.63
DOLTSP	≈ 1.618	≈ 1.707	≈ 1.839	2
OLDARP	≈ 1.618	≈ 1.707	≈ 1.897	≈ 1.897

OLTSP y DOLTSP en su variante Homing son problemas cerrados, lo mismo que la variante Nomadic de DOLTSP contra un adversario *standard*. Esto es, existen algoritmos conocidos cuya competitividad coincide con la correspondiente cota inferior.[3] Para el resto de las variantes, el problema permanece abierto. Esto significa que aún existe la posibilidad de encontrar cotas inferiores más elevadas o de desarrollar algoritmos más eficientes en el peor caso.

Si comparamos los valores obtenidos para OLTSP y DOLTSP, podemos notar que el problema discreto tiene asociadas cotas inferiores más elevadas que el problema continuo. Esto muestra la dificultad adicional que supone el hecho de que el servidor no tenga la libertad de cambiar de dirección en cualquier momento.

En realidad, las cotas de DOLTSP se parecen más a las correspondientes a OLDARP. Esta llamativa semejanza se debe a una similitud en las restricciones que tiene el servidor en ambos problemas. Si bien OLDARP está definido sobre un espacio continuo, una vez que el servidor levanta un objeto en un punto, está obligado a trasladarlo hacia el punto de destino y hasta entonces no le es permitido levantar otros objetos. Esta restricción se asemeja a la que caracteriza a DOLTSP, donde el servidor no puede cambiar de dirección hasta llegar al próximo vértice.

[3] Pueden encontrarse los algoritmos que cierran la variante Homing de OLTSP en [Lip03] y los que cierran los distintos casos de DOLTSP en el Capítulo 4 de este trabajo.

Capítulo 4

Estrategias

En este capítulo proponemos algunas estrategias online que pueden utilizarse para DOLTSP sobre cadenas. Las primeras dos que planteamos son muy naturales y válidas para cadenas de cualquier tipo (incluidas las semicadenas). Ambas pertenecen a la categoría de los *algoritmos entusiastas* (*Zealous algorithms* [Kru01]) que tienen la particularidad de no dejar quieto al servidor mientras haya trabajo por realizar. En ambos casos, se trata de algoritmos online determinísticos 2-competitivos para la variante Homing y 3-competitivos para la variante Nomadic, tanto contra un adversario *fair* como contra uno *standard*. Veremos que estas competitividades son las mejores posibles para la categoría de algoritmos entusiastas. La primera de estas estrategias, que denominamos `Zig-Zag`, consiste básicamente en recorrer el espacio de un extremo a otro mientras queden requerimientos pendientes. La segunda, que llamamos `Replan`, es una adaptación de una conocida estrategia homónima que consiste en ajustar la solución cada vez que se presenta un nuevo requerimiento.

Luego definimos otros dos algoritmos que denominamos *cautelosos* debido a que evitan tomar decisiones apresuradas que potencialmente impliquen un mayor costo global. Si bien ambos podrían definirse para cualquier tipo de cadena, analizamos su competitividad sólo en semicadenas. El primer algoritmo, definido para la variante Homing, se llama `Wait Before Return` (`WBR`) y consiste en alejar al servidor del origen todo lo necesario para servir al requerimiento pendiente más lejano y esperar allí **antes de regresar**. Este algoritmo es una variación de `Wait Smartly` (`WS`), presentado en [BKdPS00]. `WBR` tiene una sutil diferencia respecto a `WS` relacionada con la condición discreta del espacio métrico. Cuando el servidor se encuentra regresando, si se presenta un requerimiento a su derecha, `WS` cambia de dirección para atenderlo mientras que `WBR` no lo hace sino hasta llegar al origen. `WBR` alcanza la mejor competitividad posible tanto contra un adversario *fair* (≈ 1.618) como contra uno *standard* (≈ 1.707).

La segunda estrategia cautelosa se llama `Wait Before Begin` (`WBB`) y en la variante Nomadic es 2-competitiva contra ambos tipos de adversario, alcanzando la mejor competitividad posible contra un adversario *standard*. En este caso, el servidor espera en la posición donde se encuentre **antes de realizar cada movimiento**. Una idea similar fue aplicada previamente para la estrategia `Wait`

`and Anticipate` (`WA`), presentada en [Lip03], aunque el cálculo de la espera y del recorrido a seguir en cada momento difieren sustancialmente de `WBB`.

Las estrategias *cautelosas* buscan demorar la decisión de comenzar ciertos movimientos. De esta forma aumentan sus posibilidades de modificar el recorrido planificado, si se presentara un nuevo requerimiento durante la espera. Esta actitud precavida tiene más valor cuando el espacio métrico es discreto. Aquí, comenzar un movimiento representa una apuesta de mayor riesgo que en la versión continua del problema, puesto que la capacidad de reacción ya no es inmediata: una vez que el servidor abandona un vértice, no puede modificar su dirección hasta llegar al vértice destino.

En la última sección de este capítulo introducimos algunas estrategias adicionales, con las que no buscamos alcanzar ninguna competitividad sino reflejar algunas ideas intuitivas que pueden aplicarse para DOLTSP. Logramos así una variedad mayor de algoritmos, enriqueciendo el análisis empírico que presentaremos en los capítulos siguientes.

4.1 Algoritmos entusiastas

Los algoritmos entusiastas ya han sido estudiados en la literatura. La idea que los caracteriza es muy simple: siempre que haya trabajo pendiente, hacerlo cuanto antes. Intuitivamente, el servidor comandado por un algoritmo entusiasta no debería permanecer pasivo mientras haya requerimientos pendientes. Más aún: debería moverse directamente hacia la posición de alguno de ellos sin desviarse en ningún momento. Traducir esta idea intuitiva en una definición rigurosa requiere cierto cuidado.

Definición 4.1. Un algoritmo online para DOLTSP se dice *entusiasta* si, cuando el servidor está en un vértice v, satisface las siguientes condiciones:

1. Si hay pedidos pendientes, se dirige a servir uno de ellos por el camino más corto posible.
2. Si no hay pedidos pendientes y se trata de la variante Homing, se dirige hacia el origen por el camino más corto posible.
3. Si llega a v porque estaba dirigiéndose a un vértice $w \neq v$, sólo altera su ruta si mientras transitaba el último eje se presentó un nuevo requerimiento.

Los siguientes teoremas muestran que ningún algoritmo online entusiasta puede ser mejor que 2-competitivo en Homing DOLTSP ni mejor que 3-competitivo en Nomadic DOLTSP. Esto vale tanto contra un adversario *fair* como contra uno *standard* (tomando cualquier grafo no trivial). Notemos la diferencia que existe respecto a las cotas inferiores para un algoritmo general, que, como vimos en el capítulo anterior, son cercanas a 1.6 para la variante Homing y a 1.8 para la variante Nomadic.

Teorema 4.2. *Sea* `ALG` *un algoritmo online entusiasta para Homing DOLTSP. Entonces, la competitividad de* `ALG` *contra un adversario* fair *o* standard *no es menor que* 2*.*

Demostración. En el momento 0 se presenta el requerimiento r_1 en el vértice más cercano al origen, que notamos con x. Por tratarse de un algoritmo entusiasta, el servidor comandado por `ALG` parte hacia allí inmediatamente. Una vez servido r_1, no quedan más requerimientos pendientes y por lo tanto el servidor comienza sin demorar su regreso al origen. En ese momento, se presenta un segundo requerimiento r_2 en el mismo vértice que el anterior.

El adversario offline puede atender ambos requerimientos yendo una única vez hasta x y regresando en tiempo $2\bar{x}$. Por su parte, el servidor online en el momento $2\bar{x}$ habrá llegado al origen pero aún tendrá pendiente r_2. Es decir que no terminará el servicio antes que $4\bar{x}$ y por lo tanto la competitividad de `ALG` es al menos 2. □

Teorema 4.3. *Sea* `ALG` *un algoritmo online entusiasta para Nomadic DOLTSP. Entonces, la competitividad de* `ALG` *contra un adversario* fair *o* standard *no es menor que* 3*.*

Demostración. En el momento 0 se presenta el requerimiento r_1 en el vértice más cercano al origen, que notamos con x. Por tratarse de un algoritmo entusiasta, el servidor comandado por `ALG` parte hacia allí inmediatamente. Apenas comienza a moverse, se presenta un segundo requerimiento r_2 en el origen.

Una vez servido r_1, queda pendiente r_2 y por lo tanto el servidor comienza a regresar en el momento $\bar{x}$. Entonces se presenta un tercer requerimiento $r_3 = (\bar{x}, x)$ que el servidor online podrá atender recién en el momento $3\bar{x}$, puesto que deberá completar el trayecto al origen y luego regresar al vértice x.

La misma secuencia de requerimientos puede ser completada por un servidor offline en el momento $\bar{x}$, si atiende primero r_2 en el origen y luego se dirige a x para servir r_1 y r_3. Es decir que la competitividad de `ALG` no es menor que 3. □

4.1.1 Zig-Zag y Replan

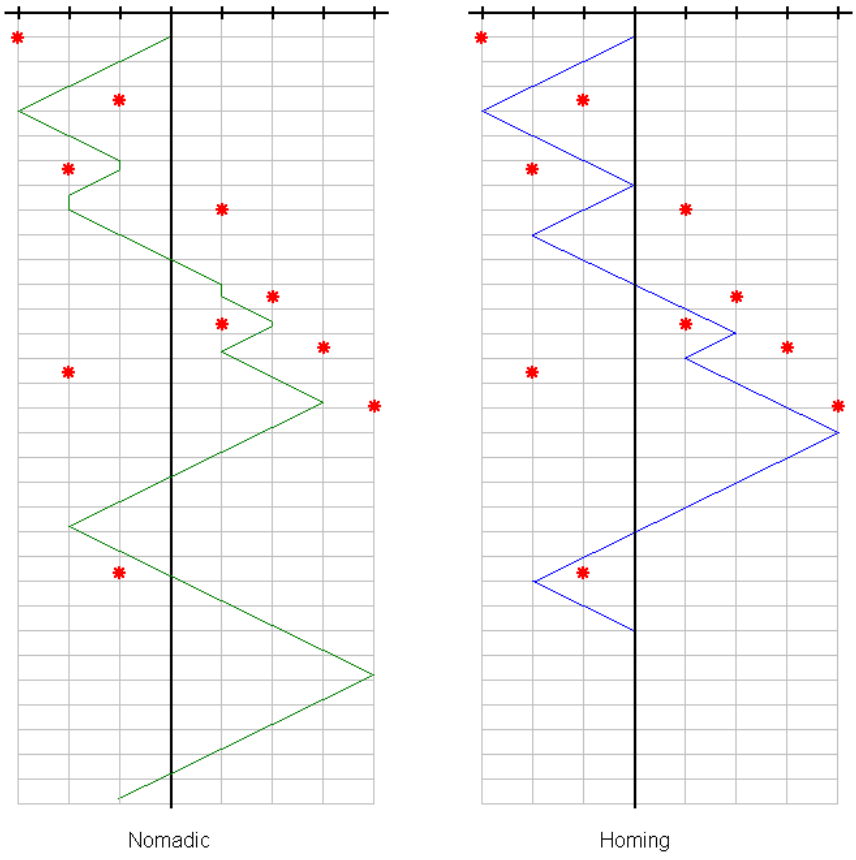
A continuación proponemos dos estrategias entusiastas muy naturales que alcanzan la mejor competitividad posible dentro de su categoría para las variantes Homing y Nomadic de DOLTSP en una cadena cualquiera.

Estrategia Zig-Zag (`ZZG`)

Repetir el siguiente procedimiento:

1. Mientras queden requerimientos pendientes a la izquierda del servidor, servirlos.
2. Mientras queden requerimientos pendientes a la derecha del servidor, servirlos.
3. En la variante Homing, mientras no queden requerimientos pendientes y el servidor esté fuera del origen, regresar.

Notemos que `Zig-Zag` consiste en realizar sucesivos tramos a izquierda y derecha hasta atender a todos los requerimientos y, si se trata de la variante Homing, regresar al origen. Llamamos *tramo* a una sucesión de movimientos en un mismo sentido, sin que el servidor se detenga.



Aquí podemos apreciar los recorridos generados por `Zig-Zag` en las variantes Nomadic y Homing de DOLTSP. La secuencia de requerimientos es la misma en ambos casos.

Teorema 4.4. *`Zig-Zag` es una estrategia 2-competitiva para la variante Homing de DOLTSP en cualquier cadena, contra un adversario* fair *o* standard

Demostración. Sea $\sigma = r_1, \ldots, r_n$ una secuencia de requerimientos cualquiera. Si σ no incluye requerimientos fuera del origen, entonces el servidor comandado por `Zig-Zag` no se mueve en ningún momento y el costo de su recorrido coincide trivialmente con el costo óptimo. Caso contrario, la estrategia `Zig-Zag` consiste en desplazar el servidor en sucesivos tramos a izquierda y derecha hasta servir todos los requerimientos y regresar finalmente al origen, ya que estamos en la variante Homing.

Tomemos el último retorno al origen y llamemos x al vértice donde comenzó ese tramo. Si hubiera requerimientos que se presentan en el origen una vez finalizado este tramo, nuevamente el costo de `Zig-Zag` coincidiría trivialmente con el costo óptimo. Consideraremos, por lo tanto, que el último requerimiento de σ se presenta antes de que el servidor llegue por última vez al origen.

Es claro que debió presentarse algún requerimiento en x para que el servidor comenzara su último tramo desde allí. Llamemos $r_i = (t_i, x)$ al último requerimiento que se presentó en ese vértice. Podemos asumir, sin pérdida de generalidad, que $x < 0$. La situación opuesta nos llevaría a un análisis equivalente al que mostramos a continuación, pero sustituyendo entre sí los términos *izquierda* y *derecha*.

1. Podría suceder que en algún momento posterior a t_i el servidor estuviera alejándose del origen hacia la derecha. En tal caso, ese tramo finalizará al servir un requerimiento $r_j = (t_j, y)$ y entonces el servidor se dirigiría hacia x para atender r_i y regresar finalmente al origen. Para que suceda esto, en el momento t_i el servidor online debe estar moviéndose hacia la derecha y en una posición también a la derecha de x (de otro modo, r_i sería atendido antes que r_j). Es decir que podemos acotar el costo del algoritmo online por $ZZG(\sigma) < t_i + \bar{x} + 2\bar{x} + 2\bar{y}$. Como el costo óptimo no es menor que $t_i + \bar{x}$ ni que $2\bar{x} + 2\bar{y}$, tenemos
$$\frac{ZZG(\sigma)}{OPT(\sigma)} < \frac{t_i + \bar{x}}{OPT(\sigma)} + \frac{2\bar{x} + 2\bar{y}}{OPT(\sigma)} \leq 2$$

2. La situación contraria a la analizada en el punto anterior es que en ningún momento posterior a t_i el servidor estuviera alejándose del origen hacia la derecha. En este caso, el costo de `Zig-Zag` no sería mayor que lo que le lleva desplazar el servidor hasta el origen para luego servir r_i y regresar finalmente al origen. El servicio finalizaría a lo sumo en el momento $t_i + \bar{s}_{t_i} + 2\bar{x}$. Puesto que $\bar{s}_{t_i} \leq t_i$ (el servidor se desplaza a velocidad unitaria), la expresión anterior no es mayor que $2t_i + 2\bar{x}$. Notando nuevamente que el costo óptimo offline es al menos $t_i + \bar{x}$, resulta
$$\frac{ZZG(\sigma)}{OPT(\sigma)} \leq \frac{2t_i + 2\bar{x}}{t_i + \bar{x}} = 2$$

□

Teorema 4.5. *`Zig-Zag` es una estrategia 3-competitiva para la variante Nomadic de DOLTSP en cualquier cadena, contra un adversario* fair *o* standard.

Demostración. Sea $\sigma = r_1, \ldots, r_n$ una secuencia de requerimientos cualquiera. Si σ no incluye requerimientos fuera del origen, entonces el servidor comandado por `Zig-Zag` no se mueve en ningún momento y el costo de su recorrido coincide trivialmente con el costo óptimo. Caso contrario, sea $r = (t, x) \in \sigma$ el último requerimiento atendido por el servidor.

1. Si a partir del momento t el servidor comandado por `Zig-Zag` nunca se aleja de x, entonces el costo online es a lo sumo $t + \bar{s}_t + \bar{x}$. Como el servidor se desplaza a velocidad unitaria, sabemos que $\bar{s}_t \leq t$, y por lo tanto $ZZG(\sigma) \leq 2t + \bar{x}$. Por otro lado, el costo óptimo offline no es menor que t ni que $\bar{x}$. Así obtenemos que
$$\frac{ZZG(\sigma)}{OPT(\sigma)} \leq \frac{2t}{OPT(\sigma)} + \frac{\bar{x}}{OPT(\sigma)} \leq 3$$
2. Por el contrario, podría ocurrir que una vez presentado r, en algún momento el servidor se aleje de x. Debido a la naturaleza de `Zig-Zag`, esto significa que ya en el momento t el servidor se estaba alejando de x. Ese tramo finalizará al servir un requerimiento en un vértice y, desde donde el servidor se dirigirá hacia x para atender a r, llegando no más tarde que $t + 2d(x, y)$. El costo óptimo offline no es menor que t ni que $d(x, y)$, de modo que
$$\frac{ZZG(\sigma)}{OPT(\sigma)} \leq \frac{t}{OPT(\sigma)} + \frac{2d(x, y)}{OPT(\sigma)} \leq 3$$

□

Estrategia Replan (`REP`)

Se trata de un algoritmo muy estudiado en distintos problemas de optimización online que consiste básicamente en ajustar la solución cada vez que se presenta un nuevo requerimiento. En el caso de DOLTSP, con cada replanificación se computa un nuevo recorrido que le permite al servidor online completar el servicio en el menor tiempo posible (recorriendo la menor distancia). Como los movimientos no pueden ser interrumpidos, si se presenta un requerimiento mientras el servidor está atravesando un eje, debemos postergar la replanificación hasta llegar al próximo vértice. Para simplificar el cálculo del recorrido de menor distancia cuando la instancia del problema está definida sobre una cadena, el concepto de *extremo* nos será de gran ayuda.

Definición 4.6. Dada una instancia de DOLTSP sobre una cadena, consideremos el conjunto de vértices que le resta visitar al servidor online para finalizar el servicio, de acuerdo a los requerimientos presentados hasta un momento arbitrario. En la variante Nomadic, este conjunto está compuesto por los vértices donde quedan requerimientos pendientes. En la variante Homing incluye, además, al origen. De todos los vértices que componen dicho conjunto, llamamos *extremo izquierdo* al que está más a la izquierda y *extremo derecho* al que está más a la derecha.

De acuerdo a esta definición, en la variante Homing el extremo izquierdo está en la semicadena izquierda y el extremo derecho está en la semicadena derecha (aunque uno o ambos extremos pueden ser el origen). Esto implica que siempre hay al menos un extremo en la misma semicadena que el servidor. Teniendo en cuenta esto, el recorrido más corto para esta variante puede ser descripto de la siguiente manera:

1. Si el servidor está en el origen, ir a un extremo cualquiera. Si está fuera, ir al extremo más alejado del origen que esté en la misma semicadena que el servidor.
2. Ir al otro extremo.
3. Ir al origen.

En la variante Nomadic no podemos asumir ninguna relación entre las posiciones de los extremos y del origen. El recorrido más corto para esta variante consiste en:

1. Ir al extremo más cercano al servidor.
2. Ir al otro extremo.

Claro está, en cualquier momento del recorrido se puede presentar un nuevo requerimiento. Entonces, tan pronto como el servidor alcance un vértice, se vuelve a calcular el recorrido de menor longitud para finalizar el servicio.

Los siguientes resultados muestran que, al igual que `Zig-Zag`, `Replan` alcanza la mejor competitividad posible para estrategias entusiastas.

Teorema 4.7. *`Replan` es una estrategia 2-competitiva para la variante Homing de DOLTSP en cualquier cadena, contra un adversario* fair *o* standard*.*

Demostración. Sea σ una secuencia de requerimientos cualquiera. Llamemos L al vértice que está más a la izquierda y R al que está más a la derecha entre los vértices que debe visitar el servidor para atender a todos los requerimientos de σ, incluyendo al origen. Es claro que en todo momento ambos extremos están entre L y R. Más precisamente, el extremo izquierdo está entre L y el origen, y el extremo derecho está entre el origen y R. También el servidor comandado por `Replan` está siempre entre L y R, puesto que éste no se mueve si no es para servir un requerimiento. Consideremos ahora dos situaciones posibles, notando con T al momento en que se presenta el último entre todos los requerimientos de σ:

1. En el momento T el servidor está sobre un vértice o atravesando un eje que lo aleja del origen. Para completar su trabajo, el servidor primero se dirigirá hacia un extremo, luego hacia el otro,

y finalmente regresará al origen. Tal recorrido finalizará a lo sumo en el momento $T+2\bar{L}+2\bar{R}$. Puesto que T y $2\bar{L}+2\bar{R}$ son cotas inferiores para el costo óptimo, tenemos

$$\frac{REP(\sigma)}{OPT(\sigma)} \leq \frac{T}{OPT(\sigma)} + \frac{2\bar{L}+2\bar{R}}{OPT(\sigma)} \leq 2$$

2. En el momento T el servidor está atravesando un eje que lo acerca al origen. Llamemos $v \neq o$ al vértice donde comienza ese movimiento. Por lo dicho anteriormente, sabemos que v está entre L y R. Consideremos dos subcasos:

   (a) Una vez que el servidor abandona v, no se presenta ningún requerimiento en v ni en ningún otro vértice de la misma semicadena que esté más alejado del origen que v. Al igual que en la situación anterior, el servidor irá primero a un extremo, luego al otro y finalmente al origen, terminando su tarea a lo sumo en el momento $T+2\bar{L}+2\bar{R}$. Como vimos en la situación anterior, esta expresión es como máximo 2 veces el costo óptimo.

   (b) Después de abandonar v, se presenta al menos un requerimiento en v o en otro vértice de la misma semicadena, pero más alejado del origen que v. Llamemos $r = (t, x)$ a uno de esos requerimientos. Es claro que la longitud del eje que está atravesando el servidor es menor o igual que $\bar{x}$ y, como dijimos, ese movimiento comenzó antes del momento t. Por lo tanto, antes del momento $t+\bar{x}$ el servidor llega a un nuevo vértice y puede modificar su recorrido para atender a todos los requerimientos pendientes, terminando a más tardar en el momento $t+\bar{x}+2\bar{L}+2\bar{R}$. El costo óptimo no es menor que $t+\bar{x}$ ni que $2\bar{L}+2\bar{R}$ y, por lo tanto,

$$\frac{REP(\sigma)}{OPT(\sigma)} \leq \frac{t+\bar{x}}{OPT(\sigma)} + \frac{2\bar{L}+2\bar{R}}{OPT(\sigma)} \leq 2$$

□

Teorema 4.8. *`Replan` es una estrategia 3-competitiva para la variante Nomadic de DOLTSP en cualquier cadena, contra un adversario* fair *o* standard*.*

Demostración. Sea σ una secuencia de requerimientos cualquiera. Al igual que en la demostración anterior, llamemos L al vértice que está más a la izquierda y R al que está más a la derecha entre los vértices que debe visitar el servidor para atender a todos los requerimientos de σ, incluyendo al origen. Notemos que también en la variante Nomadic el servidor permanece durante todo su recorrido entre L y R. Sea T el momento en que se presenta el último requerimiento de σ. Considerando los extremos vigentes en el momento T, para completar su trabajo el servidor debe realizar los siguientes trayectos:

1. Si está atravesando un eje, llegar hasta el próximo vértice para poder replanificar su recorrido.
2. Luego, ir desde ese vértice al extremo más cercano.
3. Por último, viajar desde allí hasta el otro extremo.

Si en el momento T el servidor ya está en un vértice, el primer trayecto no tiene costo. En cualquier caso, la longitud que suman los dos primeros trayectos no es mayor que la distancia entre L y R. Y la longitud del tercer trayecto tampoco.

En conclusión, el costo total del recorrido es a lo sumo $T + 2d(L, R)$. Considerando que el costo óptimo no es inferior a T ni a $d(L, R)$, podemos decir que

$$\frac{REP(\sigma)}{OPT(\sigma)} \leq \frac{T}{OPT(\sigma)} + \frac{2d(L,R)}{OPT(\sigma)} \leq 3$$

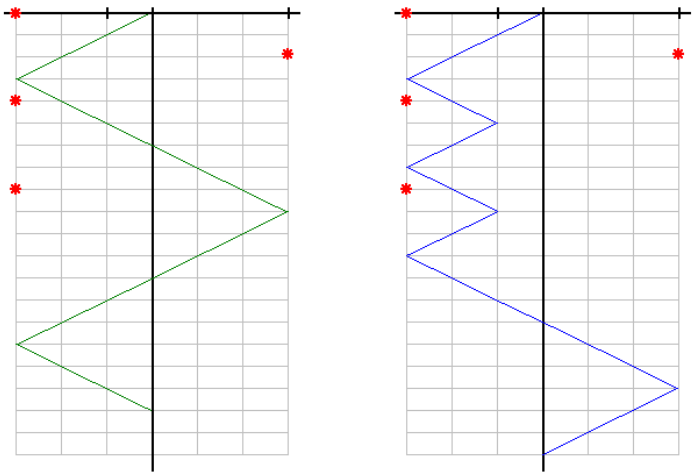
□

De esta forma hemos mostrado que tanto `Zig-Zag` como `Replan` tienen la mejor competitividad que un algoritmo entusiasta puede tener, para las variantes Homing y Nomadic de DOLTSP en cualquier tipo de cadenas.

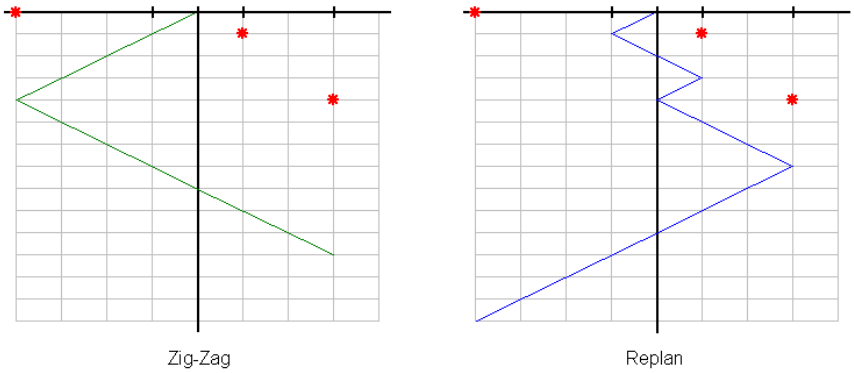
Una conclusión interesante que podemos obtener de estos resultados es que, a diferencia del caso general, en la categoría de algoritmos entusiastas, considerar un adversario *fair* o uno *standard* no influye en ninguna medida sobre el análisis de competitividad para DOLTSP en cadenas. Esto es, en ambas variantes de DOLTSP, las cotas inferior y superior coinciden y son independientes del modelo de adversario.

Un análisis superficial del comportamiento de estas estrategias sugiere que para cada instancia de DOLTSP, el recorrido generado por `Replan` debería tener menor costo que el que se obtiene con `Zig-Zag`, puesto que la primera estrategia ajusta su solución con más frecuencia que la segunda y en cada replanificación calcula el recorrido restante de menor costo. Sin embargo, esta intuición es falsa. Veamos dos ejemplos, uno para la variante Homing y otro para la variante Nomadic, en los que `Zig-Zag` obtiene una solución de menor costo que la de `Replan`.

Homing



Nomadic



Ejemplos que contradicen la idea intuitiva de que
`Replan` produce siempre mejores soluciones que `Zig-Zag`.

Por otro lado, considerando la rigidez de la categoría de algoritmos entusiastas y la simplicidad de `Zig-Zag` y `Replan`, no queda claro si su eficiencia se debe a una característica particular o es compartida por todas las estrategias entusiastas. En este contexto, tiene sentido la pregunta *"¿qué tan mala puede ser una estrategia entusiasta?"*. A continuación presentamos una estrategia entusiasta arbitrariamente mala que puede ser utilizada para ambas variantes de DOLTSP, tanto en cadenas completas como en semicadenas.

Estrategia Regrets

Cada vez que se presentan requerimientos cuando el servidor está detenido, atender el requerimiento más cercano al servidor. Cada vez que el servidor termina un movimiento en un vértice donde atiende un pedido y quedan otros requerimientos pendientes:

1. Si queda algún requerimiento pendiente en el sentido opuesto al que traía el servidor, atender al más cercano de ellos.
2. Si sólo quedan requerimientos pendientes en el mismo sentido que traía el servidor, atender al más cercano de ellos.

En la variante Homing, si no quedan requerimientos pendientes y el servidor está fuera del origen, regresar. Si durante ese regreso se presenta un pedido, al llegar a un vértice dirigirse a servir el requerimiento pendiente más cercano al servidor.

Veamos cómo se comporta `Regrets` sobre la siguiente instancia. Consideremos una semicadena que contiene n vértices $A = \{a_1, \ldots, a_n\}$ tales que $\bar{a}_n < \ldots < \bar{a}_1 < \varepsilon$ y otros n vértices $B = \{b_1, \ldots, b_n\}$ tales que $1 - \varepsilon < \bar{b}_1 < \ldots < \bar{b}_n \leq 1$, donde ε es un número real en el intervalo $(0, \frac{1}{2})$.

En el momento 0 se presentan n requerimientos, uno en cada vértice $b_i \in B$ y en el momento ε se presentan otros n requerimientos, uno en cada vértice $a_i \in A$. El recorrido generado por `Regrets` atiende a los requerimientos en el siguiente orden: b_1, a_1, b_2, a_2, ..., b_n, a_n, y regresa finalmente al origen si se trata de la variante Homing. Este recorrido implica viajar $2n$ veces entre uno de los vértices de A y uno de los vértices de B, con un costo total de al menos $2n(1 - 2\varepsilon)$. Si ε es suficientemente pequeño, este valor será cercano a $2n$. Por otro lado, el costo óptimo offline para servir la misma secuencia de requerimientos es menor o igual que 2.

Acabamos de verificar que `Regrets` es una estrategia entusiasta que puede generar recorridos que tienen un costo n veces mayor que el óptimo, con n arbitrariamente grande, lo que equivale a decir que `Regrets` no es competitiva. En conclusión, a pesar de que en toda estrategia entusiasta el servidor mantiene el foco constantemente en atender pedidos, no hay ninguna garantía de que el orden en que se atienden sea adecuado.

4.2 Algoritmos cautelosos

En la Sección 4.1 vimos que ningún algoritmo entusiasta puede ser mejor que 2-competitivo en la variante Homing ni mejor que 3-competitivo en la variante Nomadic. También vimos que estas

cotas inferiores son notablemente más elevadas que las correspondientes para un algoritmo general. Esta diferencia se debe principalmente a que un algoritmo entusiasta, en su apuro por atender a los requerimientos pendientes, comete errores que podrían evitarse con una visión más global de la situación.

En esta sección consideramos algoritmos online en los que el servidor puede permanecer un tiempo inmóvil, aunque queden requerimientos pendientes. Las estrategias *cautelosas* que presentamos aprovechan esta posibilidad demorando al máximo la decisión de comenzar determinados movimientos. De esta forma aumentan sus posibilidades de modificar el recorrido planificado, si se presentara un nuevo requerimiento durante la espera.

Para establecer cuánto es lo máximo que puede esperar el servidor online antes de comenzar un movimiento, debemos notar que la precaución no puede ir en contra del objetivo principal de las estrategias, que es minimizar el tiempo de finalización del servicio. En esta sección nos proponemos encontrar estrategias que funcionen bien en el peor caso; es decir, que tengan la menor competitividad posible. Entonces, el límite está determinado por el momento en que, si el servidor continuara esperando, la competitividad aumentaría.

Si conocemos la competitividad que queremos alcanzar con una estrategia `ALG`, podemos calcular ese límite en cualquier momento t, acotando el costo óptimo offline para servir los requerimientos conocidos y comparándolo contra lo que le resta recorrer al servidor online para terminar de atender a esos mismos requerimientos. Notaremos con ρ a la competitividad que queremos alcanzar y con α al tiempo que necesita el servidor online para atender los requerimientos pendientes (regresando al origen si fuera necesario). También calcularemos una cota inferior para el costo óptimo offline, es decir, un valor Z^* tal que $Z^* \leq OPT(\sigma_{\leq t})$. Sin superar la competitividad ρ, el servidor online podría quedarse inmóvil en su posición hasta el momento w, calculado como

$$w = \rho \; Z^* - \alpha \tag{4.2.1}$$

puesto que en ese caso tendríamos

$$ALG(\sigma_{\leq t}) = w + \alpha = \rho \; Z^* \leq \rho \; OPT(\sigma_{\leq t})$$

Para cada una de las estrategias cautelosas, determinaremos valores para ρ, Z^* y α de modo tal que el servidor online pueda comenzar su próximo movimiento en el momento w. Esto es, que si en un momento t calculamos una espera w, resulte $w \geq t$.

Hasta aquí hemos visto cómo calcular el momento en que debe concluir la espera. Resta determinar cuáles son aquellos movimientos que conviene postergar o, dicho de otro modo, en qué momento comenzar la espera. La respuesta a esta pregunta depende de la variante de DOLTSP que estemos considerando. Haremos este análisis sólo para semicadenas.

En la variante Nomadic, por ejemplo, el servidor online se arriesga cada vez que abandona un vértice, porque desconoce si será necesario regresar allí en el futuro o si el recorrido que minimiza el costo total debe comenzar hacia la izquierda o hacia la derecha. En la variante Homing, en cambio,

no implica ningún riesgo alejarse del origen porque cualquier requerimiento que se presente entre el origen y la posición del servidor será atendido eventualmente en el camino de regreso. En este caso, las decisiones que conviene postergar son aquellas que impliquen que el servidor se acerque al origen.

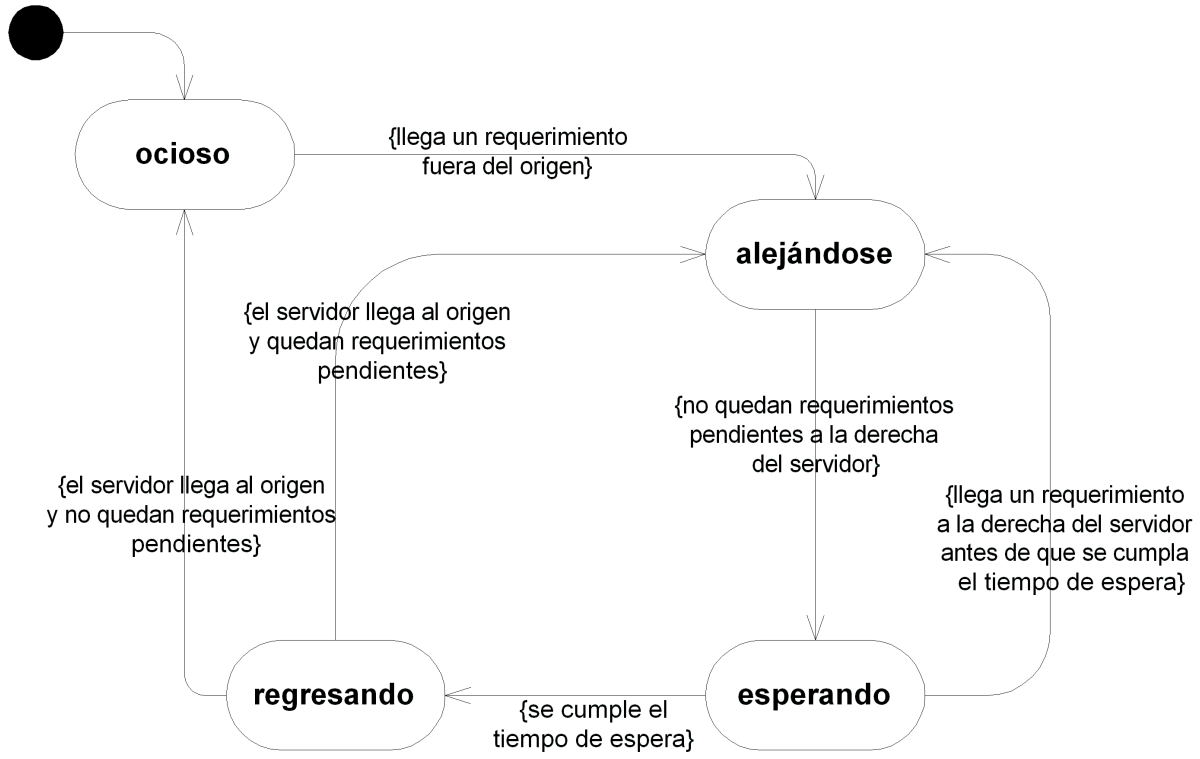
Las dos estrategias cautelosas que mostraremos a continuación reflejan estas dos posturas. En `Wait Before Begin` se planifica una ruta completa con cada nuevo requerimiento que se presenta pero el servidor online espera en su posición antes de comenzarla. Una vez terminada la espera sin que aparezcan nuevos requerimientos, el servidor completa esa ruta sin detenerse. En `Wait Before Return` el servidor se aleja del origen cada vez que es necesario para atender un requerimiento y la espera ocurre sólo antes de comenzar a acercarse al origen.

4.3 Demorar el regreso al origen

`Wait Before Return` es una estrategia cautelosa para Homing DOLTSP que consiste, básicamente, en atender primero los requerimientos más alejados del origen, dejando para el final aquellos que pueden atenderse al regresar. Antes de emprender este regreso, el servidor espera en su posición según lo expuesto en la Sección 4.2. Analizamos este algoritmo sólo para semicadenas, aunque con ciertas modificaciones podría utilizarse una idea similar para cadenas completas.

Estrategia Wait Before Return (`WBR`)

Definimos esta estrategia a partir de un autómata con cuatro estados. En el diagrama que mostramos a continuación obviamos aquellas transiciones que llevan al autómata al mismo estado del que parten.



- **ocioso** el servidor está en el origen y no hay requerimientos pendientes.
- **alejándose** el servidor se está moviendo a su derecha para atender los requerimientos pendientes.
- **esperando** el servidor está esperando antes de regresar al origen y no hay requerimientos pendientes a su derecha.
- **regresando** finalizó la espera y el servidor está regresando al origen.

WBR calcula un nuevo tiempo de espera cada vez que entra en el estado *esperando*, considerando en cada cálculo un único requerimiento, que denominamos *relevante*.

Definición 4.9. Llamamos *relevante* al último requerimiento presentado en la posición que tiene el servidor cuando WBR entra en el estado *esperando*. De acuerdo a esto podemos inferir que, si $r = (t, x)$ es un requerimiento relevante, se cumplen las siguientes propiedades:

i. Cuando se presenta r, el servidor comandado por WBR está a su izquierda y no hay otros requerimientos pendientes a la derecha de x.

ii. No se presenta otro requerimiento en x ni a la derecha de x hasta que r es atendido.

4.3.1 Cálculo de la espera

Como mencionamos en la Sección 4.2, nuestros algoritmos cautelosos determinan la espera en función de tres valores:

ρ	la competitividad que se pretende alcanzar
α	el tiempo que necesita el servidor online para atender a los requerimientos pendientes y regresar al origen
Z^*	una cota inferior para el costo óptimo offline

Para determinar el valor de ρ, notemos que con WBR pretendemos alcanzar las cotas inferiores que presentamos en el Capítulo 3. Puesto que dichas cotas son diferentes dependiendo del modelo de adversario, vamos a tomar

$$\rho = \begin{cases} \frac{2+\sqrt{2}}{2} \approx 1.707 & \text{si el adversario es } \textit{standard} \\ \frac{1+\sqrt{5}}{2} \approx 1.618 & \text{si el adversario es } \textit{fair} \end{cases}$$

Podemos obtener el valor de α considerando que WBR calcula una nueva espera cada vez que entra en el estado *esperando*, es decir, cuando acaba de servir un pedido relevante $r = (t, x)$ y sólo le resta regresar al origen. Es claro entonces que $\alpha = \bar{x}$.

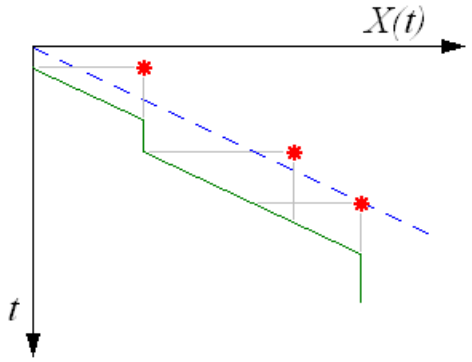
Nos resta determinar un valor adecuado para Z^*. Para esto, debemos tener en cuenta que el costo óptimo depende de las limitaciones que tiene el servidor offline. Cuando se trata de un adversario *standard*, éste puede moverse libremente, teniendo a su velocidad unitaria como única barrera. En el caso *fair*, además de esta limitación, el adversario no puede alejarse del origen más allá del límite impuesto por los requerimientos ya presentados. Para poder abstraernos del modelo de adversario, introduciremos una cota superior $X(t)$ para la distancia del servidor offline al origen en un momento arbitrario t.

Definición 4.10. En cualquier momento t, si el adversario es *standard*, vale $X(t) = t$. Si es *fair*, definimos $X(t)$ en forma recursiva:

$$X(t) = \begin{cases} 0 & \text{si antes de } t \text{ no se presentó ningún requerimiento} \\ \min\{\bar{\chi}_t, X(\theta_t) + t - \theta_t\} & \text{en caso contrario} \end{cases}$$

donde θ_t es el momento más tardío y χ_t es el vértice más alejado del origen en el que se haya presentado un requerimiento antes de t.

La parte recursiva de la definición para el caso *fair* indica que $X(t)$ no es mayor que el límite impuesto por los requerimientos ya presentados al momento t ni que la posición más alejada del origen a la que podría llegar el adversario, si en el momento θ_t comenzara a alejarse desde $X(\theta_t)$.



En el ejemplo vemos, en línea punteada, cómo varía $X(t)$ cuando el adversario es *standard* y, en línea llena, cómo lo hace cuando el adversario es *fair*.

Veamos algunas propiedades que cumple la función $X(t)$.

Lema 4.11. *Si t_0 y t son momentos tales que $t_0 \leq t$, entonces valen las siguientes desigualdades:*

i. $X(t_0) \leq X(t)$

ii. $X(t) \leq X(t_0) + t - t_0$

iii. $X(t) \leq t$

Demostración. Si el adversario es *standard*, las desigualdades del enunciado se deducen trivialmente de la Definición 4.10, puesto que $X(t) = t$ y $X(t_0) = t_0$. Si el adversario es *fair*, para ver que valen (i) y (ii) haremos inducción en la cantidad de pedidos que se presentaron antes que t. La desigualdad (iii) puede deducirse directamente de (ii), notando que cuando $t_0 = 0$, es $t_0 \leq t$ y vale $X(t_0) = 0$.

En el caso base, cuando no se presentó ningún pedido antes del momento t, vale $X(t) = 0$. Como $t_0 \leq t$, tampoco se presentó ningún pedido antes que t_0, y por lo tanto $X(t_0) = 0$. De tal modo, tenemos $X(t) = 0 = X(t_0) \leq X(t_0) + t - t_0$.

En el caso en que se presentó al menos un pedido antes que t, distinguimos dos situaciones:

1. Si $t_0 > \theta_t$, el último pedido anterior a t también es anterior a t_0. Por lo tanto, $\theta_t = \theta_{t_0}$ y $\chi_t = \chi_{t_0}$, de modo que $X(t_0) = \min\{\bar{\chi}_t, X(\theta_t) + t_0 - \theta_t\}$. Como $t_0 \leq t$, tenemos $X(t_0) \leq \min\{\bar{\chi}_t, X(\theta_t) + t - \theta_t\} = X(t)$.

   Por otro lado, $X(t_0)+t-t_0 = \min\{\bar{\chi}_t, X(\theta_t)+t_0-\theta_t\}+t-t_0 = \min\{\bar{\chi}_t+t-t_0, X(\theta_t)+t-\theta_t\}$. Como $t_0 \leq t$, tenemos $X(t_0) + t - t_0 \geq \min\{\bar{\chi}_t, X(\theta_t) + t - \theta_t\} = X(t)$.

2. Si $t_0 \leq \theta_t$, como la cantidad de pedidos que se presentaron antes que θ_t es menor que la cantidad de pedidos que se presentaron antes que t, por hipótesis inductiva tenemos que $X(\theta_t) \leq X(t_0) + \theta_t - t_0$. Por la Definición 4.10 sabemos que $X(t) \leq X(\theta_t) + t - \theta_t$. Uniendo estas dos desigualdades podemos deducir que $X(t) \leq X(t_0) + \theta_t - t_0 + t - \theta_t = X(t_0) + t - t_0$.

   Nos resta mostrar que $X(t_0) \leq X(t)$. Para esto, consideremos los dos posibles valores que puede tomar $X(t)$.

   (a) $X(t) = \bar{\chi}_t$. Si antes de t_0 no se presentaron pedidos, es $X(t_0) = 0 \leq \bar{\chi}_t = X(t)$. Si se presentó algún pedido antes que t_0, también se presentó antes que t. Es decir, $X(t_0) \leq \bar{\chi}_{t_0} \leq \bar{\chi}_t = X(t)$.

   (b) $X(t) = X(\theta_t) + t - \theta_t$. Por hipótesis inductiva, es $X(t_0) \leq X(\theta_t)$ y como $\theta_t < t$ resulta $X(t_0) < X(\theta_t) + t - \theta_t = X(t)$.

□

Lema 4.12. *Sea ς_t la posición que tiene en el momento t un servidor que siempre está dentro de la* región fair. *Entonces $\bar{\varsigma}_t \leq X(t)$.*

Demostración. Recordemos en primer lugar que, cuando se trata de semicadenas, la *región fair* está compuesta por el origen, el vértice más alejado donde se hayan presentado requerimientos, y todos los vértices que están entre ellos.

La demostración del lema es por inducción sobre la cantidad de pedidos que se presentaron antes del momento t. Si no se presentó ninguno, vale $X(t) = 0$ y el servidor, para permanecer dentro de la *región fair*, hasta el momento t no se mueve del origen. Es decir que $\bar{\varsigma}_t = 0 = X(t)$.

Si antes del momento t se presentó al menos un requerimiento, analicemos por separado cada uno de los valores que puede tomar $X(t)$ según la Definición 4.10.

1. Si $X(t) = \bar{\chi}_t$, como χ_t es el vértice más alejado del origen donde se haya presentado un requerimiento antes de t y el servidor permanece siempre en la *región fair*, es claro que $\bar{\varsigma}_t \leq X(t)$.

2. Si $X(t) = X(\theta_t) + t - \theta_t$, como θ_t indica el momento en que se presentó el último requerimiento anterior a t, la cantidad de pedidos que se presentaron antes que θ_t es menor que la cantidad de pedidos que se presentaron antes que t. Entonces, por hipótesis inductiva vale que $\bar{\varsigma}_{\theta_t} \leq X(\theta_t)$. Debido a que el servidor se desplaza a velocidad unitaria, es $\bar{\varsigma}_t \leq \bar{\varsigma}_{\theta_t} + t - \theta_t$. Uniendo esta expresión con la desigualdad anterior, llegamos a que $\bar{\varsigma}_t \leq X(\theta_t) + t - \theta_t = X(t)$.

□

Corolario 4.13. *En todo momento t, la distancia del servidor offline al origen es a lo sumo $X(t)$.*

Demostración. Si el adversario es *standard*, vale $X(t) = t$ y la validez del enunciado se desprende de que el servidor offline se desplaza a velocidad unitaria. Si el adversario es *fair*, permanece siempre dentro de la *región fair* y, por lo tanto, el enunciado se deduce del Lema 4.12. □

Corolario 4.14. *En todo momento t, la distancia del servidor comandado por `WBR` al origen es a lo sumo $X(t)$.*

Demostración. El servidor comandado por `WBR` sólo se aleja del origen si tiene requerimientos pendientes a su derecha, es decir, permanece siempre dentro de la *región fair*. Por lo tanto, podemos deducir el enunciado directamente del Lema 4.12. □

Ahora que conocemos que en cada momento t la distancia del servidor offline al origen es a lo sumo $X(t)$, podemos definir Z^* abstrayéndonos del modelo de adversario. Si $r = (t, x)$ es el requerimiento relevante al momento de calcular una espera, Z^* queda definido de la siguiente manera:

$$Z^* = \begin{cases} t + \bar{x} & \text{si } \bar{x} < X(t) \\ t + 2\bar{x} - X(t) & \text{si } \bar{x} \geq X(t) \end{cases} \tag{4.3.2}$$

Veamos que Z^* es, como queríamos, una cota inferior para el costo óptimo offline.

Lema 4.15. *Sea $r = (t, x) \in \sigma$ un requerimiento relevante y sea Z^* calculado a partir de t y x. Entonces $Z^* \leq OPT(\sigma_{\leq t})$.*

Demostración. Según la definición, si $\bar{x} < X(t)$, entonces $Z^* = t + \bar{x}$, momento antes del cual ningún servidor puede finalizar el servicio si tiene que atender a r y regresar al origen.

Si, por el contrario, $\bar{x} \geq X(t)$, vale $Z^* = t + 2\bar{x} - X(t)$. El Corolario 4.13 asegura que en el momento t la distancia del adversario al origen es a lo sumo $X(t)$ y, por lo tanto, no podrá alcanzar el vértice x antes del momento $t + \bar{x} - X(t)$. Como luego de servir r todavía le resta regresar al origen, no podrá concluir su tarea antes del momento $t + 2\bar{x} - X(t) = Z^*$. □

En conclusión, podemos reemplazar los parámetros ρ, α y Z^* de la expresión 4.2.1 con los valores que establecimos en esta sección. Así, si $r = (t, x)$ es el requerimiento relevante al calcular una espera de `WBR`, ésta queda definida de la siguiente manera:

$$w = \begin{cases} \rho(t + \bar{x}) - \bar{x} & \text{si } \bar{x} < X(t) \\ \rho(t + 2\bar{x} - X(t)) - \bar{x} & \text{si } \bar{x} \geq X(t) \end{cases} \tag{4.3.3}$$

4.3.2 Correctitud de WBR

De la definición del algoritmo podemos concluir que `WBR` obtiene una solución factible para cualquier secuencia de requerimientos. Sin embargo, debemos verificar que cada vez que se calcula una espera, la misma no es anterior al momento de cálculo. Si fuera anterior, significaría que la competitividad deseada no siempre es alcanzable. A continuación mostramos algunas propiedades que nos permitirán garantizar que la espera es válida cada vez que se la calcula, ya sea compitiendo contra un adversario *fair* o contra uno *standard*.

Lema 4.16. *Sea $r = (t, x)$ un requerimiento relevante y sea w la espera calculada por `WBR` cuando r es atendido. Entonces valen a la vez:*

i. $w \geq (2\rho - 1)\bar{x}$

ii. $w \geq t + 2(\rho - 1)\bar{x}$

iii. $w \geq t + \bar{x}$

Demostración. Primero verifiquemos que valen (i) y (ii), analizando por separado cada una de las dos situaciones contempladas en la expresión 4.3.3. Si $\bar{x} < X(t)$, tenemos que $w = \rho(t + \bar{x}) - \bar{x} = t + (\rho - 1)(t + \bar{x})$. Como $t \geq X(t) > \bar{x}$, de la primera forma de expresar w deducimos que $w > 2\rho\bar{x} - \bar{x} = (2\rho - 1)\bar{x}$, mientras que la segunda forma nos dice que $w > t + 2(\rho - 1)\bar{x}$.

En cambio, si $\bar{x} \geq X(t)$, ocurre que $w = \rho(t + 2\bar{x} - X(t)) - \bar{x} = \rho(t - X(t)) + (2\rho - 1)\bar{x}$. Como $X(t) \leq t$, tenemos $w \geq (2\rho - 1)\bar{x}$. Aquí también podríamos escribir la espera como $w = t - X(t) +$

$(\rho-1)(t-X(t))+(2\rho-1)\bar{x}$, y usando una vez más que $X(t)\leq t$, resulta $w\geq t-X(t)+(2\rho-1)\bar{x}$. Finalmente, como $\bar{x}\geq X(t)$, concluimos que $w\geq t-\bar{x}+(2\rho-1)\bar{x}=t+2(\rho-1)\bar{x}$.

Por último, si observamos que $2(\rho-1)>1$ para los dos valores posibles de ρ, se deduce (iii) directamente de (ii). □

Lema 4.17. *Sea t_0 el momento en que el servidor comandado por* `WBR` *comienza un regreso al origen, compitiendo contra un adversario* fair. *Entonces $t_0\geq(2\rho-1)X(t_0)$.*

Demostración. Según su definición, `WBR` comienza un regreso al origen sólo si se cumplió una espera w, es decir, $t_0\geq w$. Esa espera se calcula en función de un requerimiento relevante $r=(t,x)$, donde x es la posición del servidor en el momento t_0. De acuerdo a esto, el Corolario 4.14 afirma que $\bar{x}\leq X(t_0)$. Distinguiremos dos situaciones:

1. Si $\bar{x}=X(t_0)$, usamos el Lema 4.16 para afirmar que $t_0\geq w\geq(2\rho-1)\bar{x}=(2\rho-1)X(t_0)$.

2. Si $\bar{x}<X(t_0)$, debe ser $X(t_0)>0$, por lo que según la Definición 4.10 antes del momento t_0 se presentó al menos un pedido. De acuerdo a esa definición es $X(t_0)\leq\bar{\chi}_{t_0}$, donde χ_{t_0} es el vértice más alejado del origen donde se hayan presentado requerimientos antes del momento t_0. Como por hipótesis $\bar{x}<X(t_0)$, entonces χ_{t_0} está a la derecha de x. Esto implica que los pedidos presentados en χ_{t_0} debieron ser atendidos antes del momento t_0, puesto que en ese momento el servidor comienza un regreso desde x.

   Por la definición de `WBR` sabemos que antes de atender a r en x, y por lo tanto también antes de t_0, el servidor emprendió un regreso al origen desde χ_{t_0} luego de una espera w'. Usando nuevamente el Lema 4.16, obtenemos que $t_0>w'\geq(2\rho-1)\bar{\chi}_{t_0}\geq(2\rho-1)X(t_0)$.

□

Lema 4.18. *Sea $r=(t,x)$ un requerimiento relevante que se presenta cuando el servidor comandado por* `WBR` *está regresando al origen, compitiendo contra un adversario* fair*, con $\bar{x}<X(t)$. Entonces $t+\bar{x}>2\rho X(t)$.*

Demostración. Haremos un razonamiento similar al que usamos para demostrar el Lema 4.17. Como $\bar{x}<X(t)$, resulta $X(t)>0$. La Definición 4.10 nos indica entonces que antes del momento t se presentó al menos un pedido. De acuerdo a esa definición es $X(t)\leq\bar{\chi}_t$, y como por hipótesis es $\bar{x}<X(t)$, entonces χ_t está a la derecha de x. Esto implica que los pedidos presentados en χ_t debieron ser atendidos antes del momento t, dado que al presentarse $r=(t,x)$ no hay pedidos pendientes a la derecha de x, porque r es relevante.

Por la definición de WBR sabemos que, una vez atendidos los requerimientos presentados en χ_t, en algún momento t_0 anterior a t, el servidor comenzó desde ese vértice un regreso al origen. Según el Lema 4.17, es $t_0 \geq (2\rho - 1)X(t_0)$.

Más arriba vimos que $X(t) \leq \bar{\chi}_t$. Como χ_t es la posición del servidor en el momento t_0, el Corolario 4.14 asegura que $\bar{\chi}_t \leq X(t_0)$. Y como $t_0 < t$, el Lema 4.11 afirma que $X(t_0) \leq X(t)$. En resumen, $X(t) \leq \bar{\chi}_t \leq X(t_0) \leq X(t)$, por lo que $X(t) = \bar{\chi}_t = X(t_0)$, lo que nos permite concluir que

$$t_0 \geq (2\rho - 1)X(t) \tag{4.3.4}$$

Que $r = (t, x)$ sea relevante significa que en el momento t el servidor está a la izquierda de x. Pero luego de que comienza el regreso al origen desde χ_t, esto sólo ocurre a partir del momento $t_0 + \bar{\chi}_t - \bar{x}$. Es claro entonces que debe ser $t > t_0 + \bar{\chi}_t - \bar{x}$. O, sumando $\bar{x}$ en ambos miembros, $t + \bar{x} > t_0 + \bar{\chi}_t = t_0 + X(t)$ que, según la expresión 4.3.4, es mayor que $2\rho X(t)$. □

Como ya dijimos, las propiedades que acabamos de exponer no persiguen otro fin más que permitirnos mostrar la correctitud de WBR tanto contra un adversario *fair* como contra uno *standard*, lo que haremos a continuación.

Teorema 4.19. *Sea $r = (t, x)$ un requerimiento relevante y sea w la espera calculada por WBR cuando r es atendido, compitiendo contra un adversario* fair. *Entonces w no es anterior al momento en que r es atendido.*

Demostración. Si r se presenta cuando WBR está *ocioso*, *alejándose* o *esperando*, entonces el servidor online se dirige directamente hacia x, atendiendo r a lo sumo en el momento $t + \bar{x}$, que según el Lema 4.16 no supera a w.

En cambio, si r se presenta mientras WBR está *regresando*, el servidor online terminará su retorno al origen antes de dirigirse hacia x. Distinguiremos dos situaciones:

1. Si $\bar{x} < X(t)$, sea s la posición del servidor online en el momento t. Como en ese momento está regresando, llegará al origen en el momento $t + \bar{s}$ y, puesto que r es relevante, debe ser $\bar{s} < \bar{x}$. En conclusión, el servidor online llega al origen antes del momento $t + \bar{x}$. Es decir que r será atendido a más tardar en el momento $t + 2\bar{x}$.

   De acuerdo a la expresión 4.3.3, es $w = \rho(t + \bar{x}) - \bar{x} = t + (\rho - 1)(t + \bar{x})$, que por el Lema 4.18 es mayor que $t + (\rho - 1)2\rho X(t)$. Por tratarse de un adversario *fair*, resulta $\rho(\rho - 1) = 1$, de modo que concluimos que $w > t + 2X(t) > t + 2\bar{x}$.

2. Consideremos ahora el caso $\bar{x} \geq X(t)$, en el que $w = \rho(t + 2\bar{x} - X(t)) - \bar{x}$. Sea t_0 el momento en que el servidor online comenzó su actual regreso al origen, desde un vértice que, según el

Corolario 4.14, no puede estar más lejos que $X(t_0)$. Dado que llega al origen no más tarde que $t_0 + X(t_0)$, podemos afirmar que r será atendido a más tardar en el momento $t_0 + X(t_0) + \bar{x}$.

Por otro lado, como $t_0 < t$, el Lema 4.11 asegura que $X(t) \leq X(t_0) + t - t_0$. Esto implica que $w = \rho(t+2\bar{x}-X(t))-\bar{x} \geq \rho(t+2\bar{x}-(X(t_0)+t-t_0))-\bar{x} = \rho(t_0+2\bar{x}-X(t_0))-\bar{x}$. Reacomodando, $w \geq t_0 + \bar{x} + (\rho-1)t_0 + 2(\rho-1)\bar{x} - \rho X(t_0)$. Dado que $x \geq X(t) \geq X(t_0)$ (también por el Lema 4.11), resulta $w \geq t_0 + \bar{x} + (\rho-1)t_0 + (\rho-2)X(t_0)$. Por el Lema 4.17, sabemos que $t_0 \geq (2\rho-1)X(t_0)$, de modo que nos queda $w \geq t_0 + \bar{x} + (\rho-1)(2\rho-1)X(t_0) + (\rho-2)X(t_0)$. Debido al valor de ρ correspondiente a un adversario *fair*, sucede que $(\rho-1)(2\rho-1)+(\rho-2) = 1$ y entonces la desigualdad anterior es equivalente a $w \geq t_0 + \bar{x} + X(t_0)$.

□

Teorema 4.20. *Sea $r = (t, x)$ un requerimiento relevante y sea w la espera calculada por* `WBR` *cuando r es atendido, compitiendo contra un adversario* standard. *Entonces w no es anterior al momento en que r es atendido.*

Demostración. Si r se presenta cuando `WBR` está *ocioso*, *alejándose* o *esperando*, entonces el servidor online se dirige directamente hacia x, atendiendo r a lo sumo en el momento $t + \bar{x}$, que según el Lema 4.16 no supera a w.

En cambio, si r se presenta mientras `WBR` está *regresando*, el servidor online terminará su retorno al origen antes de dirigirse hacia x. Si llamamos t_0 y x_0 respectivamente al momento y al vértice de comienzo del actual regreso al origen, entonces r será servido en el momento $t_0 + \bar{x}_0 + \bar{x}$. Distinguiremos dos situaciones:

1. Si $\bar{x}_0 \leq (2\rho-3)\bar{x}$, por el Lema 4.16 sabemos que $w \geq t + 2(\rho-1)\bar{x}$, y como $t > t_0$ resulta $w > t_0 + 2(\rho-1)\bar{x} = t_0 + (2\rho-3)\bar{x} + \bar{x} \geq t_0 + \bar{x}_0 + \bar{x}$.

2. Consideremos ahora el caso $\bar{x}_0 > (2\rho-3)\bar{x}$. Según la expresión 4.3.3, cuando $\bar{x} < X(t)$, la espera se calcula como $w = \rho(t+\bar{x}) - \bar{x}$, y cuando $\bar{x} \geq X(t)$, se calcula como $w = \rho(t + 2\bar{x} - X(t)) - \bar{x}$. Pero, por tratarse de un adversario *standard*, es $X(t) = t$, lo que implica que en este último caso es $w = \rho(2\bar{x}) - \bar{x} \geq \rho(t+\bar{x}) - \bar{x}$. Dicho de otro modo, resulta $w \geq \rho(t+\bar{x}) - \bar{x}$, independientemente de si $\bar{x}$ es mayor o menor que $X(t)$. Dado que $t > t_0$ resulta $w > \rho(t_0+\bar{x}) - \bar{x} = t_0 + (\rho-1)(t_0+\bar{x})$. Como el regreso comienza desde x_0 en el momento t_0 (al finalizar una espera) podemos usar el Lema 4.16 (i) para afirmar que $t_0 \geq (2\rho-1)\bar{x}_0$ y concluir que $w > t_0 + (\rho-1)((2\rho-1)\bar{x}_0 + \bar{x})$. Debido al valor de ρ resulta $(\rho-1)(2\rho-2) = 1$, entonces llegamos a $w > t_0 + \bar{x}_0 + (\rho-1)(\bar{x}_0+\bar{x})$. Finalmente, utilizando que $\bar{x}_0 > (2\rho-3)\bar{x}$ llegamos a $w > t_0 + \bar{x}_0 + (\rho-1)((2\rho-3)\bar{x} + \bar{x}) = w > t_0 + \bar{x}_0 + \bar{x}$.

□

Ahora que sabemos que WBR obtiene siempre soluciones factibles y calcula esperas válidas, sólo nos resta enunciar su competitividad.

Teorema 4.21. *WBR es una estrategia ρ-competitiva para la variante Homing de DOLTSP en semi-cadenas, con $\rho = \frac{1+\sqrt{5}}{2}$ cuando el adversario es* fair *y $\rho = \frac{2+\sqrt{2}}{2}$ cuando es* standard.

Demostración. En los Teoremas 4.19 y 4.20 mostramos para ambos modelos de adversario que la espera es válida cada vez que se la calcula. Además, como vimos en la Sección 4.2, si el servidor comienza a moverse en el momento $w = \rho\ Z^* - \alpha$, termina su recorrido exactamente en el momento ρZ^*. Finalmente, en el Lema 4.15, mostramos que Z^* (tal como la definimos en la expresión 4.3.2) es una cota para el costo óptimo offline cada vez que se calcula una espera. Resumiendo, para toda secuencia de requerimientos σ, vale que $WBR(\sigma) = \rho\ Z^* \leq \rho\ OPT(\sigma)$. □

4.4 Demorar el próximo movimiento

Al igual que `WBR`, la estrategia `Wait Before Begin` está dentro de la categoría de algoritmos cautelosos. Consiste esencialmente en mantener al servidor en su posición mientras se presentan requerimientos y, luego de una espera definida según lo expuesto en la Sección 4.2, comenzar un recorrido para atenderlos. A continuación definimos el comportamiento de `WBB` para la variante Nomadic de DOLTSP en semicadenas y analizamos su competitividad. En la Sección 4.5 extenderemos su definición para cadenas completas y también para la variante Homing.

Para definir el comportamiento de esta estrategia no es necesario conocer la posición de todos los requerimientos pendientes ni los momentos en que se presentaron. Alcanza con identificar dos vértices tales que un recorrido que los incluya le permita al servidor atender a todos los requerimientos pendientes. En `WBB`, estos vértices se llaman *extremos*.

Definición 4.22. Cuando el servidor está sobre un vértice v, consideremos el conjunto formado por v y por todos los vértices que le quedan por visitar para finalizar su servicio, es decir, aquellos donde haya requerimientos pendientes. En la variante Homing este conjunto también incluye al origen. De los vértices de este conjunto, llamamos *extremo izquierdo* al que está más a la izquierda y *extremo derecho* al que está más a la derecha. A lo largo de esta sección notaremos con x al extremo izquierdo y con y al extremo derecho.

Esta definición nos será útil por diferentes motivos. En primer lugar, garantiza que el servidor comandado por `WBB` está siempre entre el extremo izquierdo y el extremo derecho. Esto nos ayudará luego, cuando calculemos el tiempo que necesita el servidor para finalizar el recorrido. Notemos la diferencia con la Definición 4.6 que usamos para `Replan`, según la cual el servidor no siempre se encuentra entre los extremos.

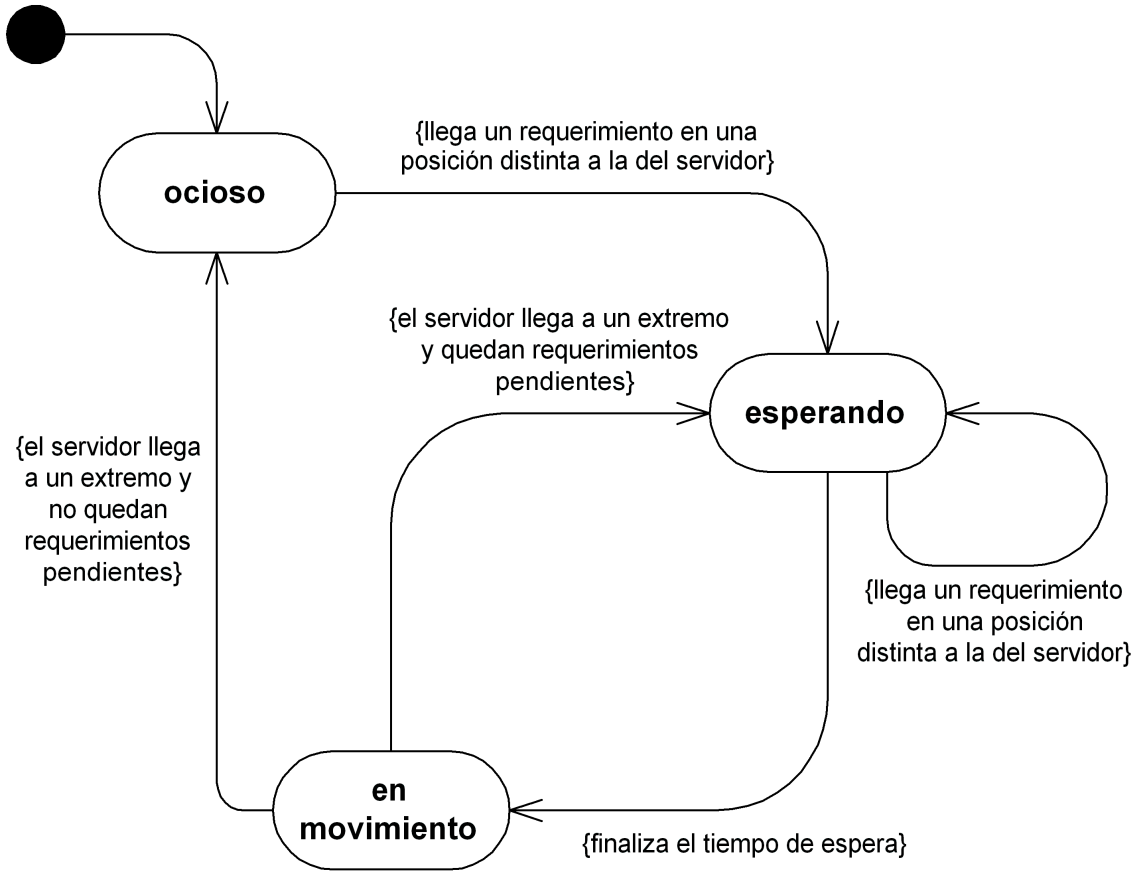
Por otro lado, veremos que el servidor comandado por `WBB` sólo se aleja del origen si tiene a su derecha algún pedido pendiente. Esto significa que, según la Definición 4.22, ambos extremos están siempre entre el origen y el vértice más alejado donde se haya presentado algún requerimiento. Con lo cual, todos los vértices que en algún momento son extremos forman parte de cualquier recorrido que atiende a todos los pedidos. Usaremos esta propiedad para calcular una cota inferior para el costo óptimo offline.

Finalmente, para atender a todos los requerimientos pendientes, el servidor sólo necesita visitar el extremo derecho y el extremo izquierdo. El orden en que estos vértices deben visitarse es una de las decisiones que tomaremos a continuación.

Estrategia Wait Before Begin (WBB)

Definimos esta estrategia a partir de un autómata con tres estados:

- **ocioso** no hay requerimientos pendientes, el servidor permanece en su lugar.
- **esperando** hay requerimientos pendientes, pero el servidor espera sin moverse.
- **en movimiento** el servidor se mueve hasta uno de los extremos.



WBB calcula un tiempo de espera con cada una de las tres transiciones que lo llevan al estado *esperando*. Si se presenta un requerimiento en la misma posición del servidor cuando éste se encuentra *ocioso* o *esperando*, ese requerimiento queda automáticamente servido y no se genera un nuevo cálculo de espera. Cada vez que finaliza una espera, el servidor comienza a moverse y se detiene cuando llega a uno de los extremos que estaban vigentes al momento de partir.

4.4.1 Cuánto esperar y hacia dónde partir

Es oportuno recordar que, de acuerdo a lo explicado en la Sección 4.2, calculamos la espera a partir de tres valores: ρ, Z^* y α.

El primero de estos valores es la competitividad que pretendemos alcanzar con `WBB`. Dado que en el Capítulo 3 hemos mostrado que ningún algoritmo online puede ser mejor que 2-competitivo contra un adversario *standard*, tomaremos $\rho = 2$.

Z^* es una cota inferior para el costo óptimo offline. Podemos encontrar un valor apropiado si recordamos que, como dijimos al comienzo de esta sección, todos los vértices que en algún momento son extremos forman parte de cualquier recorrido que atiende a todos los pedidos. En particular, los extremos vigentes al momento de calcular la espera forman parte del recorrido del adversario. Pero el adversario no sólo debe visitarlos, sino también atender todos los requerimientos que se presenten en estos vértices. Es decir que si la espera se calcula a partir de los requerimientos que se presentaron hasta cierto momento t, algunas cotas inferiores que se pueden establecer para el costo óptimo offline surgen de considerar los siguientes recorridos:

i. ir al extremo izquierdo y atender los requerimientos que se presentaron allí

ii. ir al extremo derecho y atender los requerimientos que se presentaron allí

iii. ir a uno de los extremos, atender los requerimientos que se presentaron allí y moverse hasta el otro extremo

Usaremos la siguiente notación para indicar el menor tiempo que le lleva a cualquier servidor llegar a un vértice en particular y atender todos los requerimientos que se hayan presentado allí hasta cierto momento.

Definición 4.23. Dado un vértice cualquiera v, notemos con $\theta_t(v)$ al momento más tardío en que se presentó un requerimiento en v hasta el momento t inclusive. Si no se presentó ninguno, sea $\theta_t(v) = 0$. A partir de este valor, definimos

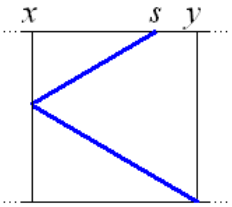
$$\hat{v}_t = \max\{\theta_t(v), \bar{v}\}$$

Resulta claro que antes del momento $\hat{v}_t$ ningún servidor puede moverse hasta el vértice v y atender a todos los requerimientos que se hayan presentado en v hasta el momento t. De esta forma, podemos reescribir las tres cotas inferiores que mencionamos como $\hat{x}_t$, $\hat{y}_t$ y $\min\{\hat{x}_t, \hat{y}_t\} + d(x, y)$, respectivamente. Para la definición de Z^* usaremos el mayor de estos tres valores:

$$Z^* = \max \begin{cases} \max\{\hat{x}_t, \hat{y}_t\} \\ \min\{\hat{x}_t, \hat{y}_t\} + d(x, y) \end{cases} \tag{4.4.5}$$

Resta determinar el valor de α, es decir, el tiempo que necesita el servidor online para atender a los requerimientos pendientes, lo que equivale a visitar ambos extremos. Pero ¿qué extremo debería visitar primero el servidor? Veremos que esta decisión es crucial para obtener la competitividad deseada. En esta estrategia, el próximo movimiento será siempre hacia el extremo izquierdo, a menos que el servidor ya esté allí, en cuyo caso se moverá hacia el extremo derecho. Por lo tanto, el tiempo α varía de acuerdo a las siguientes condiciones:

Si $x < s < y$, será $\alpha = d(s, x) + d(x, y)$.



Si $s = x$ o $s = y$, será $\alpha = d(x, y)$.

Entonces, combinando los valores definidos para ρ, Z^* y α según la expresión 4.2.1, en cada momento t en que `WBB` calcula una espera w, lo hace de la siguiente forma:

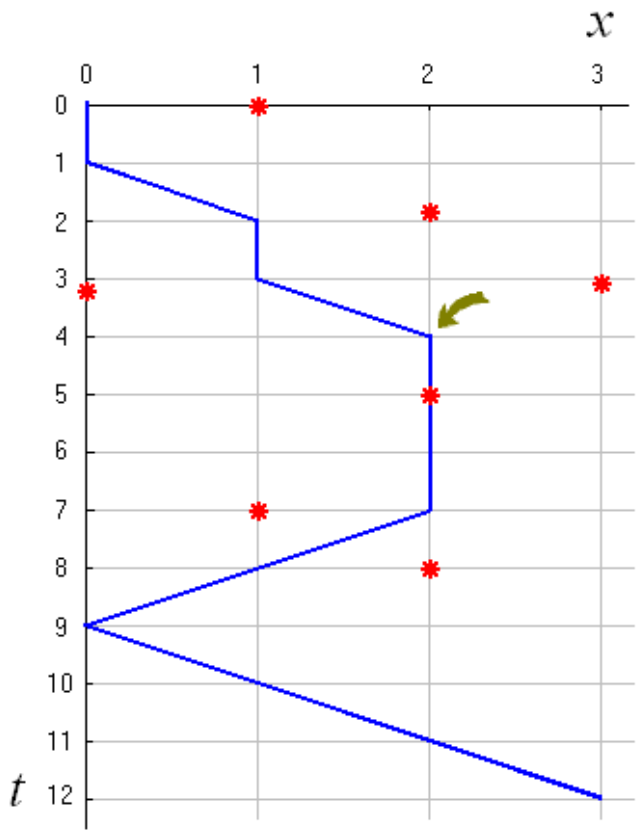
Si el servidor tiene requerimientos pendientes a ambos lados ($x < s < y$)

$$w = \max \begin{cases} 2\max\{\hat{x}_t, \hat{y}_t\} - (d(s, x) + d(x, y)) \\ 2\min\{\hat{x}_t, \hat{y}_t\} + d(s, y) \end{cases} \tag{4.4.6}$$

Si el servidor está en alguno de los extremos ($s = x$ o $s = y$)

$$w = \max \begin{cases} 2\max\{\hat{x}_t, \hat{y}_t\} - d(x, y) \\ 2\min\{\hat{x}_t, \hat{y}_t\} + d(x, y) \end{cases} \tag{4.4.7}$$

Quizás con un ejemplo podamos apreciar más claramente cómo es el funcionamiento de `WBB` en una semicadena.



En el gráfico se observa cómo el servidor espera en su lugar en los momentos 0, 2 y 4, pese a la existencia de requerimientos pendientes. En el cálculo de la espera que indica la flecha, se determina que el costo óptimo no será menor que 6 y, en consecuencia, `WBB` se propone terminar en el momento 12. A partir de entonces los requerimientos que aparecen no modifican los extremos y, por lo tanto, a pesar de que el costo óptimo termina siendo 8, no se extiende el costo del recorrido online.

Es claro que el recorrido propuesto no siempre coincide con el de menor longitud. Otra posibilidad hubiera sido determinar el próximo extremo a servir en función de cuál sea el recorrido restante más corto en ese momento. Sin embargo, si fuera así, podemos encontrar situaciones en las que el servidor online queda en una desventaja notoria respecto de su adversario, consiguiendo que el costo online sea más de dos veces mayor que el costo óptimo offline.

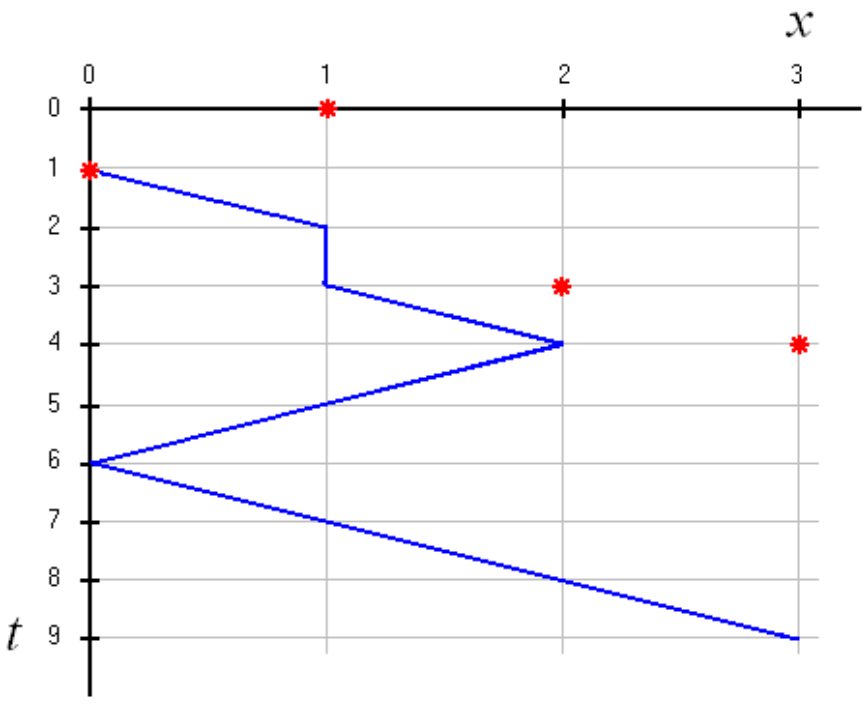
Analicemos brevemente cómo funcionaría esta estrategia hipotética. El servidor online se dirigiría siempre al extremo que tenga más cerca y luego hacia el extremo opuesto. Entonces, cada vez que se comienza una espera en cierto momento t, el tiempo restante para finalizar el recorrido online sería $\alpha = \min\{d(s,x), d(s,y)\} + d(x,y)$ y por lo tanto,

$$w = \max \begin{cases} 2\max\{\hat{x}_t, \hat{y}_t\} - \min\{d(s,x), d(s,y)\} - d(x,y) \\ 2\min\{\hat{x}_t, \hat{y}_t\} - \min\{d(s,x), d(s,y)\} + d(x,y) \end{cases} \tag{4.4.8}$$

Ahora apliquemos esta forma de calcular la espera a una instancia de Nomadic DOLTSP formada por los siguientes requerimientos:

$$r_1 = (0, 1)$$
$$r_2 = (1, 0)$$
$$r_3 = (3, 2 - \varepsilon)$$
$$r_4 = (4, 3)$$

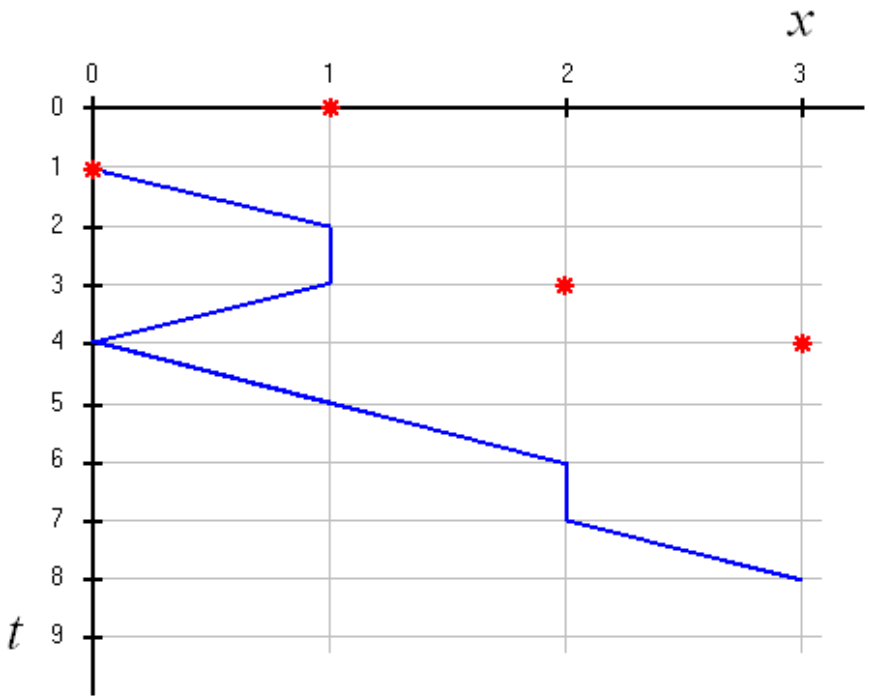
En el siguiente análisis, el lector notará que forzamos arbitrariamente el orden de los sucesos, de modo de llegar a la situación buscada. Esto es así para simplificar la comprensión del caso. De todas formas, añadiendo convenientemente pequeñas modificaciones a los tiempos o las posiciones donde se presentan los requerimientos, puede encontrarse una situación real donde el mismo orden de sucesos se dé en forma natural. Hechas las aclaraciones correspondientes, veamos cómo sería el comportamiento de una estrategia similar a `WBB` en la que el servidor busca recorrer siempre el camino más corto.



$t = 0$) Mientras el servidor está *ocioso*, se presenta r_1 en $x = 1$ y obtenemos $w = 1$.
$t = 1$) Se cumple la espera y el servidor parte hacia $x = 1$ para servir r_1.
$t = 1$) Mientras el servidor está *en movimiento*, se presenta r_2 en $x = 0$.
$t = 2$) El servidor llega a $x = 1$ y encuentra que r_2 está pendiente. Obtenemos $w = 3$.
$t = 3$) Mientras el servidor está *esperando*, se presenta r_3 en $x = 2 - \varepsilon$ y obtenemos $w = 3$.
$t = 3$) Como r_3 es el requerimiento más cercano para el servidor, parte hacia $x = 2 - \varepsilon$.
$t = 4$) El servidor llega a $x = 2 - \varepsilon$ y encuentra que r_2 está pendiente. Obtenemos $w = 4$.
$t = 4$) El servidor parte hacia $x = 0$.
$t = 4$) Mientras el servidor está *en movimiento*, se presenta r_4 en $x = 3$.
$t = 6$) El servidor llega a $x = 0$ y encuentra que r_4 está pendiente. En este punto descubrimos que para ser 2-competitivo, el servidor hubiera tenido que partir hacia $x = 3$ en el momento $w = 5$.

Como se advierte en el gráfico, el adversario puede atender primero el requerimiento r_2 en el origen y luego alejarse sin necesidad de regresar en ningún momento. Ese requerimiento es el que le queda pendiente al servidor online y lo obliga más tarde a realizar un costoso regreso. Por esta razón resulta más seguro atender primero los requerimientos cercanos al origen para luego permitirle al servidor online alejarse sin correr riesgos que el adversario puede evitar.

A continuación mostramos cómo se comporta `WBB` ante la misma secuencia de requerimientos que en el último ejemplo. Para ser consistentes, suponemos las mismas alteraciones en los tiempos y en las posiciones donde se presentan los requerimientos.



Claro está que la solución propuesta puede aplicarse en semicadenas pero no en cadenas completas, puesto que en este caso podrían aparecer requerimientos del otro lado del origen. Algunos

autores se han topado con este mismo problema y lo han resuelto de diversas maneras. El lector puede encontrar una solución alternativa en [Lip03], en la definición de la estrategia *Wait and Anticipate* (`WA`), que es válida para rectas y semirrectas, aunque la competitividad alcanzada por `WA` es un poco mayor que la de `WBB`.

4.4.2 Correctitud de WBB

Puede observarse en la definición del algoritmo que, dada una secuencia de requerimientos cualquiera, `WBB` genera un recorrido con el que el servidor atiende a todos los requerimientos. Resta verificar, como hicimos para `WBR`, que cada vez que se calcula una espera, ésta no es anterior al momento de cálculo. Con este fin, veremos algunas propiedades que se satisfacen cuando `WBB` calcula una espera de acuerdo a las expresiones 4.4.6 y 4.4.7.

Lema 4.24. *Si el servidor comandado por `WBB` se encuentra en un extremo al calcular una espera w, entonces resulta $w \geq d(x, y)$.*

Demostración. Sea t el momento en que se calcula la espera. Surge de la expresión 4.4.7 que si el servidor está en un extremo, entonces $w \geq 2\min\{\hat{x}_t, \hat{y}_t\} + d(x, y)$. Puesto que tanto $\hat{x}_t$ como $\hat{y}_t$ son valores positivos, vale también $w \geq d(x, y)$. □

Lema 4.25. *Si `WBB` calcula una espera w en cierto momento t, resulta*

$$w \geq \max \begin{cases} 2\max\{\hat{x}_t, \hat{y}_t\} - (d(s, x) + d(x, y)) \\ 2\min\{\hat{x}_t, \hat{y}_t\} + d(s, y) \end{cases}$$

Demostración. Basta notar que, según las expresiones 4.4.6 y 4.4.7, la espera calculada cuando el servidor está en un extremo es siempre mayor que la calculada cuando el servidor tiene requerimientos pendientes a ambos lados. Más precisamente, la diferencia entre ambos valores es $d(s, x)$.

□

Lema 4.26. *Sea t el momento en que el servidor comandado por `WBB` comienza a moverse hacia el vértice v. Entonces $t \geq \bar{v}$.*

Demostración. El servidor comandado por `WBB` comienza a moverse sólo si finaliza una espera w, es decir, $t \geq w$. Sea t_0 el momento en que se calcula dicha espera. Según la definición del algoritmo, v es uno de los extremos vigentes en ese momento. Si v es el extremo izquierdo, entonces el Lema 4.25 nos dice que $t \geq w \geq 2\min\{\hat{x}_{t_0}, \hat{y}_{t_0}\} + d(s, y) \geq \min\{\hat{x}_{t_0}, \hat{y}_{t_0}\} \geq \min\{\bar{x}, \bar{y}\} \geq \bar{x} = \bar{v}$. Si v es el extremo derecho, entonces al momento de partir debió estar sobre el extremo izquierdo y, según la expresión 4.4.7, debió ser $t \geq w \geq 2\max\{\hat{x}_{t_0}, \hat{y}_{t_0}\} - d(x, y) \geq 2\hat{y} - d(x, y) \geq 2\bar{y} - d(x, y) \geq \bar{y} = \bar{v}$. □

Lema 4.27. *Si en cierto momento t el servidor comandado por* `WBB` *se encuentra en un vértice v a la derecha del extremo izquierdo ($\bar{v} > \bar{x}$), entonces $\hat{x}_t > \bar{v}$.*

Demostración. Que el servidor se encuentre a la derecha del extremo izquierdo significa que en x hay al menos un pedido pendiente. Por otro lado, como $\bar{v} > \bar{x} \geq 0$, el vértice v no puede ser el origen. Entonces, es claro que debió existir un movimiento que finalizó en v. De acuerdo a la definición de `WBB`, ese movimiento forma parte de una secuencia de movimientos en un mismo sentido que notaremos con m. Sea t_0 el momento en que comenzó m.

Si m fue hacia la izquierda, entonces v era el extremo izquierdo en el momento t_0, lo que implica que el requerimiento que determinó el extremo izquierdo en x se presentó después de t_0. En cambio, si m fue hacia la derecha, el servidor debió partir desde el extremo izquierdo vigente en t_0, quedando servidos todos los requerimientos que se hayan presentado hasta el momento t_0. Dado que en el momento t hay un requerimiento pendiente en x, este requerimiento debió presentarse en un momento posterior a t_0.

Hasta aquí tenemos que, independientemente de la dirección que tuvo m, es $\hat{x}_t > t_0$. Como el Lema 4.26 asegura que $t_0 \geq \bar{v}$, podemos concluir que $\hat{x}_t > \bar{v}$. □

Lema 4.28. *En cualquier momento t en el que el servidor comandado por* `WBB` *esté sobre un vértice, vale* $\min\{\hat{x}_t, \hat{y}_t\} \geq \bar{s}$.

Demostración. Si en el momento t el servidor online está en el extremo izquierdo ($s = x$), resulta $\min\{\hat{x}_t, \hat{y}_t\} \geq \min\{\bar{x}, \bar{y}\} = \bar{x} = \bar{s}$. Caso contrario, si el servidor está a la derecha del extremo izquierdo ($s > x$), el Lema 4.27 nos asegura que $\hat{x}_t > \bar{s}$. Es decir que $\min\{\hat{x}_t, \hat{y}_t\} \geq \min\{\bar{s}, \bar{y}\} = \bar{s}$. □

Las propiedades que acabamos de mostrar nos serán útiles para probar que cada vez que `WBB` calcula una espera w en determinado momento t, es $w \geq t$. Veamos qué sucede en cada una de las situaciones en las que se calcula una espera.

Teorema 4.29. *Si en el momento t* `WBB` *calcula una espera w en una transición* ocioso → esperando*, entonces $w \geq t$.*

Demostración. Que la transición sea de *ocioso* a *esperando* significa que en el momento t se presenta al menos un requerimiento en una posición distinta a la del servidor. Por supuesto, podrían aparecer simultáneamente varios requerimientos. Si todos ellos aparecieran en un mismo lado del servidor, su posición determinaría un extremo y, según la expresión 4.4.7, la espera calculada sería $w \geq 2\max\{\hat{x}_t, \hat{y}_t\} - d(x, y)$. Dado que en el extremo opuesto al servidor se presenta un requerimiento en el momento t, tenemos que $\max\{\hat{x}_t, \hat{y}_t\} \geq t$ y queda $w \geq t + \max\{\hat{x}_t, \hat{y}_t\} - d(x, y) \geq t + \bar{y} - d(x, y)$. Como $o \leq x < y$, la expresión anterior es mayor o igual que t.

En caso contrario, es decir, si en el momento t apareciera al menos un requerimiento a cada lado del servidor, en ambos extremos habría al menos un requerimiento cuyo tiempo de presentación es t, es decir, sería $\hat{y}_t \geq \hat{x}_t \geq t$. Según la expresión 4.4.6, sería $w \geq 2\min\{\hat{x}_t, \hat{y}_t\} + d(s,y) \geq 2\min\{\hat{x}_t, \hat{y}_t\} \geq 2t \geq t$. □

Teorema 4.30. *Si en el momento t `WBB` calcula una espera w en una transición* esperando → esperando*, entonces $w \geq t$.*

Demostración. Que la transición sea de *esperando* a *esperando* significa que en el momento t se presenta al menos un requerimiento en una posición distinta a la del servidor. Tal requerimiento puede determinar un extremo o no. Pero si no determinara ningún extremo, la nueva espera coincidiría con la calculada por última vez. Y como en el momento t el servidor está *esperando*, w todavía no fue alcanzado. Consideraremos entonces que al menos uno de los requerimientos presentados en el momento t determina un extremo que puede coincidir o no con alguno de los extremos que existían previamente.

Analizaremos por separado dos posibles situaciones:

1. Supongamos en primer lugar que $\max\{\hat{x}_t, \hat{y}_t\} \geq \min\{\hat{x}_t, \hat{y}_t\} + d(x,y)$. El Lema 4.25 afirma que $w \geq 2\max\{\hat{x}_t, \hat{y}_t\} - (d(s,x) + d(x,y))$. Puesto que al menos uno de los requerimientos pendientes en algún extremo se presentó en el momento t, tenemos que $\max\{\hat{x}_t, \hat{y}_t\} \geq t$. Considerando esto y la hipótesis de esta situación, podemos decir que $w \geq t + \min\{\hat{x}_t, \hat{y}_t\} - d(s,x)$. Ahora, usando el Lema 4.28, concluimos que $w \geq t + \bar{s} - d(s,x) \geq t$.

2. La situación contraria es que $\max\{\hat{x}_t, \hat{y}_t\} < \min\{\hat{x}_t, \hat{y}_t\} + d(x,y)$. En este caso, utilizaremos el Lema 4.25 para afirmar que $w \geq 2\min\{\hat{x}_t, \hat{y}_t\} + d(s,y)$. Y, nuevamente con el Lema 4.28, llegamos a que $w \geq \bar{s} + \min\{\hat{x}_t, \hat{y}_t\} + d(s,y) \geq \min\{\hat{x}_t, \hat{y}_t\} + d(x,y) > \max\{\hat{x}_t, \hat{y}_t\} \geq t$.

□

Teorema 4.31. *Si en el momento t `WBB` calcula una espera w en una transición* en movimiento → esperando*, entonces $w \geq t$.*

Demostración. Puesto que en el momento t `WBB` sale del estado *en movimiento*, en algún momento anterior pasó de *esperando* a *en movimiento*. Sea t_0 el momento en que `WBB` hizo la última transición *esperando* → *en movimiento* anterior a t. Llamemos m a la sucesión de movimientos en un mismo sentido que realizó el servidor desde el momento t_0 hasta el momento t. Dado que `WBB` no toma ninguna decisión hasta salir del estado *en movimiento*, nos referiremos a m como si fuera un único movimiento para simplificar el análisis.

Además, considerando la cantidad de variables que intervienen en la presente demostración, haremos algunas simplificaciones en la notación para facilitar la lectura. Notaremos con s, x e y a la posición del servidor y la de los extremos en el momento t, y con s_0, x_0 e y_0 a la posición del servidor y la de los extremos vigentes al momento t_0. Siempre que utilicemos las expresiones $\hat{x}_0$ o $\hat{y}_0$, nos estaremos refiriendo a los valores contemplados al calcular la espera que determinó que m comience en el momento t_0. Y cuando usemos las expresiones $\hat{x}$ o $\hat{y}$, nos estaremos refiriendo a los valores que intervienen en el cálculo de la nueva espera w.

Al finalizar un movimiento, `WBB` calcula la espera sólo si queda algún requerimiento pendiente. Podría ocurrir que los requerimientos pendientes en los extremos al finalizar el movimiento m se hayan presentado antes de comenzarlo. En este caso, el cálculo que determinó que este movimiento comience en el momento t_0 debió contemplar el tiempo necesario para atender a todos los requerimientos.

Nos ocuparemos, por lo tanto, del caso en que al menos uno de los requerimientos pendientes en el momento t en alguno de los extremos se presentó durante el transcurso del movimiento m, después del momento t_0 ($\max\{\hat{x}, \hat{y}\} > t_0$). Analizaremos dos situaciones principales, de acuerdo a la dirección que tuvo este último movimiento.

1. Supongamos que m fue un movimiento hacia la derecha, es decir, desde el extremo x_0 hasta el extremo y_0. Esto significa que fueron atendidos todos los requerimientos que estaban pendientes en el momento t_0.

   (a) Si, al finalizar el movimiento m, el servidor descubre que tiene requerimientos pendientes a ambos lados, indudablemente estos requerimientos se presentaron cuando el servidor ya estaba en movimiento. Es decir que tanto $\hat{x}$ como $\hat{y}$ son valores mayores que t_0. Por el Lema 4.25 sabemos que $w \geq 2\min\{\hat{x}, \hat{y}\} + d(s, y) > 2t_0$ y, combinando esta desigualdad con el Lema 4.24, obtenemos que $w > t_0 + d(x_0, y_0) = t_0 + d(s_0, s) = t$.

   (b) Caso contrario, en el momento t el servidor se encuentra en un extremo. Distinguiremos dos situaciones:

      i. $\max\{\hat{x}, \hat{y}\} \geq \min\{\hat{x}, \hat{y}\} + d(x, y)$. En esta situación, la Definición 4.4.7 establece que la espera es $w \geq 2\max\{\hat{x}, \hat{y}\} - d(x, y) > t_0 + \min\{\hat{x}, \hat{y}\}$.

      ii. $\max\{\hat{x}, \hat{y}\} < \min\{\hat{x}, \hat{y}\} + d(x, y)$. Aquí, podemos usar la misma Definición 4.4.7) para notar que $w \geq 2\min\{\hat{x}, \hat{y}\} + d(x, y) > \max\{\hat{x}, \hat{y}\} + \min\{\hat{x}, \hat{y}\} > t_0 + \min\{\hat{x}, \hat{y}\}$.

   Como vemos, en cualquiera de estas situaciones ocurre que $w > t_0 + \min\{\hat{x}, \hat{y}\}$. Y, utilizando el Lema 4.28, llegamos a que $w > t_0 + \bar{s} \geq t_0 + d(s_0, s) = t$.

2. Supongamos ahora que m fue un movimiento hacia la izquierda. Es decir que en el momento t el servidor online llega a x_0. Puesto que $t_0 \geq \bar{s}_0 > \bar{x}_0$ (porque el servidor se mueve a velocidad unitaria), en caso de haberse presentado un nuevo requerimiento a la izquierda de x_0 en un momento posterior a t_0, sería $\hat{x} > t_0 > \hat{x}_0$.

También puede variar el extremo derecho. Si el nuevo extremo derecho coincide o está más alejado que el anterior ($\bar{y} > \bar{y}_0$), es claro que $\hat{y} > \hat{y}_0$. Pero dado que el servidor se mueve de derecha a izquierda, podría suceder que m haya comenzado en y_0 y que el nuevo extremo derecho esté más cerca del origen ($\bar{y} < \bar{y}_0$). Claro está, esto sólo es posible si en el momento t_0 el servidor parte desde y_0. Usando nuevamente como argumento a la velocidad unitaria, podemos decir que $t_0 \geq \bar{y}_0$. Y como el nuevo requerimiento se presenta después de t_0 mientras que el del extremo anterior se presentó antes, tenemos $\hat{y} > t_0 \geq \hat{y}_0$.

Combinando los resultados de los dos últimos párrafos, podemos concluir que $\min\{\hat{x}, \hat{y}\} \geq \min\{\hat{x}_0, \hat{y}_0\}$. Y, utilizando el Lema 4.28, obtenemos la siguiente desigualdad:

$$\min\{\hat{x}, \hat{y}\} \geq \bar{s}_0 \tag{4.4.9}$$

Ahora también distinguiremos dos situaciones:

(a) $\max\{\hat{x}, \hat{y}\} \geq \min\{\hat{x}, \hat{y}\} + d(x, y)$. En esta situación, el Lema 4.25 asegura que $w \geq 2\max\{\hat{x}, \hat{y}\} - d(s, x) - d(x, y) > t_0 + \min\{\hat{x}, \hat{y}\} - d(s, x)$.

(b) $\max\{\hat{x}, \hat{y}\} < \min\{\hat{x}, \hat{y}\} + d(x, y)$. Aquí usamos el mismo Lema 4.25 para afirmar que $w \geq 2\min\{\hat{x}, \hat{y}\} + d(s, y) = 2\min\{\hat{x}, \hat{y}\} + d(x, y) - d(s, x)$. Considerando la hipótesis de esta situación, podemos notar que $w > \max\{\hat{x}, \hat{y}\} + \min\{\hat{x}, \hat{y}\} - d(s, x) > t_0 + \min\{\hat{x}, \hat{y}\} - d(s, x)$.

Como vimos, en ambas situaciones sucedió que $w > t_0 + \min\{\hat{x}, \hat{y}\} - d(s, x)$. Si combinamos esta expresión con la desigualdad 4.4.9, obtenemos que $w > t_0 + \bar{s}_0 - d(s, x) \geq t_0 + d(s_0, s) = t$.

□

Ahora que sabemos que `WBB` obtiene siempre soluciones factibles y calcula esperas válidas, sólo nos resta enunciar su competitividad.

Teorema 4.32. *`WBB` es una estrategia 2-competitiva para la variante Nomadic de DOLTSP en semicadenas, tanto cuando el adversario es* fair *como cuando es* standard*.*

Demostración. En los Teoremas 4.29, 4.30 y 4.31 mostramos que si en un momento t se calcula una espera w, resulta $w \geq t$. Además, como vimos en la Sección 4.2, si el servidor comienza a moverse en el momento $w = \rho\ Z^* - \alpha$, termina su recorrido exactamente en el momento ρZ^*. Finalmente, como $\rho = 2$ y Z^* (tal como la definimos en la expresión 4.4.5) es una cota para el costo óptimo offline, concluimos que para toda secuencia de requerimientos σ, vale que $WBB(\sigma) = \rho\ Z^* \leq 2\ OPT(\sigma)$.

□

4.5 Otros algoritmos online

En esta sección definimos algunos algoritmos adicionales para los que no haremos un análisis de competitividad como hicimos con las estrategias presentadas hasta ahora. Nos limitaremos a estudiar su comportamiento en forma empírica, comparando los resultados con los de `Zig-Zag`, `Replan`, `WBR` y `WBB`. Al contar con un conjunto más grande y variado de estrategias, el análisis empírico que expondremos en el Capítulo 6 resultará aún más provechoso.

Como primer paso, extendemos la estrategia `WBB` para cadenas completas y para que pueda funcionar también en la variante Homing. Luego presentamos dos estrategias que pueden concebirse como variaciones de otras que ya definimos: `Statistic WBB` y `Statistic Replan`. La primera, al igual que `WBB`, es cautelosa y consiste en esperar antes de ejecutar cada movimiento. Pero en lugar de calcular la espera para lograr una competitividad predeterminada, el servidor comienza a moverse recién cuando el "riesgo" de realizar el próximo movimiento esté por debajo de cierto umbral. La segunda es una variación de `Replan` que es válida sólo para la variante Nomadic y puede resumirse de la siguiente manera: cada vez que no quedan requerimientos pendientes, en vez de quedarse en su lugar, el servidor se dirige al vértice que consideramos más conveniente para atender nuevos requerimientos que eventualmente puedan presentarse.

4.5.1 WBB en cadenas completas

Definimos `WBB` para la variante Nomadic en cadenas completas de manera similar a la versión para semicadenas: el servidor se mantiene inmóvil mientras se presentan requerimientos y comienza un recorrido para atenderlos recién cuando el costo previsto para el servidor online es exactamente el doble que la cota calculada para el costo óptimo offline. No hay ninguna variación en el diagrama de estados y usamos la misma cota para el costo óptimo offline, expresada en 4.4.5, que se basa en la Definición 4.22 de extremos.

El único cambio radica en la forma de determinar el recorrido que le resta transitar al servidor online para atender a los requerimientos pendientes. Como se trata de cadenas completas, elegimos ir primero al extremo más cercano al servidor y, una vez allí, al otro extremo. Esto altera el valor de α y el consiguiente cálculo de la espera. Entonces, si en cierto momento t notamos con s, x e y a la posición del servidor y la de los extremos, podemos calcular el tiempo que falta para finalizar el recorrido como $\alpha = \min\{d(s,x), d(s,y)\} + d(x,y)$ y, por lo tanto, la espera finaliza en el momento

$$w = \max \begin{cases} 2\max\{\hat{x}_t, \hat{y}_t\} - \min\{d(s,x), d(s,y)\} - d(x,y) \\ 2\min\{\hat{x}_t, \hat{y}_t\} - \min\{d(s,x), d(s,y)\} + d(x,y) \end{cases}$$

En la Sección 4.4 ya mencionamos esta forma de determinar el valor de α, pero la descartamos porque la versión para semicadenas de `WBB` dejaba de ser 2-competitiva. Para que lo fuera, decidimos que el servidor se dirija primero al extremo más cercano al origen, aunque eso significara recorrer una

distancia mayor. A pesar de que esa solución resultó apropiada según el análisis de competitividad, no parece una idea muy intuitiva, ya que si queremos que el servidor termine su trabajo lo antes posible, lo más razonable es que recorra la distancia más corta. Como nuestro objetivo ahora es presentar algoritmos más intuitivos, al definir `WBB` para cadenas completas nos inclinamos por esta última idea. En los casos en que el cálculo de la espera determine un momento anterior al momento en que se calcula ($w < t$), el servidor comenzará de inmediato con el recorrido restante, a riesgo de que el costo final sea más de dos veces el óptimo.

En resumen, cuando se trate de semicadenas, `WBB` calculará el recorrido restante tal como vimos en la Sección 4.4: dirigiendo el servidor primero al extremo izquierdo (más cercano al origen) y luego al extremo derecho. En cambio, cuando se trate de cadenas completas, se visitará primero el extremo más cercano al servidor y luego el otro extremo.

4.5.2 WBB para la variante Homing

Como ya dijimos, es posible extender `WBB` para que funcione también en cadenas completas para Homing DOLTSP, utilizando el mismo diagrama de estados que vimos en la Sección 4.4. Para ello, basta redefinir el recorrido del servidor y el cálculo de la espera a partir de los valores α, Z^* y ρ.

Determinaremos el recorrido del servidor online a partir de la Definición 4.22 de extremos, que usamos para la versión de `WBB` en semicadenas para la variante Nomadic. Pero ahora, como estamos definiendo el comportamiento para la variante Homing, entre los vértices que debe visitar el servidor para finalizar su servicio también incluimos al origen. Con lo cual, si notamos con x e y a los extremos, con o al origen y con s al vértice donde se encuentra el servidor al momento de calcular una espera, valen a la vez $x \leq s \leq y$ y $x \leq o \leq y$.

Teniendo en cuenta estas consideraciones, el recorrido del servidor online puede ser descripto del mismo modo que para la estrategia `Replan`:

1. Si el servidor está en el origen, ir a un extremo cualquiera. Si está fuera, ir al extremo más alejado del origen que esté en la misma semicadena que el servidor.
2. Ir al otro extremo.
3. Ir al origen.

Dado que $x \leq s \leq y$, podemos calcular el costo de este recorrido como $\alpha = 2d(x, y) - \bar{s}$.

El valor de Z^* también sufre algunos cambios. Recordemos que, tal como vimos en la Sección 4.4, todos los vértices que en algún momento son extremos forman parte del recorrido óptimo offline. Por lo tanto, podemos acotar el costo óptimo por:

i. el costo de ir a un extremo, atender los pedidos que se hayan presentado allí y regresar al origen

ii. el costo de ir a un extremo, atender los pedidos que se hayan presentado allí, ir al otro extremo y regresar al origen

De acuerdo a esto, podemos establecer la siguiente cota inferior para el costo óptimo offline:

$$Z^* = \max \begin{cases} \max\{\hat{x}_t + \bar{x}, \hat{y}_t + \bar{y}\} \\ \min\{\hat{x}_t + \bar{y}, \hat{y}_t + \bar{x}\} + d(x, y) \end{cases}$$

Finalmente, la competitividad que queremos alcanzar ya no es 2, puesto que ésta surgió de la cota inferior hallada en el Capítulo 3 para la variante Nomadic contra un adversario *standard*. En este caso, tomaremos como referencia a la cota que mostramos en el Teorema 3.3 para Homing DOLTSP contra un adversario *standard*, es decir,

$$\rho = \frac{2 + \sqrt{2}}{2} \approx 1.707$$

Para definir la espera, sólo resta remitirnos a la expresión 4.2.1, reemplazando α, Z^* y ρ por los valores que acabamos de describir. Al igual que en la versión de `WBB` para cadenas completas en la variante Nomadic, si al calcular una espera en el momento t resulta $w < t$, entonces el servidor comenzará a moverse de inmediato, aunque el cociente entre el costo final de su recorrido y el costo óptimo offline sea mayor a ρ.

4.5.3 Statistic WBB

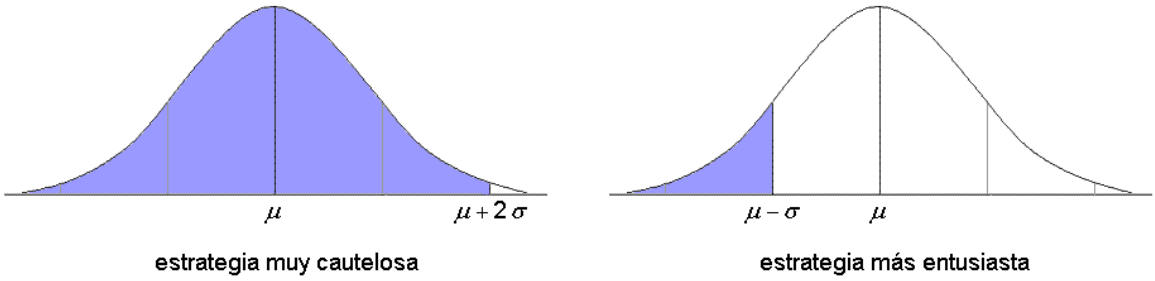
Manteniendo la idea de esperar antes de comenzar a mover el servidor, podemos pensar en otra forma de calcular la espera, que no esté condicionada por una competitividad a alcanzar. En lugar de tener como referencia al costo óptimo offline, `Statistic WBB` determina el tiempo que el servidor debe permanecer inmóvil estimando el momento en que se presentará un nuevo requerimiento.

Con este objetivo, asumimos como hipótesis que el tiempo que tarda en presentarse un nuevo requerimiento a partir de la última presentación está regido por una variable aleatoria con distribución normal que llamamos Ψ. Para cada requerimiento $r_i = (t_i, x_i)$, el valor de Ψ_i indica cuánto tiempo después de t_i tarda en presentarse un nuevo requerimiento, si es que r_i no es el último de la secuencia. El valor de Ψ_i es desconocido hasta que se presenta el requerimiento r_{i+1}. Entonces, para decidir cuánto debe esperar el servidor, estimaremos la media (μ) y la varianza (σ^2) utilizando como muestra a los valores de los Ψ_j con $j < i$, que ya son conocidos.

Como la frecuencia de presentación de pedidos puede ir variando a lo largo del tiempo, fijamos un valor k que consideramos razonable y sólo tomamos los tiempos de presentación de los últimos k requerimientos para estimar la media y la varianza de Ψ. En nuestra implementación, hemos fijado el valor $k = 20$.

De acuerdo a este modelo, podemos afirmar con cierta confianza que, si r_i no es el último requerimiento, lo más probable es que el próximo pedido se presente en un momento cercano a $t_i + \mu$. En otras palabras, si no se presenta un nuevo requerimiento pasado ese momento, lo más probable es que r_i haya sido el último. Es oportuno recordar que la motivación que habíamos identificado para demorar el movimiento del servidor está relacionada con la anticipación a nuevos requerimientos. En este sentido, suponer que no habrá nuevos pedidos es un muy buen argumento para suspender la espera y proseguir con el recorrido.

Ahora bien, ¿cuándo estamos suficientemente seguros de que no se presentarán más requerimientos? Haber asumido que Ψ tiene una distribución normal nos ayuda a relativizar el concepto de algoritmos entusiastas y cautelosos, determinando el nivel de certidumbre que mejor se adecúe a nuestras expectativas. Por ejemplo, una estrategia *muy cautelosa* comenzará a mover el servidor recién cuando transcurra $\mu + 2\sigma$ desde la aparición del último requerimiento, puesto que la probabilidad de que no haya más requerimientos es casi del 98% [1]. Por el contrario, una estrategia *más entusiasta*, podría decidir esperar sólo $\mu - \sigma$, con una probabilidad cercana al 16% de que no haya nuevos requerimientos. Como vemos, este modelo nos permite parametrizar la estrategia según el *nivel de cautela* que consideremos más apropiado.



estrategia muy cautelosa — estrategia más entusiasta

Otra característica que incorporamos en `Statistic WBB` es la capacidad de modificar el recorrido en cada vértice, del mismo modo que lo hace `Replan`. Con esta decisión, intentamos diseñar una estrategia con suficiente capacidad de reacción ante los cambios y, a la vez, con la cautela necesaria como para no incurrir en los mismos riesgos que las estrategias entusiastas.

A partir de esto último, surge una nueva pregunta: ¿tiene siempre el mismo riesgo abandonar un vértice? Ciertamente, la respuesta es no. Si el vértice hacia donde se dirige el servidor está muy lejos, el costo del movimiento es alto. En estos casos, no resulta nada agradable descubrir, al completar el movimiento, que éste no fue provechoso. Sin embargo, si el eje que tiene que cruzar el servidor es suficientemente corto, quizás no valga la pena dudarlo demasiado puesto que, en caso de equivocarnos, la vuelta atrás no es muy costosa. Tiene sentido entonces que el nivel de cautela no sea el mismo a lo largo de todo el recorrido, sino que esté determinado en función de la longitud del eje que está por atravesar el servidor en cada paso.

[1] El lector puede encontrar una tabla de *áreas de la curva normal estándar* en [Dev98].

Una forma de ponderar esta longitud surge de comparar el costo de atravesar el próximo eje contra el costo que tiene atravesar de lado a lado el espacio "conocido", compuesto por el origen y los vértices donde ya se presentaron requerimientos. Buscaremos mayor cautela cuando estos costos sean similares y mayor entusiasmo en la medida que el próximo paso sea pequeño en comparación con el tamaño del espacio "conocido". Concretamente, cada vez que el servidor comandado por `Statistic WBB` llega a un vértice, espera hasta el momento $w = t_i + \mu + \lambda\sigma$, donde t_i es el momento en que se presentó el último requerimiento y λ determina el nivel de cautela a partir de la longitud relativa del próximo paso, como se muestra en la siguiente tabla.

longitud relativa del próximo paso	nivel de cautela	λ
hasta 0.1	10%	-1.28
hasta 0.2	20%	-0.84
hasta 0.3	30%	-0.52
hasta 0.4	40%	-0.25
hasta 0.5	50%	0.00
hasta 0.6	60%	0.25
hasta 0.7	70%	0.52
hasta 0.8	80%	0.84
hasta 0.9	90%	1.28
hasta 1.0	98%	2.00

Nivel de cautela en función de la longitud relativa del próximo paso

Al igual que en las dos estrategias anteriores, en los casos que al calcular la espera en un momento t resulte $w < t$, el servidor comenzará a moverse sin ninguna espera.

De este modo queda definida `Statistic WBB`, una estrategia cautelosa en la que, antes de comenzar cada movimiento, el servidor espera un tiempo que es determinado en función de la historia conocida. Esta estrategia es válida para las variantes Homing y Nomadic de DOLTSP y para cualquier tipo de cadenas. El recorrido a seguir es simplemente el más corto que le permita al servidor terminar su servicio y se calcula de la misma forma que mostramos para la estrategia `Replan` en la Sección 4.1.

4.5.4 Statistic Replan

Habíamos comentado en el Capítulo 3 que en la variante Nomadic de DOLTSP las estrategias no pueden dar por seguro ningún destino final para el recorrido del servidor, lo que supone un mayor grado de incertidumbre respecto a la variante Homing. Con `Statistic Replan` nos enfocamos en esta cuestión. Aquí no usamos la estadística para calcular un tiempo de espera (puesto que

se trata de una estrategia completamente entusiasta), sino para determinar el vértice hacia donde debe dirigirse el servidor al finalizar un recorrido para la variante Nomadic.

Mientras hay requerimientos pendientes, el servidor se comporta exactamente de la misma forma que se definió para la versión original de `Replan`. La diferencia es que, al terminar el recorrido, el servidor no se queda quieto, sino que se desplaza hacia el vértice donde esperamos minimizar el costo adicional que tendría el servidor si aparecieran nuevos requerimientos.

¿Cómo determinamos ese vértice? Lo que hacemos es asumir que las posiciones donde se presentan los requerimientos mantienen cierta distribución a lo largo del tiempo. Entonces, resulta natural que el servidor se dirija hacia el vértice más cercano al promedio de estas posiciones. Sin embargo, considerar que las posiciones de todos los requerimientos se distribuyen siempre de la misma manera puede resultar una premisa muy restrictiva. Una forma de atenuar esta limitación consiste en fijar un número k y reducir la muestra a los últimos k requerimientos presentados. Al igual que antes, elegimos el valor $k = 20$.

`Statistic Replan` es válida sólo para la variante Nomadic de DOLTSP y sirve tanto para semicadenas como para cadenas completas.

Capítulo 5

Modelo de simulación

En el Capítulo 2 definimos a una instancia de DOLTSP como una secuencia de requerimientos que se presentan a lo largo del tiempo, solicitando que el servidor visite distintos vértices de cierto grafo. En el capítulo anterior, presentamos algunas estrategias para este problema que son válidas sólo en cadenas, es decir, grafos en los que el servidor sólo puede desplazarse en dos direcciones, que denominamos *izquierda* y *derecha*. El objetivo en este capítulo es definir un modelo para estudiar empíricamente el comportamiento de estas estrategias.

5.1 Atributos de una instancia

A fin de construir un conjunto de instancias definidas sobre cadenas que sea suficientemente representativo, las clasificamos de acuerdo a ciertos atributos que abarcan dos aspectos: algunos describen la forma de la cadena y otros se refieren a la secuencia de requerimientos.

Hemos clasificado a las cadenas según tres dimensiones:

1. **Cantidad de vértices.** La cantidad de vértices indica en cuántos puntos de la cadena pueden aparecer requerimientos o puede cambiar el recorrido del servidor. En realidad, si no delimitamos el espacio en que se encuentran los vértices, esta medida puede ser poco descriptiva. Por lo tanto, analizaremos siempre cadenas con una misma longitud, fijada de antemano. De esta forma, podemos relacionar la cantidad de vértices con la *densidad* de la cadena y con la libertad que tiene el servidor para modificar su recorrido.

2. **Tipo de cadena.** El origen es el vértice donde comienza el recorrido del servidor y, en la variante Homing, a donde debe retornar finalmente. Aquí distinguimos las cadenas en las que el origen está más cerca de un extremo que del otro, de aquellas donde las semicadenas positiva y negativa tienen longitudes similares. Este atributo tiene particular interés si consideramos

que hay estrategias que están definidas únicamente para cadenas en las que el origen coincide con uno de los extremos, es decir, semicadenas.

3. **Distribución de vértices.** La *cantidad de vértices*, tal como la definimos, sólo permite describir la densidad *promedio* de una cadena. Pero lo cierto es que, en nuestro modelo, no hay ninguna restricción en cuanto a la disposición que deben tener los vértices dentro de la cadena. Este nuevo atributo describe si hay sectores de la cadena más densos que otros y, en ese caso, en qué posiciones se concentran los vértices.

Haber fijado una distancia entre los extremos de la cadena nos permite, también, acotar el conjunto de secuencias de requerimientos que son significativas para nuestro análisis. Especialmente, en lo que se refiere a los tiempos de presentación. Por ejemplo, dado que la velocidad del servidor es constante y unitaria, si se presentaran requerimientos en momentos desproporcionadamente tardíos en relación a la longitud de la cadena, el recorrido del servidor dejaría de representar un problema, puesto que el costo de cualquier solución estaría sujeto, principalmente, al momento en que se presenta el último requerimiento. Entonces, a fin de analizar instancias en las que podamos diferenciar claramente los costos de las distintas soluciones, debemos mantener una relación entre la longitud de la cadena y el rango temporal en que se presentan los requerimientos. Para esto, descartaremos del análisis a las instancias en las que aparezcan pedidos después de determinado momento, acorde a la longitud de la cadena.

Al quedar acotado el tiempo en que pueden presentarse requerimientos, también parece razonable limitar el tamaño de la secuencia de pedidos. Es decir, considerando que analizamos cadenas con un conjunto finito de vértices, si la cantidad de requerimientos fuera excesivamente grande respecto a la *cantidad de vértices*, probablemente aparecerían en un mismo vértice muchos más pedidos de los que son necesarios para solicitar la visita del servidor.

Además de especificar el rango temporal y la cantidad de requerimientos, debemos decidir cómo se distribuyen los pedidos en el tiempo y el espacio. En definitiva, respecto a la secuencia de requerimientos, hemos considerado de interés otros cuatro atributos:

4. **Cantidad de requerimientos.** Si se presentan muchos o pocos requerimientos, respecto a la *cantidad de vértices*.

5. **Distribución espacial de requerimientos.** Cómo se distribuyen los requerimientos en cada sector de la cadena.

6. **Rango temporal.** Intervalo de tiempo durante el cual pueden presentarse requerimientos, ligado al tamaño del espacio.

7. **Distribución temporal de requerimientos.** Cómo se distribuyen los requerimientos dentro del *rango temporal*.

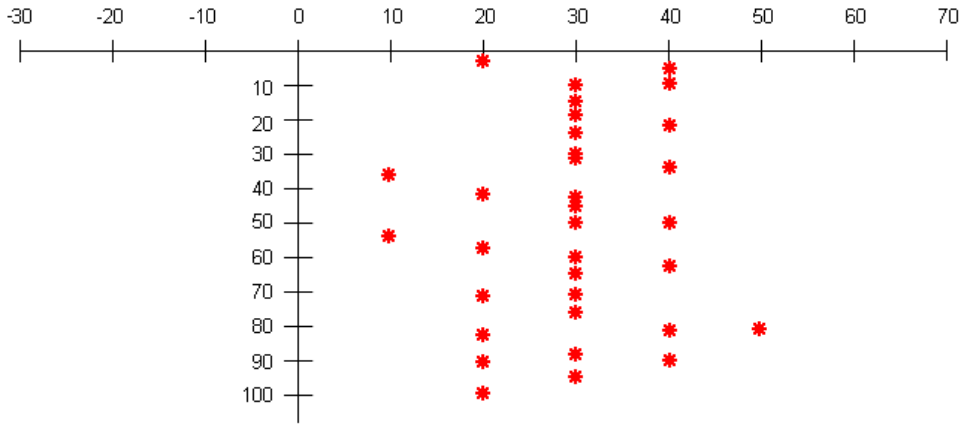
Con el fin de tomar una muestra del conjunto de instancias que sea significativa para nuestro análisis empírico, asignaremos a cada uno de estos atributos algunos valores que consideramos representativos. Llamamos *meta-instancia* a cada configuración de estos atributos. Como veremos luego, en algunos atributos intervienen distribuciones aleatorias, por lo que una meta-instancia puede estar relacionada con muchas instancias concretas. Para esclarecer este concepto, veamos un par de ejemplos de meta-instancias en una cadena donde la distancia entre los extremos es de 100 unidades.

Ejemplo 1.

Meta-instancia

Cantidad de vértices:	11
Tipo de cadena:	Cadena asimétrica (-30 / 70)
Distribución de vértices:	Equidistante, entre -30 y 70
Cantidad de requerimientos:	33 (tres veces la *cantidad de vértices*)
Distribución espacial de requerimientos:	Normal ($\mu = 30, \sigma = 10$)
Rango temporal:	Igual a la longitud de la cadena
Distribución temporal de requerimientos:	Uniforme, entre 0 y 100

Instancia concreta

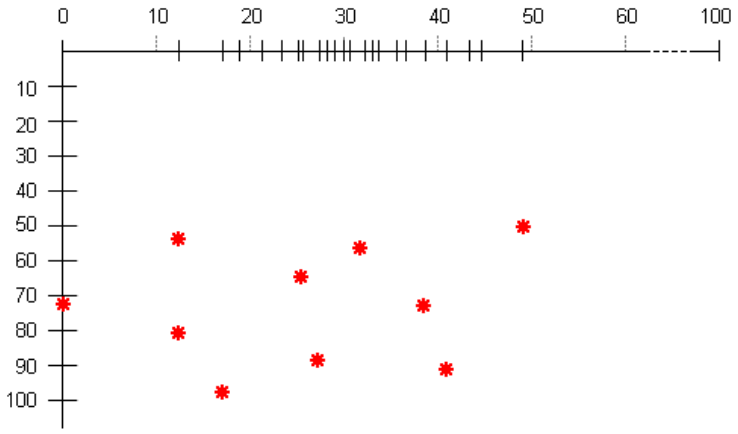


Ejemplo 2.

Meta-instancia

Cantidad de vértices:	20
Tipo de cadena:	Semicadena (0 / 100)
Distribución de vértices:	Normal ($\mu = 30, \sigma = 10$)
Cantidad de requerimientos:	10 (la mitad de la *cantidad de vértices*)
Distribución espacial de requerimientos:	Uniforme, entre 0 y 50
Rango temporal:	Igual a la longitud de la cadena
Distribución temporal de requerimientos:	Uniforme, entre 50 y 100

Instancia concreta



Como podemos ver, el número de meta-instancias posibles es infinito, como así también el número de instancias relacionadas con cada meta-instancia. En la siguiente sección detallaremos el conjunto de meta-instancias que consideramos de interés para nuestra simulación. Pero antes, haremos una breve reseña sobre las distintas distribuciones que contemplamos y describiremos para qué atributos utilizamos cada una de ellas.

1. **Equidistante.** Aquí, los valores están dispuestos de modo tal que, si los ordenamos, cada par de valores consecutivos está separado por la misma distancia. Utilizamos esta distribución (no aleatoria) tanto para determinar la *distribución de vértices* como para precisar la *distribución temporal de requerimientos*.

2. **Uniforme.** Según esta distribución, todos los valores de cierto conjunto tienen la misma probabilidad de ocurrencia. Debido a su neutralidad, la consideramos para definir valores de *distribución de vértices*, *distribución espacial de requerimientos* y *distribución temporal de requerimientos*.

3. **Normal.** Esta es la familia de distribuciones más frecuentemente utilizada para realizar análisis estadísticos y probabilísticos en campos muy diversos. Se trata de distribuciones

simétricas, donde la *media* (μ) y la *varianza* (σ^2) de la población son parámetros conocidos. Se la conoce también por la forma de campana que tiene la gráfica asociada a la función de probabilidad puntual. De todas las distribuciones con una media y una varianza conocidas, la normal es la que tiene mayor entropía y, por lo tanto, también puede considerarse, en cierta medida, neutral. La utilizamos para determinar la *distribución de vértices*, la *distribución espacial de requerimientos* y la *distribución temporal de requerimientos*, rechazando aquellos valores que no pertenezcan al rango establecido para cada atributo.

4. **Poisson.** A diferencia de las distribuciones normal o uniforme, la distribución de Poisson se aplica a variables aleatorias discretas. Una de sus aplicaciones más importantes se relaciona con el acontecimiento de eventos a lo largo del tiempo, en lo que se conoce como *proceso de Poisson* [Dev98]. Básicamente, esta distribución describe la probabilidad que tiene un evento de ocurrir una determinada cantidad de veces en un intervalo de tiempo establecido. En nuestro caso, el evento que nos interesa representar es la aparición de un requerimiento, así que la usaremos en el atributo *distribución temporal de requerimientos*, simulando cómo se comportarían las estrategias si viéramos a la presentación de pedidos como un proceso de Poisson.

5.2 Espacio muestral

Como ya dijimos, el conjunto de instancias posibles para DOLTSP es infinito. Por este motivo analizaremos sólo una muestra que procuraremos sea suficientemente representativa. Con este fin, detallamos los valores que elegimos para cada uno de los atributos antes expuestos. En total, la muestra está formada por 7506 meta-instancias, todas ellas definidas para cadenas de 100 unidades de longitud. A continuación, esta cantidad aparecerá desglosada entre los valores que correspondan a cada atributo.

5.2.1 Cantidad de vértices

Tomando a la longitud de la cadena como referencia, consideramos que son representativos tres valores para la *cantidad de vértices*: 10, 30 y 100. Para cada uno de estos valores, la cantidad resultante de meta-instancias es la misma.

Intuitivamente, podríamos esperar que las soluciones generadas por una estrategia como `Replan`, que calcula el mejor recorrido cada vez que el servidor llega a un vértice, sean mejores cuanto más densa es la cadena. Sin embargo, esta densidad no debería afectar de igual manera al comportamiento de una estrategia como `Zig-Zag`, que modifica el recorrido sólo cuando el servidor está en una posición donde acaba de servir un requerimiento.

Más del 99% de las meta-instancias que construimos responden a la clasificación que acabamos de describir y las distinguiremos como Grupo A. Con el 1% restante, analizamos dos clases particulares de cadenas, en las que reducimos al mínimo la libertad que tiene el servidor para cambiar de dirección. Estamos hablando de semicadenas con un solo vértice además del origen (dos en total) y de cadenas que tienen un único vértice a cada lado del origen (tres en total). Nos referiremos a estas meta-instancias como Grupo B.

La siguiente tabla muestra un resumen con los distintos valores que toma este atributo e incluye la cantidad de meta-instancias asociadas a cada uno de ellos.

cantidad de vértices	meta-instancias	
	A	B
2	-	27 (50%)
3	-	27 (50%)
10	2484 (33,3%)	-
30	2484 (33,3%)	-
100	2484 (33,3%)	-

5.2.2 Tipo de cadena

Hemos considerado de interés tres tipos de cadenas:

i. *semicadenas*, donde el origen coincide con un extremo,

ii. *cadenas asimétricas*, donde el origen está más cerca de un extremo que del otro, y

iii. *cadenas simétricas*, donde la distancia que hay del origen a cada uno de los extremos es la misma.

Dado que la longitud que establecimos para las cadenas es de 100 unidades, determinamos una posición mínima y otra máxima para los vértices. En la tabla que se muestra a continuación detallamos el mínimo y máximo para cada *tipo de cadena*.

tipo de cadena	min	max	meta-instancias	
			A	B
semicadena	0	100	2430 (32,6%)	27 (50%)
cadena asimétrica	-25	75	3402 (45,7%)	-
cadena simétrica	-50	50	1620 (21,7%)	27 (50%)

5.2.3 Distribución de vértices

Aquí usaremos tres de las distribuciones que nombramos en la Sección 5.1. Las distribuciones *equidistante* y *uniforme*, representan a aquellos casos donde la densidad de vértices es similar a lo largo de la cadena. En la primera, la distancia entre cada par de vértices vecinos es siempre la misma, mientras que en la segunda, es una variable aleatoria.

Usamos la distribución *normal* para representar a aquellas cadenas donde existen zonas con mayor acumulación de vértices. De todas las formas posibles en que pueden acumularse vértices, seleccionamos, en representación, cinco configuraciones posibles:

i. vértices acumulados en el centro de la cadena.

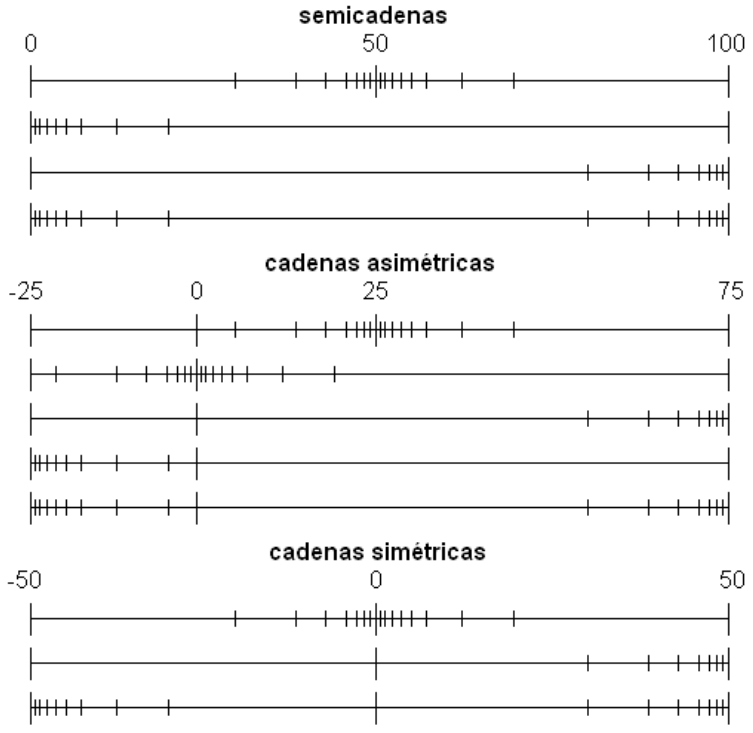
ii. vértices acumulados en el origen.

iii. vértices acumulados en el extremo derecho.

iv. vértices acumulados en el extremo izquierdo.

v. vértices acumulados en ambos extremos.

Como podemos ver en el gráfico que mostramos a continuación, según el *tipo de cadena* del que se trate, algunas de estas configuraciones se solapan. Además, para no realizar dos veces el mismo análisis, en la cadena simétrica sólo evaluaremos la acumulación en uno de los extremos.



La siguiente tabla resume los distintos valores que puede tomar el atributo *distribución de vértices*.

acumulación	tipo de cadena	distribución	μ	σ	meta-instancias	
					A	B
sin acumulación	semicadena	equidistante	-	-	405 (5,4%)	27 (50%)
	cadena asimétrica		-	-	486 (6,5%)	-
	cadena simétrica		-	-	324 (4,4%)	27 (50%)
	semicadena	uniforme	-	-	405 (5,4%)	-
	cadena asimétrica		-	-	486 (6,5%)	-
	cadena simétrica		-	-	324 (4,4%)	-
en el centro	semicadena	normal	50	10	405 (5,4%)	-
	cadena asimétrica		25	10	486 (6,5%)	-
en el origen	semicadena		0	10	405 (5,4%)	-
	cadena asimétrica		0	10	486 (6,5%)	-
	cadena simétrica		0	10	324 (4,4%)	-
en el extremo derecho	semicadena		100	10	405 (5,4%)	-
	cadena asimétrica		75	10	486 (6,5%)	-
	cadena simétrica		50	10	324 (4,4%)	-
en el extremo izquierdo	cadena asimétrica		-25	10	486 (6,5%)	-
en ambos extremos	semicadena		0 / 100	10	405 (5,4%)	-
	cadena asimétrica		-25 / 75	10	486 (6,5%)	-
	cadena simétrica		-50 / 50	10	324 (4,4%)	-

5.2.4 Cantidad de requerimientos

Consideramos que tres proporciones distintas son representativas para describir la *cantidad de requerimientos* en función de la *cantidad de vértices*. Los valores que elegimos son:

i. *pocos*: la décima parte de la *cantidad de vértices*.

ii. *suficientes*: igual que la *cantidad de vértices*.

iii. *muchos*: diez veces la *cantidad de vértices*.

Notemos que nuestro análisis contempla, en las meta-instancias del Grupo B, cadenas con muy pocos vértices. Entonces, para garantizar que la secuencia de requerimientos no sea vacía, consideraremos siempre la existencia de al menos un requerimiento, aunque esto no se corresponda con las proporciones que establecimos.

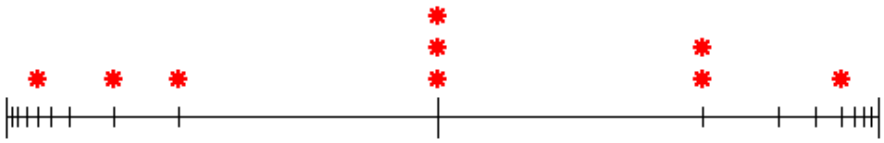
cantidad de requerimientos	meta-instancias	
	A	B
pocos	2484 (33,3%)	18 (33,3%)
suficientes	2484 (33,3%)	18 (33,3%)
muchos	2484 (33,3%)	18 (33,3%)

5.2.5 Distribución espacial de requerimientos

La siguiente tabla muestra las distribuciones que consideramos para este atributo. Esencialmente, son las mismas que hemos considerado para *distribución de vértices*, exceptuando a la distribución equidistante, porque no puede ser implementada sin asumir la misma distribución en los vértices, y esta combinación de atributos no sería muy distinta a las que ya consideramos.

acumulación espacial	tipo de cadena	distribución	μ	σ	meta-instancias	
					A	B
sin acumulación	semicadena	uniforme	-	-	486 (6,5%)	27 (50%)
	cadena asimétrica		-	-	567 (7,6%)	-
	cadena simétrica		-	-	405 (5,4%)	27 (50%)
en el centro	semicadena	normal	50	10	486 (6,5%)	-
	cadena asimétrica		25	10	567 (7,6%)	-
en el origen	semicadena		0	10	486 (6,5%)	-
	cadena asimétrica		0	10	567 (7,6%)	-
	cadena simétrica		0	10	405 (5,4%)	-
en el extremo derecho	semicadena		100	10	486 (6,5%)	-
	cadena asimétrica		75	10	567 (7,6%)	-
	cadena simétrica		50	10	405 (5,4%)	-
en el extremo izquierdo	cadena asimétrica		-25	10	567 (7,6%)	-
en ambos extremos	semicadena		0 / 100	10	486 (6,5%)	-
	cadena asimétrica		-25 / 75	10	567 (7,6%)	-
	cadena simétrica		-50 / 50	10	405 (5,4%)	-

Los requerimientos se presentan sólo en vértices. Por lo tanto, su distribución espacial está relacionada necesariamente con la ubicación de los vértices a lo largo de la cadena. El procedimiento que elegimos para poder analizar de modo independiente las distribuciones de vértices y requerimientos es el siguiente. Obtenemos un conjunto de requerimientos cuyas posiciones respecto al origen obedecen a la distribución especificada, sin considerar las posiciones de los vértices. Luego, ajustamos la posición de cada requerimiento para que coincida con el vértice más cercano. De esta forma, si analizamos por ejemplo una distribución uniforme de requerimientos en una cadena donde los vértices están acumulados en los extremos, obtenemos una instancia similar a la que se muestra en el siguiente gráfico.



Aquí vemos que, a pesar de que la probabilidad de aparición de un requerimiento no es igual en todos los vértices, puede dividirse la cadena en partes iguales de modo de tener en cada una la misma cantidad de requerimientos.

5.2.6 Rango temporal

En la Sección 5.1 definimos el *rango temporal* como el intervalo de tiempo durante el cual pueden presentarse requerimientos, y explicamos que es preferible analizarlo en relación al espacio. Con este objetivo, definimos el *rango espacial* como la distancia del origen al vértice más lejano.

En la simulación analizaremos dos situaciones bien diferenciadas. En la primera, el *rango temporal* coincide con el *rango espacial*. En la segunda, el *rango temporal* es diez veces más grande que el espacial.

rango temporal	meta-instancias	
	A	B
igual al espacial	5796 (77,8%)	42 (77,8%)
mayor al espacial	1656 (22,2%)	12 (22,2%)

El objetivo de elegir estos valores es confirmar empíricamente nuestra intuición de que cuanto más tarde se presente el último requerimiento de la secuencia, más fácil es para las estrategias obtener soluciones eficientes. O, dicho de otro modo, que el análisis de competitividad es excesivamente pesimista si el tiempo en que presentamos los requerimientos no guarda cierta proporción con la longitud de la cadena.

5.2.7 Distribución temporal de requerimientos

Como ya dijimos, probar instancias en las que el *rango temporal* es diez veces mayor que el *rango espacial* tiene un objetivo puntual, para el que no creemos necesario contar con distribuciones muy distintas. A fin de usar una distribución neutral, asumimos que los tiempos de presentación de requerimientos se rigen por una distribución equidistante o uniforme.

En cambio, cuando el *rango temporal* coincide con el *rango espacial*, contemplamos tres formas básicas de distribuir los tiempos de presentación:

1. distribuídos a lo largo del *rango temporal*, sin puntos de acumulación definidos.
2. acumulados al comienzo del *rango temporal*.
3. acumulados al final del *rango temporal*.

En el primer caso, cuando se presentan requerimientos a lo largo de todo el *rango temporal*, analizamos tres distribuciones diferentes: equidistante, uniforme y de Poisson. Cuando debemos acumular los tiempos de presentación al comienzo o al final del *rango temporal*, lo hacemos de dos

modos diferentes: uno regido por una distribución normal y otro en el que los tiempos de presentación se distribuyen uniformemente en un intervalo de tiempo mucho menor al *rango temporal*.

La tabla que sigue resume las distintas situaciones que decidimos considerar. Las columnas *min* y *max* indican el momento más temprano y más tardío en el que se presentan requerimientos, expresados en realación al *rango temporal*. Los valores de las columnas μ y σ son los parámetros de la distribución normal y también se expresan en función del *rango temporal*.

rango temporal	acumulación temporal	distribución	min	max	μ	σ	meta-instancias	
							A	B
igual al espacial	sin acumulación	equidistante	0%	100%	-	-	828 (11,1%)	6 (11,1%)
		uniforme	0%	100%	-	-	828 (11,1%)	6 (11,1%)
		de Poisson	0%	100%	-	-	828 (11,1%)	6 (11,1%)
	al comienzo	uniforme	0%	10%	-	-	828 (11,1%)	6 (11,1%)
		normal	0%	100%	25%	10%	828 (11,1%)	6 (11,1%)
	al final	uniforme	90%	100%	-	-	828 (11,1%)	6 (11,1%)
		normal	0%	100%	75%	10%	828 (11,1%)	6 (11,1%)
mayor al espacial	sin acumulación	equidistante	0%	1000%	-	-	828 (11,1%)	6 (11,1%)
		uniforme	0%	1000%	-	-	828 (11,1%)	6 (11,1%)

5.3 Calidad de las soluciones

La eficiencia empírica de cada estrategia está determinada, básicamente, por la calidad de las soluciones que genera. Una forma simple de medir la calidad de un recorrido que es solución de una instancia particular, surge de la comparación de su costo respecto al costo óptimo offline para esa misma instancia, relación que denominamos *factor de aproximación*. Es decir, dada una instancia σ, el factor de aproximación (FA) de una solución producida por un algoritmo online `ALG` se calcula como

$$FA := \frac{ALG(\sigma)}{OPT(\sigma)}$$

Podemos notar que si `ALG` es un algoritmo ρ-competitivo, entonces ρ es una cota superior para los factores de aproximación de los recorridos que genera `ALG`. Cuanto menor es el FA (más cercano a 1), mejor es la calidad de la solución generada. Este factor nos permite ubicar los resultados de las distintas pruebas en una misma escala, facilitándonos la comparación entre las estrategias.

De la definición de FA surge que, para poder calcularlo, es necesario conocer el costo óptimo offline. En todos los casos tomaremos el costo óptimo considerando un adversario *standard*, incluso para la versión de `WBR` para un adversario *fair*. Esto nos permitirá ubicar los resultados de todas las estrategias en la misma escala.

A continuación mostramos un algoritmo que usamos para obtener el costo óptimo para la variante Homing de DOLTSP en semicadenas y luego otro, más complejo, que utilizamos para calcular el costo óptimo en el resto de los casos.

5.3.1 Algoritmo óptimo de ida y vuelta

Cuando se trata de la variante Homing en semicadenas, podemos obtener una solución óptima desplazando el servidor hasta el vértice más lejano al origen donde se vayan a presentar requerimientos y luego haciéndolo regresar a medida que éstos se presenten.

Formalmente, podemos dividir el proceso en dos etapas:

1. **Alejamiento.** Sea X el vértice más lejano donde se presenta algún requerimiento. Desplazar el servidor hasta X.
2. **Regreso.** Comienza cuando el servidor arriva al vértice X. Entonces repetiremos el siguiente procedimiento hasta que el servidor llegue al origen, comenzando con $x = X$:
   (a) Permanecer en x hasta atender al último requerimiento que se presente en x. Sea t ese momento.
   (b) Si no se presentan requerimientos después de t, asignar a x la posición del origen. En caso contrario, de todos los requerimientos que se presentan después que t, sea r el que se presenta en el vértice más alejado del origen. Asignar a x el vértice donde se presenta r.
   (c) Desplazar el servidor hacia x.

No es difícil verificar que un recorrido generado de esta forma es, en efecto, una solución óptima para la variante Homing de DOLTSP en semicadenas. Para analizar la complejidad de este algoritmo, notemos que el cálculo contiene un ciclo sobre los requerimientos en el que se busca, en cada iteración, el requerimiento pendiente más alejado del origen. Ordenando la lista de requerimientos en forma conveniente, esto puede resolverse en un tiempo lineal respecto a la cantidad de pedidos, que notamos con n. Por lo tanto, concluimos que la complejidad de este algoritmo es $O(n \log n)$.

5.3.2 Algoritmo óptimo recursivo

El procedimiento no es tan sencillo cuando queremos calcular el costo óptimo para la variante Nomadic en semicadenas, o para cualquier variante en cadenas con vértices a ambos lados del origen.

Sea $\sigma = r_1, \ldots, r_n$ una secuencia de requerimientos. Definimos $OPT(\sigma, v, \tau)$ como el costo óptimo offline para atender a todos los requerimientos de σ, partiendo desde el vértice v en el momento τ. El caso base es cuando $\sigma = \emptyset$. En tal situación, el valor de $OPT(\sigma, v, \tau)$ depende de la variante que consideremos.

$$OPT(\emptyset, v, \tau) = \begin{cases} \tau + \bar{v} & \text{para la variante Homing} \\ \tau & \text{para la variante Nomadic} \end{cases}$$

Cuando $\sigma \neq \emptyset$, este valor está determinado por la siguiente recursión.

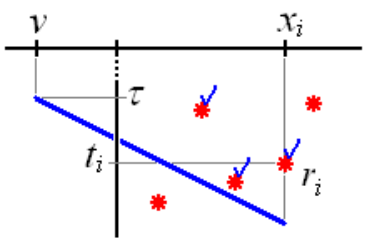
$$OPT(\sigma, v, \tau) = \min_{r_i \in \sigma} \{OPT(\sigma - \{r_i\}, x_i, \max\{t_i, \tau + d(v, x_i)\})\} \tag{5.3.1}$$

Esto es, nos quedamos con el costo del recorrido que termina antes entre todos los recorridos óptimos que surgen de comenzar atendiendo a cada uno de los requerimientos de σ.

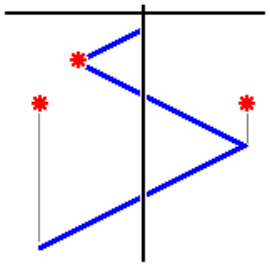
Es claro que calcular los costos óptimos de esta forma implica un esfuerzo computacional muy grande. Para ser más precisos, se trata de obtener el mínimo entre n valores, cada uno de los cuales se obtiene calculando el mínimo entre $n-1$ valores, y así sucesivamente. De modo que el esfuerzo total es $O(n!)$.

Para llevar esto a la práctica y obtener resultados en tiempos aceptables, hemos implementado un algoritmo que se basa en esta idea, pero incluye algunas optimizaciones que permiten reducir considerablemente el número de operaciones.

1. **En cada paso recursivo, removemos más requerimientos.** En la expresión 5.3.1 se invoca recursivamente a OPT sobre una secuencia de requerimientos más pequeña: $\sigma - \{r_i\}$. Veamos cómo podemos reducir esta secuencia en más de un requerimiento, de forma de acercarnos más rápidamente al caso base. Puede notarse que $\max\{t_i, \tau + d(v, x_i)\}$ indica el momento en que atendemos el requerimiento r_i, partiendo desde v en el momento τ. Esto habla de un movimiento con el que, además de servir r_i, el servidor puede atender otros requerimientos. Entonces, es razonable invocar recursivamente a OPT con la secuencia resultante de remover de σ todos los requerimientos que pueden servirse con ese movimiento.



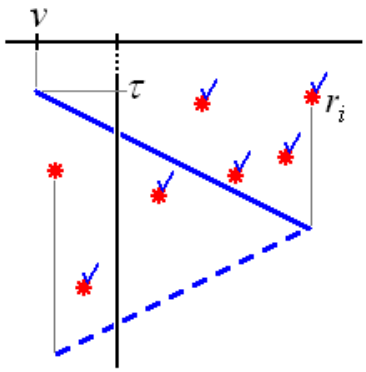
2. **Reducimos la complejidad de cada iteración.** Podemos ver a un recorrido sobre cadenas como una sucesión de movimientos hacia la izquierda y hacia la derecha. En el recorrido óptimo offline, es claro que ningún movimiento hacia la izquierda puede ir más allá de lo que fueron los anteriores. Si así fuera, tales movimientos anteriores hacia la izquierda no tendrían sentido. Lo mismo ocurre con los movimientos hacia la derecha. Es decir, cada tramo de un recorrido óptimo finaliza en el vértice más a la derecha o en el vértice más a la izquierda donde queden requerimientos por servir.



Recorrido no óptimo. Visitando primero cualquiera de los extremos, se obtiene un recorrido con menor costo.

Si hacemos uso de esta propiedad de los recorridos óptimos, podemos simplificar el caso general de nuestra recursión. En lugar de calcular el mínimo entre todos los requerimientos de σ, podemos considerar solamente los dos requerimientos extremos.

3. **Removemos aún más requerimientos.** Ahora que sabemos que en cada paso el servidor escoge entre dirigirse al extremo derecho o al extremo izquierdo, podemos asegurar que en algún momento futuro deberá dirigirse hacia el otro extremo. Es decir que, así como damos por servidos a aquellos requerimientos que el servidor atiende mientras se dirige a un extremo, podemos dar por servidos también a los requerimientos que atenderá el servidor en el trayecto hacia el otro extremo.



Con estas optimizaciones, logramos un algoritmo que, si bien tiene una complejidad exponencial de peor caso, en la práctica produce resultados en tiempos muy aceptables. Por ejemplo, en las

pruebas que hicimos sobre secuencias de hasta 2000 requerimientos, el tiempo de respuesta no superó nunca los 20 segundos en una PC de uso hogareño.

Es cierto que, con la ayuda de ciertas técnicas algorítmicas como *programación dinámica*, sería posible pensar en otros algoritmos más eficientes para obtener recorridos óptimos sobre cadenas. De hecho, en [PSM90] los autores presentan un algoritmo de complejidad cuadrática que encuentra la solución óptima para un problema similar a Nomadic DOLTSP en cadenas. Sin embargo, debido a los buenos resultados que hemos obtenido con nuestro algoritmo, consideramos que un mayor esfuerzo en este sentido escapa al alcance de nuestro trabajo. Dejaremos esta mejora para investigaciones futuras.

5.4 Ambiente de simulación

Para llevar a cabo las pruebas, hemos desarrollado un ambiente o *framework* de simulación. Este framework nos permitió implementar de un modo simple las distintas estrategias y procesar todas las instancias necesarias para analizar su comportamiento.

Los ejes que guiaron este desarrollo fueron:

- *versatilidad*: que sea posible procesar instancias muy variadas.
- *simplicidad*: que sea sencillo de utilizar, tanto para la definición de estrategias como para la definición y el procesamiento de instancias.
- *extensibilidad*: que pueda extenderse fácilmente para ser utilizado en otros problemas de ruteo online.

5.4.1 Alcance

Este ambiente permite simular instancias de DOLTSP en sus variantes Homing y Nomadic, en cualquier cadena finita. Además, está preparado para simular instancias de la versión continua de OLTSP en rectas y semirrectas y, con pequeñas modificaciones, puede extenderse el alcance a planos, circunferencias y otros espacios métricos. También pueden considerarse otros problemas de ruteo online como OLDARP y OLTRP, incluso con más de un servidor, variando su capacidad, su velocidad, etc.

5.4.2 Modelo

Con este ambiente, el procesamiento de una instancia de DOLTSP puede verse como un juego en el que participan simultáneamente varias estrategias en escenarios paralelos, compitiendo entre sí para lograr la mejor solución (terminar lo antes posible de atender a todos los requerimientos). Las

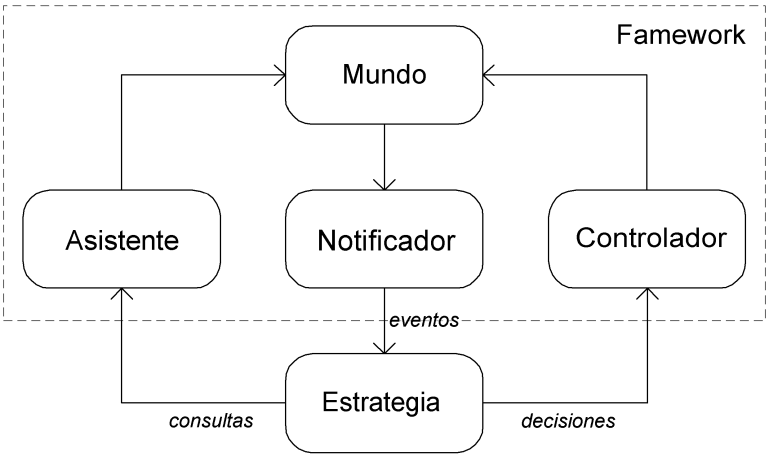
estrategias definen el comportamiento de un servidor, basándose únicamente en una serie de eventos que les son informados oportunamente. El conjunto de eventos posibles es el siguiente:

- `Game Started:` comienza el juego
- `Request Presented:` se presenta un requerimiento
- `Request Served:` el servidor atiende un requerimiento
- `Movement Started:` el servidor inicia un movimiento
- `Movement Finished:` el servidor finaliza un movimiento
- `User Reminder:` notificación solicitada por la estrategia
- `Game Finished:` no queda más trabajo por realizar

5.4.3 Diseño conceptual

Cada estrategia cuenta con algunos componentes que la comunican con el *mundo* en el que suceden los eventos y le permiten, por un lado, estar constantemente informada, y por otro, tomar decisiones sobre el recorrido a seguir por el servidor.

En el siguiente diagrama mostramos las principales relaciones que permiten interactuar a cada estrategia con el framework.



- **Mundo.** Representa a la "realidad sensible". Aquí se concentra el conocimiento sobre la instancia, el grafo, el tiempo transcurrido, la posición del servidor comandado por cada estrategia, etc. Es aquí también donde se generan los eventos que se informan a las distintas estrategias.

- **Notificador.** Es quien se encarga de notificar a las distintas estrategias de los eventos que suceden en el mundo.

- **Asistente.** Provee a cada estrategia información sobre el tiempo (¿qué hora es?), sobre la cadena (¿cuántos vértices hay? ¿cuál es el vértice más cercano que tiene el servidor a su derecha?) y sobre el servidor (velocidad, posición, movimientos realizados, etc.).

- **Controlador:** Permite a cada estrategia solicitar la ejecución de los distintos movimientos que forman su solución. Cada uno de ellos es analizado para garantizar que sea consistente con el estado del servidor correspondiente. Si el movimiento es válido, se actualiza la situación en el mundo. Las estrategias pueden generar, además, notificaciones que les serán informadas cuando lo necesiten (con eventos `User Reminder`). Una notificación de este tipo puede ser utilizada, por ejemplo, para que una estrategia conozca la finalización de una espera.

5.4.4 Definición de instancias

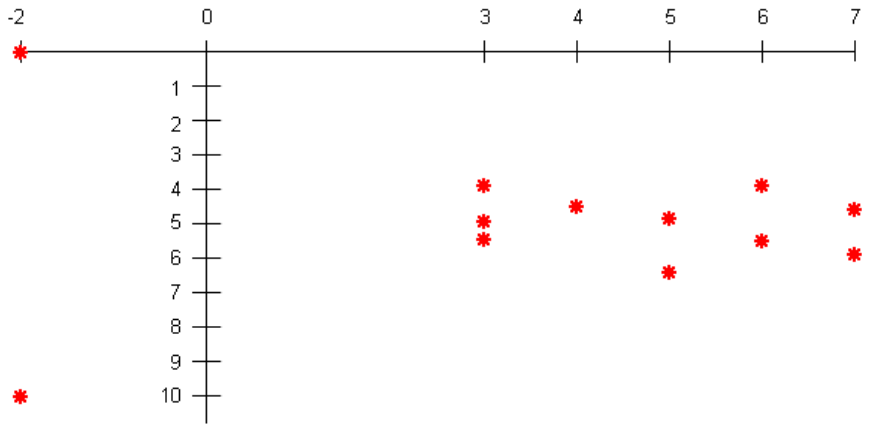
El framework permite definir múltiples instancias en formato XML, que se procesan sin interrupción, una tras otra. Si queremos definirlas a partir de meta-instancias, contamos con generadores automáticos que pueden utilizarse tanto para la definición de los vértices como para la definición de los requerimientos. Para los vértices, nos permiten definir su distribución espacial y su cantidad. Y para los requerimientos, además, podemos definir su distribución temporal.

Cada vez que se procesa una *meta-instancia* en el simulador, lo primero que se hace es convertirla en una instancia concreta, generando los vértices y los requerimientos de acuerdo a las cantidades y distribuciones especificadas. De esta forma, una misma *meta-instancia* puede ser procesada varias veces y producir diferentes resultados.

A continuación presentamos un ejemplo de cómo se definen mediante XML los vértices y los requerimientos de una *meta-instancia*:

```
<vertices>
    <vertex           position="-2" />
    <vertexGenerator  positionDistributionClass="Equidistante"
                      positionDistributionParms="3,7,1"
                      quantity="5" />
</vertices>
<requests>
    <request          time="0" position="-2" />
    <request          time="10" position="-2" />
    <requestGenerator positionDistributionClass="UniformDistribution"
                      positionDistributionParms="3,7"
                      timeDistributionClass="NormalDistribution"
                      timeDistributionParms="5,3"
                      quantity="10"   behaviour="adjustToVertices" />
</requests>
```

Como podemos apreciar, el framework permite mezclar vértices y requerimientos definidos de manera concreta con otros definidos usando generadores automáticos. También es posible usar más de un generador para definir vértices o requerimientos en una misma meta-instancia. El siguiente gráfico muestra una instancia que surge a partir de la meta-instancia del ejemplo.



5.5 Proceso de simulación

El principal objetivo del ambiente de simulación es permitirnos obtener resultados concretos que sean útiles para analizar distintas estrategias mediante un conjunto variado de meta-instancias. Pero el proceso de simulación no termina allí. A continuación describimos las distintas tareas que debimos realizar para llegar a comprender mejor el funcionamiento de las estrategias en los distintos escenarios que planteamos.

1. A partir de los valores que elegimos para cada atributo, generamos los archivos XML para todas las meta-instancias. En total, son 7506 meta-instancias que usamos para cada una de las dos variantes de DOLTSP.

2. Utilizando los generadores automáticos obtuvimos 10 instancias concretas a partir de cada meta-instancia.

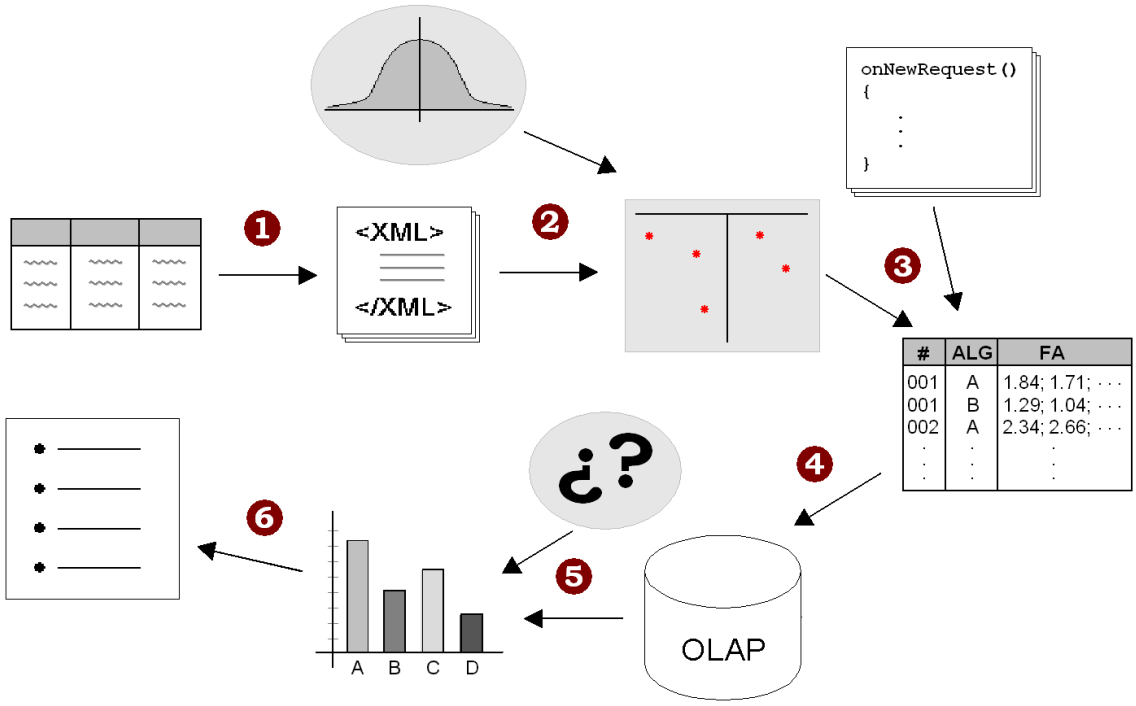
3. Procesamos todas las instancias para cada variante de DOLTSP, haciendo participar a todas las estrategias que correspondían en cada caso. Además de obtener el costo de las soluciones propuestas por cada estrategia, calculamos el costo óptimo de cada instancia.

4. Calculamos el FA de la solución generada por cada estrategia para cada instancia y almacenamos estos resultados en un cubo OLAP.

5. Formulamos un listado de preguntas para guiar nuestro análisis. Para responder cada pregunta, generamos distintos gráficos que condensan la información obtenida.

6. A partir de la observación de los resultados, llegamos a algunas conclusiones que resumen nuestro conocimiento sobre el comportamiento de las estrategias.

El siguiente gráfico resume el proceso de simulación descripto. En el próximo capítulo presentamos y analizamos la información recopilada.



Respecto al tercer paso, recordemos que no todas las estrategias fueron definidas para cualquier variante del problema, ni para cualquier grafo. La siguiente tabla detalla cuáles son los escenarios en los que participa cada una de las estrategias.

ID	estrategia	variante		grafo	
		Homing	Nomadic	semicadena	cadena
REP	Replan	X	X	X	X
STR	Statistic Replan		X	X	X
ZZG	Zig-Zag	X	X	X	X
STW	Statistic WBB	X	X	X	X
WBB	Wait Before Begin	X	X	X	X
WBRF	Wait Before Return (*fair*)	X		X	
WBRS	Wait Before Return (*std*)	X		X	

Capítulo 6

Análisis de resultados

En este capítulo nos proponemos analizar los resultados que obtuvimos con la simulación que hemos detallado en el capítulo anterior. En todos los casos, estudiaremos el comportamiento de las estrategias en la variante Homing independientemente de la variante Nomadic. También distinguiremos siempre si el grafo es una cadena simétrica, una cadena asimétrica o una semicadena.

Es importante remarcar que este análisis fue realizado exclusivamente sobre cadenas, a partir de las instancias de DOLTSP que describimos en el Capítulo 5. En otros conjuntos de instancias, nuestras conclusiones no son necesariamente válidas. Además, las estrategias fueron diseñadas sin conocer ninguna distribución a priori de los vértices ni de los requerimientos. Si acotamos el conjunto de instancias a aquellas donde estas distribuciones son conocidas, probablemente puedan diseñarse estrategias que aprovechen este conocimiento y logren mejores resultados.

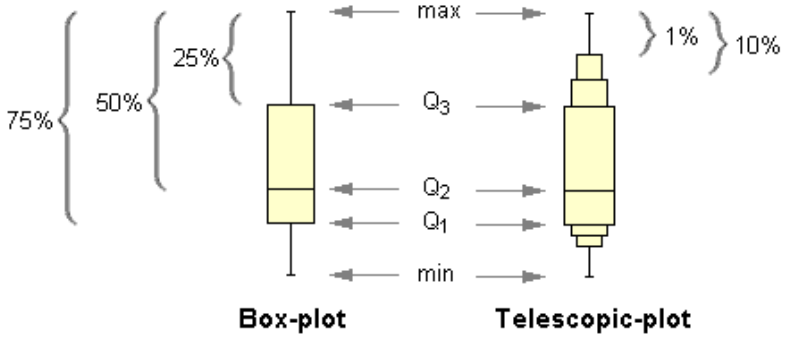
Hemos dividido el análisis en cuatro secciones. La primera está destinada a comparar la calidad de las soluciones generadas por las distintas estrategias. Sigue una descripción de las características principales de las instancias que produjeron los peores resultados para cada estrategia. En la tercera sección determinamos la proporción de instancias en las que cada estrategia cautelosa logró mejores resultados que las estrategias entusiastas. Por último, analizamos cómo influye cada uno de los atributos que detallamos en el capítulo anterior en el comportamiento de las distintas estrategias. En cada una de estas secciones enumeramos las observaciones más relevantes que surgen del análisis de los resultados y dan origen a algunas conclusiones que presentaremos en el Capítulo 7.

6.1 Comportamiento general de las estrategias

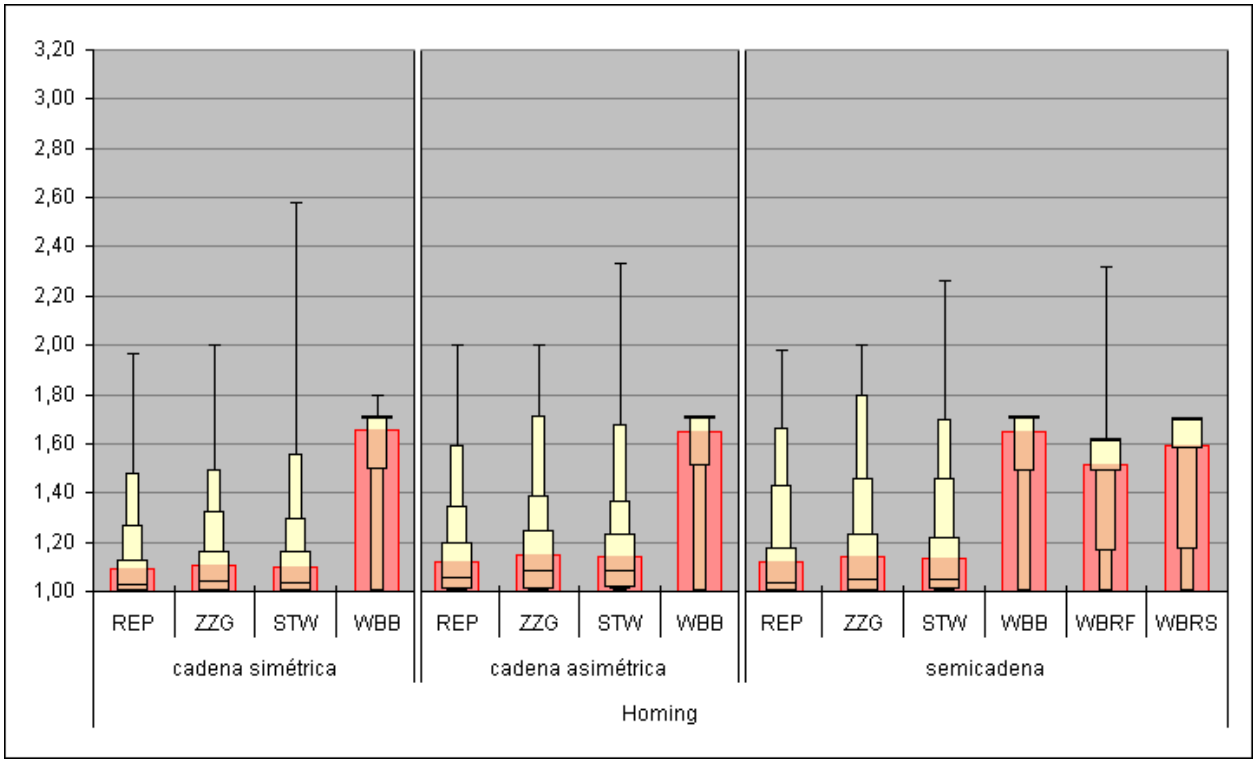
En esta sección analizamos qué estrategias produjeron soluciones de mejor calidad. Como dijimos en la Sección 5.3, medimos la calidad de una solución por su factor de aproximación (FA).

Antes de analizar el comportamiento de las estrategias vamos a describir una forma de representar la distribución de un conjunto de valores, conocida como *análisis de cuartiles*. Consiste en ordenar la muestra y determinar cuatro grupos de igual tamaño. Los cuartiles, que se denotan Q_1, Q_2 y Q_3, son aquellos valores que delimitan a estos cuatro grupos. Si bien existen diversos métodos para calcularlos, podemos decir que Q_2 es la mediana de la muestra, y que Q_1 y Q_3 son las medianas de la primera y segunda mitad, respectivamente. Los valores Q_1 y Q_3 también son frecuentemente denominados *cuartil inferior* y *cuartil superior*.

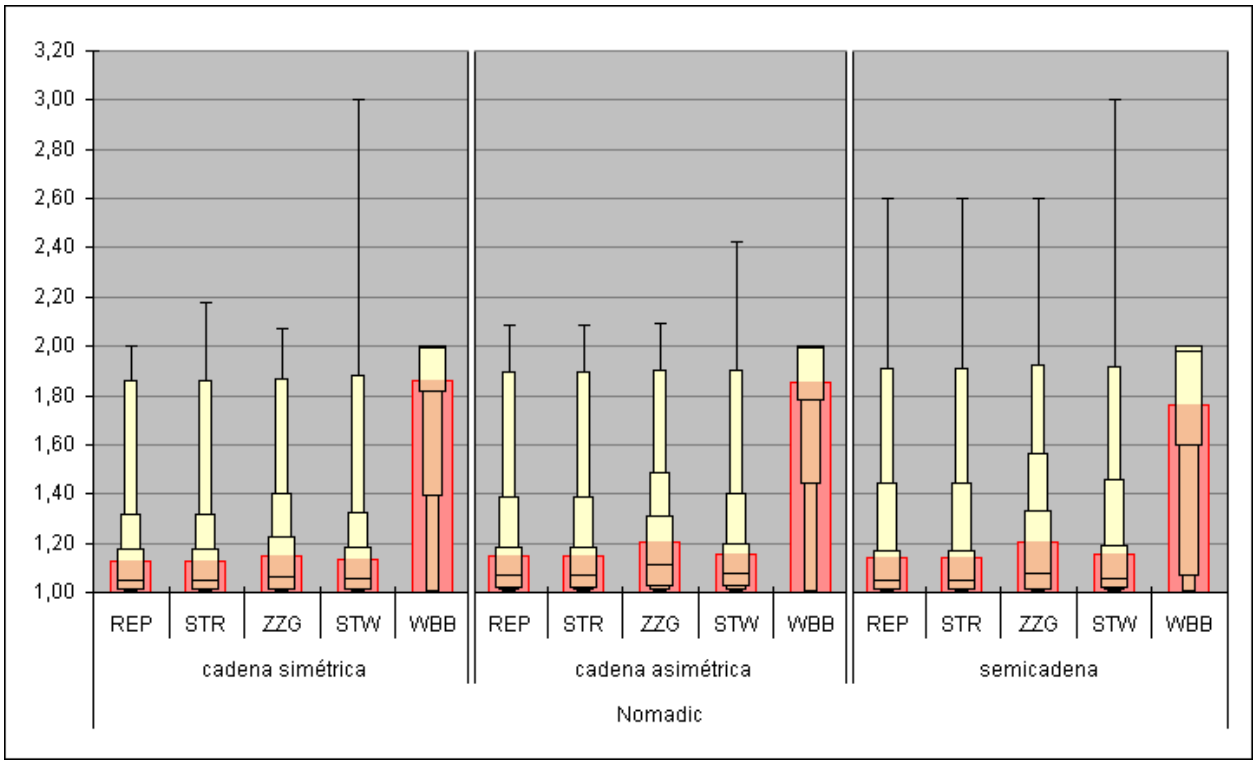
Esta información puede visualizarse utilizando un gráfico conocido como *Box and whisker diagram* o simplemente *Box-plot*. En él, además de los tres cuartiles, se representa el máximo y el mínimo. El Box-plot es una buena herramienta para representar en forma simple la distribución de un conjunto de valores, pero en ciertos casos la información es tan resumida que puede llevar a conclusiones equivocadas. Por tal motivo, hemos considerado una extensión a este diagrama que incluye además a los *percentiles* 1, 10, 90 y 99 de la muestra. Los percentiles son aquellos valores que dividen a un conjunto ordenado en cien grupos de igual tamaño. A este nuevo gráfico lo hemos denominado *Telescopic-plot*.



A continuación presentamos dos gráficos donde se visualizan con Telescopic-plots las distribuciones de todos los FA obtenidos para las variantes Homing y Nomadic, superpuestos con los respectivos gráficos de barras que indican el FA promedio de cada estrategia en cada *tipo de cadena*.



Promedio y distribución de FA para la variante Homing



Promedio y distribución de FA para la variante Nomadic

Observaciones:

1. Todas las estrategias que participan en ambas variantes tuvieron valores de FA más elevados para la variante Nomadic que para la variante Homing, en los tres tipos de cadena.

2. En ambas variantes y para todos los tipos de cadena sucedió que más de la mitad de los resultados de `REP`, `STR`, `ZZG` y `STW` estuvo por debajo de 1.10. Esto es, más de la mitad de estas soluciones tuvieron una calidad casi óptima. En cambio, las generadas por las estrategias `WBB` y `WBR` resultaron de menor calidad.

3. La cantidad de veces que una solución de `REP`, `STR`, `ZZG` o `STW` fue de menor calidad que la competitividad perseguida por `WBB` o `WBR` es inferior al 1%.

4. Si despreciamos el 1% superior, que podemos considerar *outlayers*, las estrategias `REP`, `STR`, `ZZG` y `STW` obtuvieron resultados muy similares en ambas variantes. Entrando más en detalle, `REP` obtuvo los mejores resultados, seguida por `STW` y luego por `ZZG`. La estrategia `STR` prácticamente no se diferenció de `REP`.

5. La mitad de las soluciones generadas por las estrategias que persiguen una competitividad prefijada, tuvieron un FA acorde. En la variante Homing, `WBB` fue aún más precisa: logró que el 75% de sus resultados coincidan con la competitividad esperada. Las soluciones restantes, tuvieron una mejor calidad debido a que la cota que calculan las estrategias para su adversario no siempre es ajustada.

6. La versión *fair* de `WBR` tuvo algunos FA –menos del 1%– por encima de su competitividad. Esto se debe a que el FA se calculó en base a la solución generada por un adversario *standard*.

7. Además de la versión *fair* de `WBR`, la única estrategia que generó soluciones para la variante Homing con un costo mayor que dos veces el costo óptimo fue `STW`. Si bien se trata de menos del 1% de las instancias, recordemos que esta estrategia replanifica el recorrido cada vez que el servidor llega a un vértice. `REP` hace lo mismo y es una estrategia 2-competitiva.

8. En la variante Nomadic también fue `STW` la que produjo los peores recorridos (con un costo hasta tres veces mayor al óptimo), aunque estos casos representan menos del 1% de la muestra.

6.2 Instancias difíciles

En esta sección caracterizamos a las instancias que generaron los peores resultados para cada estrategia (excluyendo a `WBB` y `WBR` porque la mayor parte de sus resultados tienen un FA cercano a su competitividad, es decir, el mayor FA posible para la estrategia). Con este objetivo, por cada estrategia y *tipo de cadena*, discriminamos distintos subconjuntos de la muestra y evaluamos el nivel de dificultad de cada uno en función de dos conceptos: *soporte* e *impacto*.

El soporte está determinado por el tamaño del subconjunto respecto al tamaño de la muestra completa. El impacto es la diferencia relativa entre el FA medio de las instancias del subconjunto y el FA medio de la muestra en general. Es decir que cuanto mayor es el soporte, más significativo es el subconjunto y cuanto mayor es el impacto, más influyente es en la calidad de las soluciones generadas. Decimos que un subconjunto determina un grupo de *instancias difíciles* si su impacto es considerablemente alto y su soporte es razonable.

En los cuadros que se muestran a continuación detallamos algunas configuraciones de atributos que generaron instancias difíciles. En general se trata de grupos de instancias con un impacto mayor a 20% y un soporte de al menos 2%. Pero en los casos donde no los hayamos encontrado, mostramos algunos conjuntos de instancias con soportes e impactos similares a los valores mencionados.

A fin de simplificar la lectura, en el atributo *distribución de vértices* agrupamos con el nombre *"lejos del origen"* a todas aquellas distribuciones en las que los vértices están acumulados fuera del origen. Con el mismo fin, en el atributo *distribución espacial de requerimientos* agrupamos con el nombre *"un extremo"* a todas aquellas distribuciones en las que los requerimientos se acumulan en un solo extremo. Además, en los casos en que el valor *sin acumulación* de la *distribución temporal de requerimientos* generó distintos resultados de acuerdo a si se utilizó una distribución uniforme, equidistante o de Poisson, este valor aparece desglosado. Por último, cuando en una configuración tomamos todos los valores de determinado atributo, lo indicamos con un guión (-) en la celda correspondiente.

estrategia	tipo de cadena	vértices		requerimientos				soporte	impacto
		cantidad	distribución	cantidad	distrib. espacial	rango temporal	distrib. temporal		
REP	cadena simétrica	-	lejos del origen	suficientes / muchos	centro	igual al espacial	sin acumulación	2,19%	31,74%
	cadena asimétrica	-	lejos del origen	-	centro	igual al espacial	sin acumulación	2,38%	29,86%
	semicadena	-	lejos del origen	-	centro	igual al espacial	sin acumulación	2,20%	44,60%
ZZG	cadena simétrica	10	lejos del origen	suficientes / muchos	centro / un extremo	igual al espacial	de Poisson / equidistante	0,97%	11,96%
	cadena asimétrica	-	lejos del origen	-	centro	igual al espacial	sin acumulación	2,38%	32,77%
	semicadena	-	-	suficientes	sin acumulación	igual al espacial	sin acumulación	2,32%	20,10%
STW	cadena simétrica	-	lejos del origen	pocos	centro / un extremo	igual al espacial	sin acumulación	2,19%	28,62%
	cadena asimétrica	-	-	pocos	centro	igual al espacial	sin acumulación	1,85%	19,59%
	semicadena	-	-	pocos	centro	igual al espacial	sin acumulación	2,20%	29,05%

Meta-instancias difíciles para la variante Homing

estrategia	tipo de cadena	vértices		requerimientos				soporte	impacto
		cantidad	distribución	cantidad	distrib. espacial	rango temporal	distrib. temporal		
REP	cadena simétrica	10	-	pocos	un extremo / sin acumulación	igual al espacial	de Poisson / uniforme / al final	2,43%	56,37%
	cadena asimétrica	10	-	pocos	centro / ext. der. / sin acumulación	igual al espacial	de Poisson / uniforme / al final	2,47%	46,79%
	semicadena	10	-	pocos	-	igual al espacial	de Poisson / uniforme	1,95%	46,18%
		10 / 30	-	pocos	extremo derecho	igual al espacial	al final	0,98%	53,27%
STR	cadena simétrica	10	-	pocos	un extremo / sin acumulación	igual al espacial	de Poisson / uniforme / al final	2,43%	56,37%
	cadena asimétrica	10	-	pocos	centro / ext. der. / sin acumulación	igual al espacial	de Poisson / uniforme / al final	2,47%	46,83%
	semicadena	10	-	pocos	-	igual al espacial	de Poisson / uniforme	1,95%	46,16%
		10 / 30	-	pocos	extremo derecho	igual al espacial	al final	0,98%	53,25%
ZZG	cadena simétrica	10	-	pocos	un extremo / sin acumulación	igual al espacial	de Poisson / uniforme / al final	2,43%	50,16%
	cadena asimétrica	10	-	pocos	centro / ext. der. / sin acumulación	igual al espacial	de Poisson / uniforme / al final	2,47%	37,36%
	semicadena	10	-	pocos	-	igual al espacial	de Poisson / uniforme	1,95%	38,32%
		10 / 30	-	pocos	extremo derecho	igual al espacial	al final	0,98%	45,04%
STW	cadena simétrica	3 / 30	lejos del origen / sin acumulación	pocos	sin acumulación	igual al espacial	sin acumulación	0,73%	53,64%
	cadena asimétrica	10	-	pocos	un ext. / sin acum	igual al espacial	de Poisson / uniforme	1,23%	45,35%
	semicadena	10	-	pocos	-	igual al espacial	de Poisson / uniforme	2,44%	45,65%

Meta-instancias difíciles para la variante Nomadic

Observaciones:

1. En general, los casos difíciles que pudimos caracterizar tuvieron más impacto en la variante Nomadic que en la variante Homing, considerando soportes similares.

2. Para la variante Homing, analizando conjuntamente el soporte y el impacto, podemos inferir que `ZZG` fue la estrategia con menos instancias difíciles caracterizables, seguida por `STW` y finalmente por `REP`. En cambio, en la variante Nomadic, estuvo en primer lugar `STW`, seguida por `ZZG` y finalmente por `REP`. Los resultados de `STR` nuevamente fueron muy similares a los de `REP`.

3. Tal como lo habíamos previsto, para ambas variantes resultó que, cuando se presentan requerimientos desproporcionadamente tardíos, las instancias difíciles no son frecuentes.

4. Para la variante Homing, las instancias más difíciles surgieron en cadenas y semicadenas con vértices lejos del origen y requerimientos presentados en el centro.

5. Para la variante Nomadic, la *distribución de vértices* no fue determinante. Los peores casos estuvieron relacionados generalmente con pocos vértices y pocos requerimientos distribuidos de formas variadas, tanto en el tiempo como en el espacio.

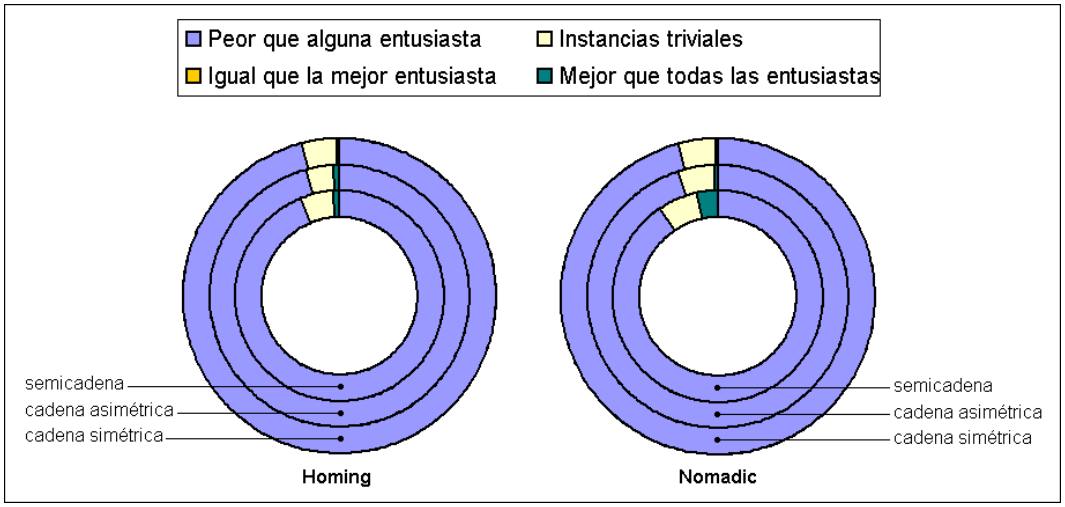
6.3 Cautela vs. entusiasmo

Hasta ahora vimos que, en la mayor parte de las instancias, las estrategias entusiastas lograron mejores resultados que `WBB` y `WBR`. Entonces cabe preguntarnos: ¿vale la pena esperar? Hemos mostrado en el Capítulo 4 que esperar es la única forma de lograr una mejor competitividad (eficiencia en peor caso). Pero ¿cuánto sentido tiene esta técnica en la práctica? ¿cuál es la mejor forma de esperar?

A fin de responder estas preguntas, para cada estrategia cautelosa (`WBB`, `WBR` y `STW`), hemos clasificado a las instancias en cuatro grupos:

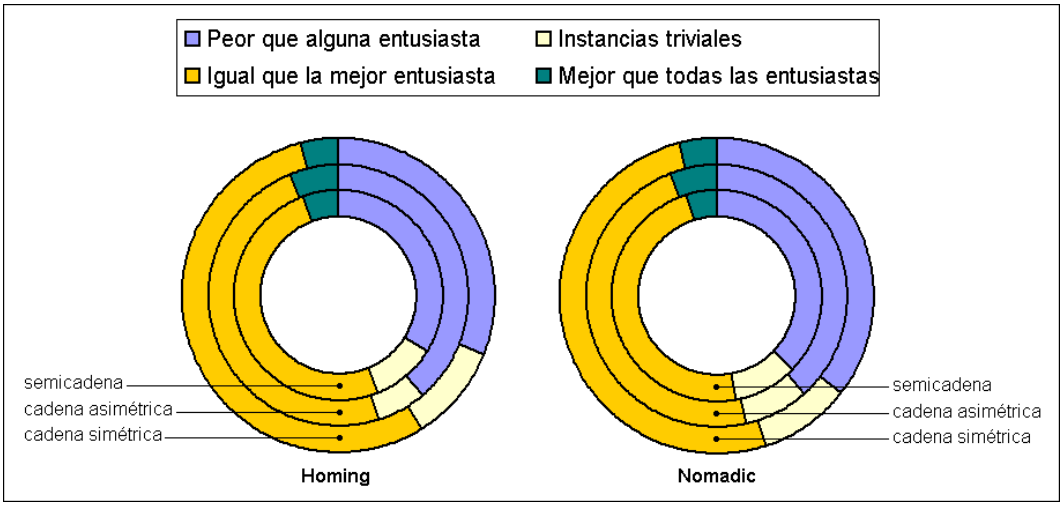
- instancias en las que la estrategia cautelosa ha generado una solución de mejor calidad que todas las entusiastas.
- instancias en las que la estrategia cautelosa ha generado una solución de igual calidad que la mejor entusiasta.
- instancias en las que la solución generada por la estrategia cautelosa ha sido superada por alguna entusiasta.
- instancias triviales, en las que todas las estrategias generaron una solución óptima.

Los gráficos que se muestran a continuación resumen la proporción de cada uno de estos grupos, según la variante y el *tipo de cadena*.



Comparación entre `WBB` y las estrategias entusiastas

Comparación entre WBR y las estrategias entusiastas



Comparación entre STW y las estrategias entusiastas

Observaciones:

1. En general, resultó mucho más frecuente que alguna estrategia entusiasta supere a las cautelosas que lo contrario.

2. En el caso de WBB y WBR –estrategias que buscan alcanzar una competitividad determinada– sus resultados fueron superados por alguna estrategia entusiasta en más del 90% de las instancias.

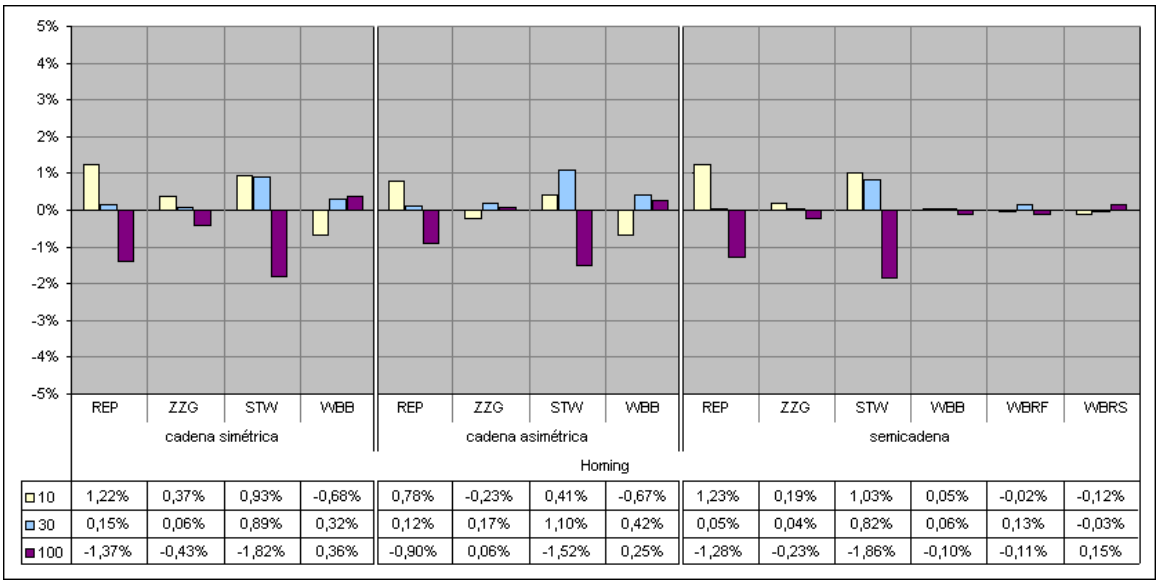
3. Distinto es el caso de STW, que se distingue de WBR y WBB en que aquí la espera no depende de una competitividad preestablecida y además el servidor replanifica su recorrido cada vez que llega a un vértice. Con esta estrategia cautelosa sólo el 30% de las soluciones fueron superadas por alguna estrategia entusiasta. Incluso STW superó a todas las estrategias entusiastas en el 5% de los casos.

6.4 Impacto de cada atributo

En esta sección analizamos cómo influye cada uno de los atributos que describimos en el Capítulo 5 en la eficiencia de las distintas estrategias. Con este fin, calculamos el impacto del mismo modo que describimos en la Sección 6.2: para cada atributo, comparamos el FA medio de las instancias asociadas a cada valor posible respecto al FA medio para el total de instancias.

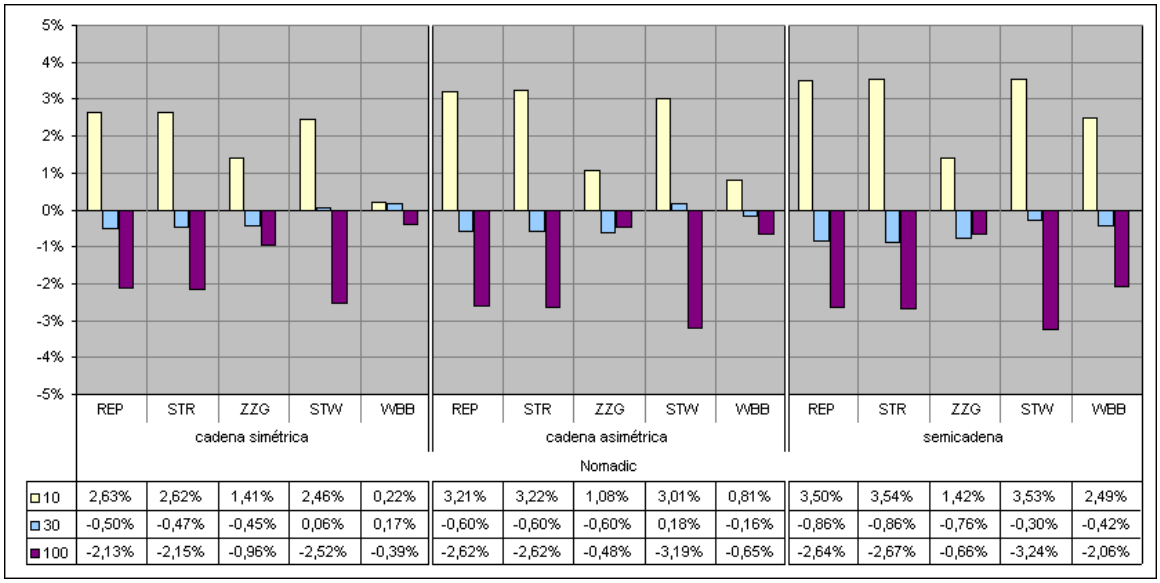
Utilizaremos como disparadores a una serie de preguntas que intentaremos responder a partir de la observación de distintos gráficos que resumen los resultados obtenidos. En todos estos gráficos, mostramos sólo los resultados correspondientes a las instancias del Grupo A (cadenas con al menos 10 vértices), ya que los resultados del Grupo B no presentaron comportamientos diferentes.

1. **¿Varía la eficiencia de las estrategias al modificar la *cantidad de vértices*?**



	REP	ZZG	STW	WBB	REP	ZZG	STW	WBB	REP	ZZG	STW	WBB	WBRF	WBRS
10	1,22%	0,37%	0,93%	-0,68%	0,78%	-0,23%	0,41%	-0,67%	1,23%	0,19%	1,03%	0,05%	-0,02%	-0,12%
30	0,15%	0,06%	0,89%	0,32%	0,12%	0,17%	1,10%	0,42%	0,05%	0,04%	0,82%	0,06%	0,13%	-0,03%
100	-1,37%	-0,43%	-1,82%	0,36%	-0,90%	0,06%	-1,52%	0,25%	-1,28%	-0,23%	-1,86%	-0,10%	-0,11%	0,15%

Impacto al modificar la *cantidad de vértices* para la variante Homing



	REP	STR	ZZG	STW	WBB	REP	STR	ZZG	STW	WBB	REP	STR	ZZG	STW	WBB
10	2,63%	2,62%	1,41%	2,46%	0,22%	3,21%	3,22%	1,08%	3,01%	0,81%	3,50%	3,54%	1,42%	3,53%	2,49%
30	-0,50%	-0,47%	-0,45%	0,06%	0,17%	-0,60%	-0,60%	-0,60%	0,18%	-0,16%	-0,86%	-0,86%	-0,76%	-0,30%	-0,42%
100	-2,13%	-2,15%	-0,96%	-2,52%	-0,39%	-2,62%	-2,62%	-0,48%	-3,19%	-0,65%	-2,64%	-2,67%	-0,66%	-3,24%	-2,06%

Impacto al modificar la *cantidad de vértices* para la variante Nomadic

Observaciones:

(a) En general, la *cantidad de vértices* tuvo un impacto mucho mayor para la variante Nomadic que para la variante Homing.

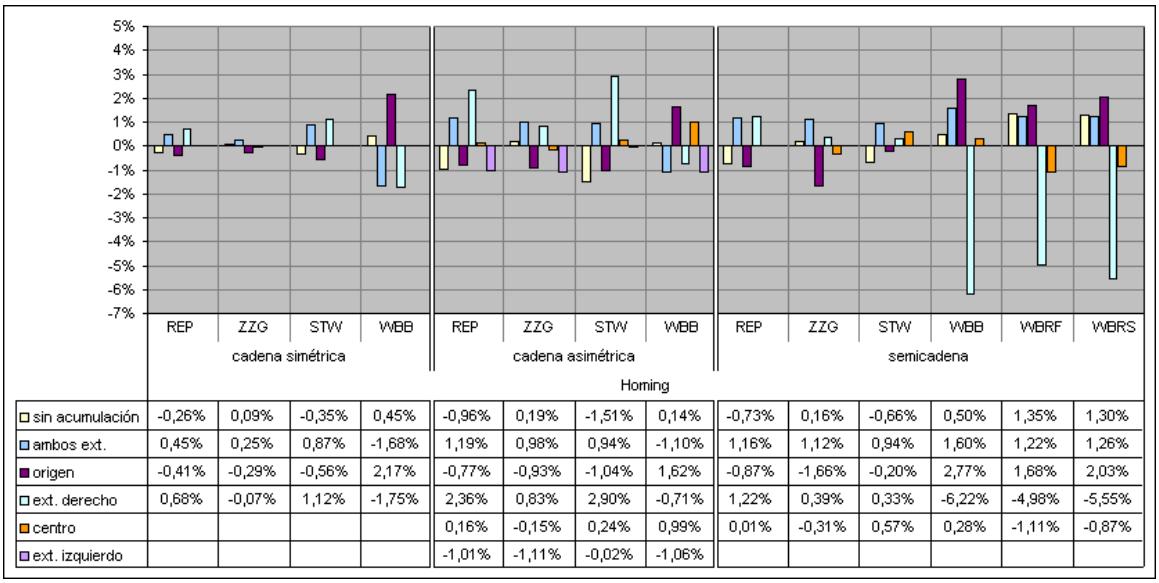
(b) Para ambas variantes, las estrategias en las que el servidor replanifica su recorrido en cada vértice (`REP`, `STR` y `STW`), mejoraron su eficiencia a medida que aumentó la cantidad de vértices, como era de esperar.

(c) En Homing, para `ZZG`, `WBB` y `WBR` la *cantidad de vértices* impactó muy poco en la eficiencia. Esto también era de esperar, puesto que estas estrategias no replanifican su recorrido en cada vértice.

(d) En Nomadic, `ZZG` mejoró su comportamiento en cadenas con más vértices, aunque no tanto como las estrategias que replanifican en cada vértice.

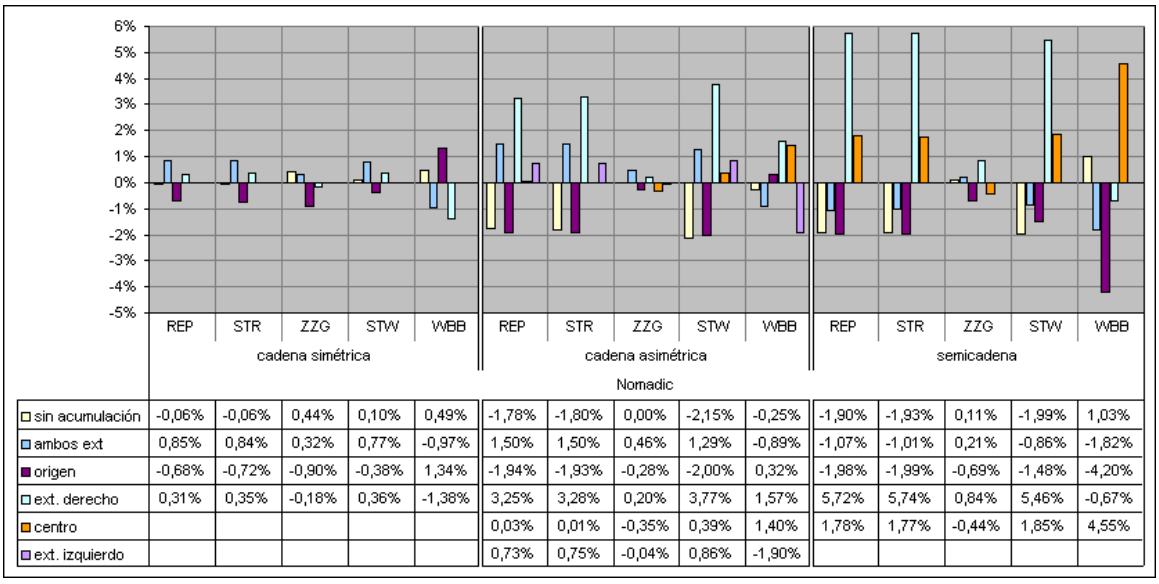
(e) Por último, en la variante Nomadic, la estrategia `WBB` tuvo un comportamiento particular, relacionado con la simetría de la cadena. Su rendimiento mejoró en todos los casos con el aumento en la cantidad de vértices, pero el impacto fue mucho mayor en semicadenas que en cadenas simétricas.

2. **¿Varía la eficiencia de las estrategias al modificar la *distribución de vértices*?**



	REP	ZZG	STW	WBB	REP	ZZG	STW	WBB	REP	ZZG	STW	WBB	WBRF	WBRS
sin acumulación	-0,26%	0,09%	-0,35%	0,45%	-0,96%	0,19%	-1,51%	0,14%	-0,73%	0,16%	-0,66%	0,50%	1,35%	1,30%
ambos ext.	0,45%	0,25%	0,87%	-1,68%	1,19%	0,98%	0,94%	-1,10%	1,16%	1,12%	0,94%	1,60%	1,22%	1,26%
origen	-0,41%	-0,29%	-0,56%	2,17%	-0,77%	-0,93%	-1,04%	1,62%	-0,87%	-1,66%	-0,20%	2,77%	1,68%	2,03%
ext. derecho	0,68%	-0,07%	1,12%	-1,75%	2,36%	0,83%	2,90%	-0,71%	1,22%	0,39%	0,33%	-6,22%	-4,98%	-5,55%
centro					0,16%	-0,15%	0,24%	0,99%	0,01%	-0,31%	0,57%	0,28%	-1,11%	-0,87%
ext. izquierdo					-1,01%	-1,11%	-0,02%	-1,06%						

Impacto al modificar la *distribución de vértices* para la variante Homing



	REP	STR	ZZG	STW	WBB	REP	STR	ZZG	STW	WBB	REP	STR	ZZG	STW	WBB
sin acumulación	-0,06%	-0,06%	0,44%	0,10%	0,49%	-1,78%	-1,80%	0,00%	-2,15%	-0,25%	-1,90%	-1,93%	0,11%	-1,99%	1,03%
ambos ext	0,85%	0,84%	0,32%	0,77%	-0,97%	1,50%	1,50%	0,46%	1,29%	-0,89%	-1,07%	-1,01%	0,21%	-0,86%	-1,82%
origen	-0,68%	-0,72%	-0,90%	-0,38%	1,34%	-1,94%	-1,93%	-0,28%	-2,00%	0,32%	-1,98%	-1,99%	-0,69%	-1,48%	-4,20%
ext. derecho	0,31%	0,35%	-0,18%	0,36%	-1,38%	3,25%	3,28%	0,20%	3,77%	1,57%	5,72%	5,74%	0,84%	5,46%	-0,67%
centro						0,03%	0,01%	-0,35%	0,39%	1,40%	1,78%	1,77%	-0,44%	1,85%	4,55%
ext. izquierdo						0,73%	0,75%	-0,04%	0,86%	-1,90%					

Impacto al modificar la *distribución de vértices* para la variante Nomadic

Observaciones:

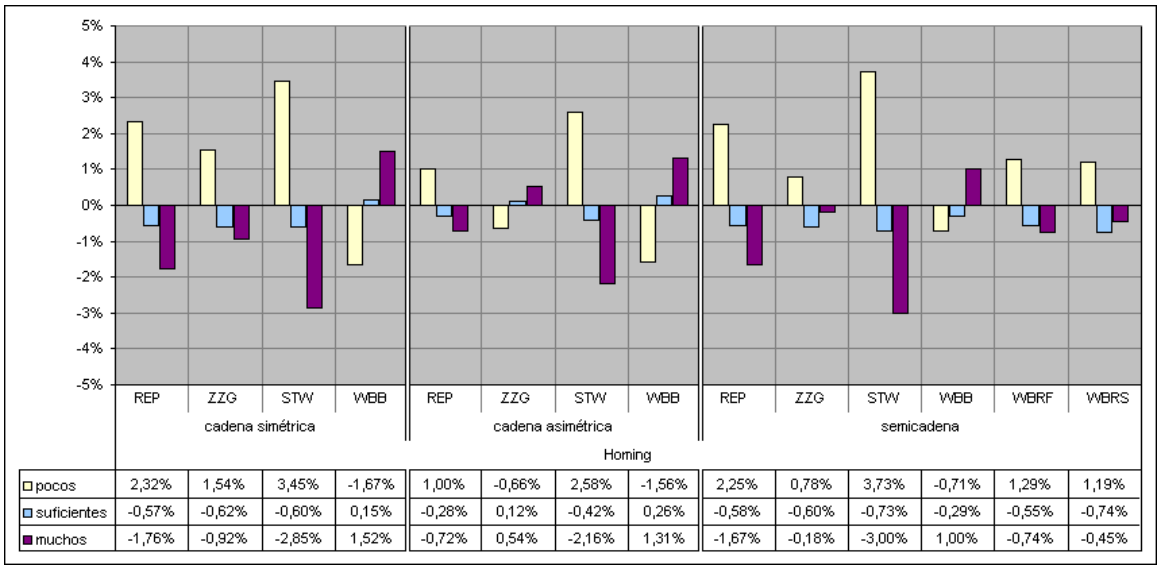
(a) En ambas variantes sucedió que, para cadenas simétricas, este atributo no tuvo mayor influencia, salvo para `WBB` que mejoró levemente su eficiencia cuando los vértices se acumularon lejos del origen y empeoró cuando estuvieron cerca.

(b) Para `ZZG`, la distribución de los vértices no influyó demasiado.

(c) `REP`, `STR` y `STW` se perjudicaron cuando los vértices se acumularon lejos del origen, principalmente en la variante Nomadic.

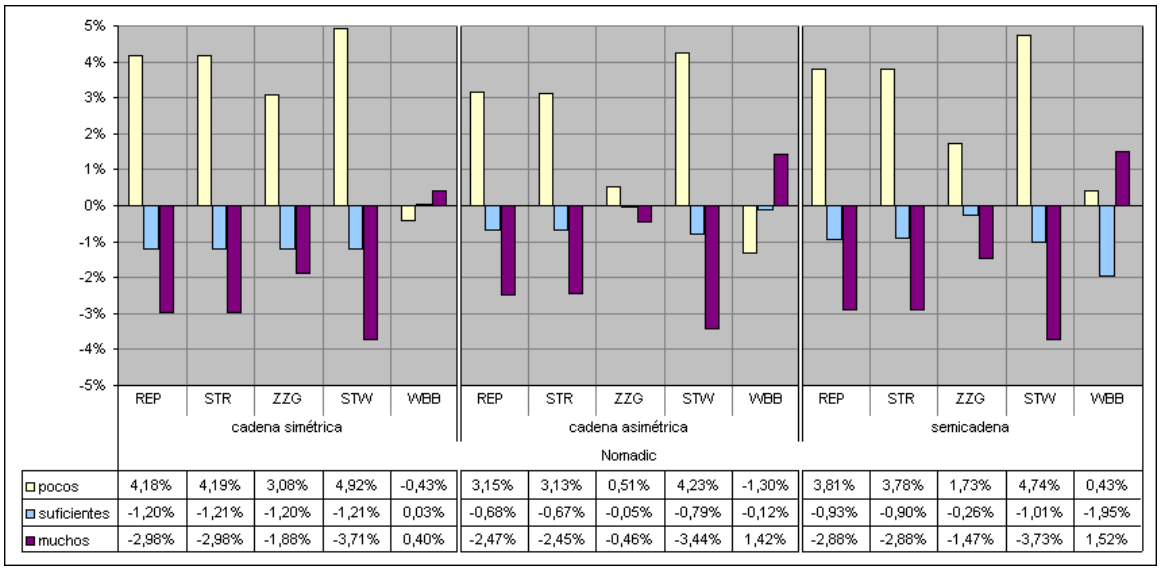
(d) En semicadenas, para la variante Homing, `WBB` y `WBR` experimentaron claras mejoras cuando los vértices se acumularon lejos del origen. Pero en la variante Nomadic el efecto fue inverso: los mejores resultados de `WBB` se observaron en semicadenas con muchos vértices cerca del origen.

3. **¿Varía la eficiencia de las estrategias al modificar la *cantidad de requerimientos*?**



	REP	ZZG	STW	WBB	REP	ZZG	STW	WBB	REP	ZZG	STW	WBB	WBRF	WBRS
pocos	2,32%	1,54%	3,45%	-1,67%	1,00%	-0,66%	2,58%	-1,56%	2,25%	0,78%	3,73%	-0,71%	1,29%	1,19%
suficientes	-0,57%	-0,62%	-0,60%	0,15%	-0,28%	0,12%	-0,42%	0,26%	-0,58%	-0,60%	-0,73%	-0,29%	-0,55%	-0,74%
muchos	-1,76%	-0,92%	-2,85%	1,52%	-0,72%	0,54%	-2,16%	1,31%	-1,67%	-0,18%	-3,00%	1,00%	-0,74%	-0,45%

Impacto al modificar la *cantidad de requerimientos* para la variante Homing



	REP	STR	ZZG	STW	WBB	REP	STR	ZZG	STW	WBB	REP	STR	ZZG	STW	WBB
pocos	4,18%	4,19%	3,08%	4,92%	-0,43%	3,15%	3,13%	0,51%	4,23%	-1,30%	3,81%	3,78%	1,73%	4,74%	0,43%
suficientes	-1,20%	-1,21%	-1,20%	-1,21%	0,03%	-0,68%	-0,67%	-0,05%	-0,79%	-0,12%	-0,93%	-0,90%	-0,26%	-1,01%	-1,95%
muchos	-2,98%	-2,98%	-1,88%	-3,71%	0,40%	-2,47%	-2,45%	-0,46%	-3,44%	1,42%	-2,88%	-2,88%	-1,47%	-3,73%	1,52%

Impacto al modificar la *cantidad de requerimientos* para la variante Nomadic

Observaciones:

(a) Variar la *cantidad de requerimientos* influyó más que variar la *cantidad de vértices* y, al igual que antes, el impacto fue mayor en la variante Nomadic.

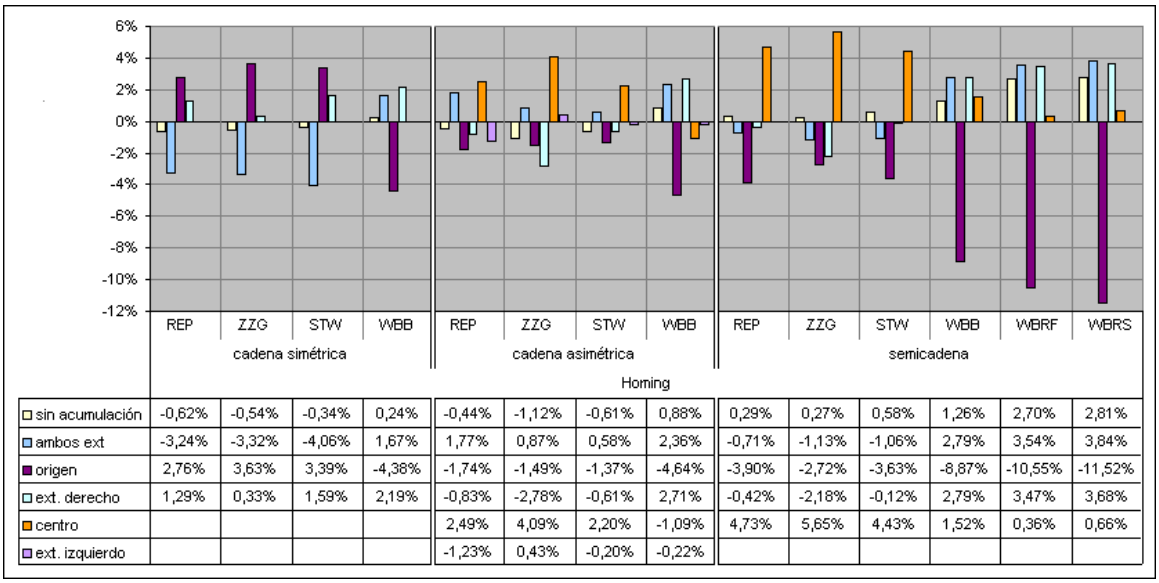
(b) Para ambas variantes, las estrategias en las que el servidor replanifica su recorrido en cada vértice (`REP`, `STR` y `STW`), mejoraron mucho su eficiencia con el aumento de la cantidad de requerimientos.

(c) También `WBR` tuvo una mejora en su comportamiento, aunque en menor magnitud.

(d) Por el contrario, para `WBB` el aumento de la cantidad de requerimientos produjo peores resultados.

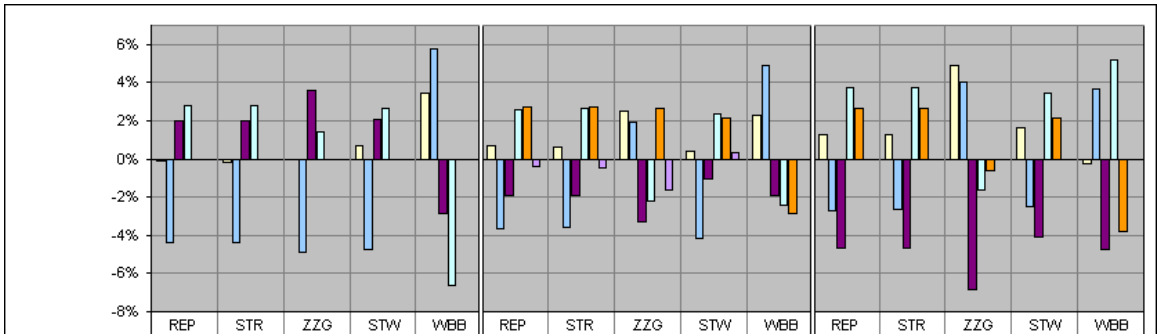
(e) Finalmente, `ZZG` mostró una leve mejora al aumentar la cantidad de requerimientos en cadenas simétricas y semicadenas (en ambas variantes), pero no se vio impactado significativamente en cadenas asimétricas.

4. **¿Varía la eficiencia de las estrategias al modificar la *distribución espacial de requerimientos*?**



	cadena simétrica				cadena asimétrica				semicadena					
	REP	ZZG	STW	WBB	REP	ZZG	STW	WBB	REP	ZZG	STW	WBB	WBRF	WBRS
sin acumulación	-0,62%	-0,54%	-0,34%	0,24%	-0,44%	-1,12%	-0,61%	0,88%	0,29%	0,27%	0,58%	1,26%	2,70%	2,81%
ambos ext	-3,24%	-3,32%	-4,06%	1,67%	1,77%	0,87%	0,58%	2,36%	-0,71%	-1,13%	-1,06%	2,79%	3,54%	3,84%
origen	2,76%	3,63%	3,39%	-4,38%	-1,74%	-1,49%	-1,37%	-4,64%	-3,90%	-2,72%	-3,63%	-8,87%	-10,55%	-11,52%
ext. derecho	1,29%	0,33%	1,59%	2,19%	-0,83%	-2,78%	-0,61%	2,71%	-0,42%	-2,18%	-0,12%	2,79%	3,47%	3,68%
centro					2,49%	4,09%	2,20%	-1,09%	4,73%	5,65%	4,43%	1,52%	0,36%	0,66%
ext. izquierdo					-1,23%	0,43%	-0,20%	-0,22%						

Impacto al modificar la *distribución espacial de requerimientos* para la variante Homing



	REP	STR	ZZG	STW	WBB	REP	STR	ZZG	STW	WBB	REP	STR	ZZG	STW	WBB
sin acumulación	-0,12%	-0,15%	-0,05%	0,68%	3,48%	0,66%	0,65%	2,50%	0,41%	2,27%	1,30%	1,30%	4,91%	1,65%	-0,26%
ambos ext.	-4,41%	-4,35%	-4,86%	-4,73%	5,79%	-3,66%	-3,61%	1,90%	-4,15%	4,93%	-2,71%	-2,65%	4,03%	-2,53%	3,64%
origen	2,01%	1,99%	3,62%	2,10%	-2,84%	-1,92%	-1,91%	-3,26%	-1,03%	-1,91%	-4,63%	-4,67%	-6,82%	-4,10%	-4,75%
ext. derecho	2,80%	2,80%	1,43%	2,68%	-6,65%	2,61%	2,62%	-2,18%	2,33%	-2,40%	3,73%	3,72%	-1,63%	3,44%	5,21%
centro						2,70%	2,69%	2,67%	2,13%	-2,85%	2,67%	2,66%	-0,63%	2,16%	-3,83%
ext. izquierdo						-0,39%	-0,44%	-1,63%	0,31%	-0,03%					

Impacto al modificar la *distribución espacial de requerimientos* para la variante Nomadic

Observaciones:

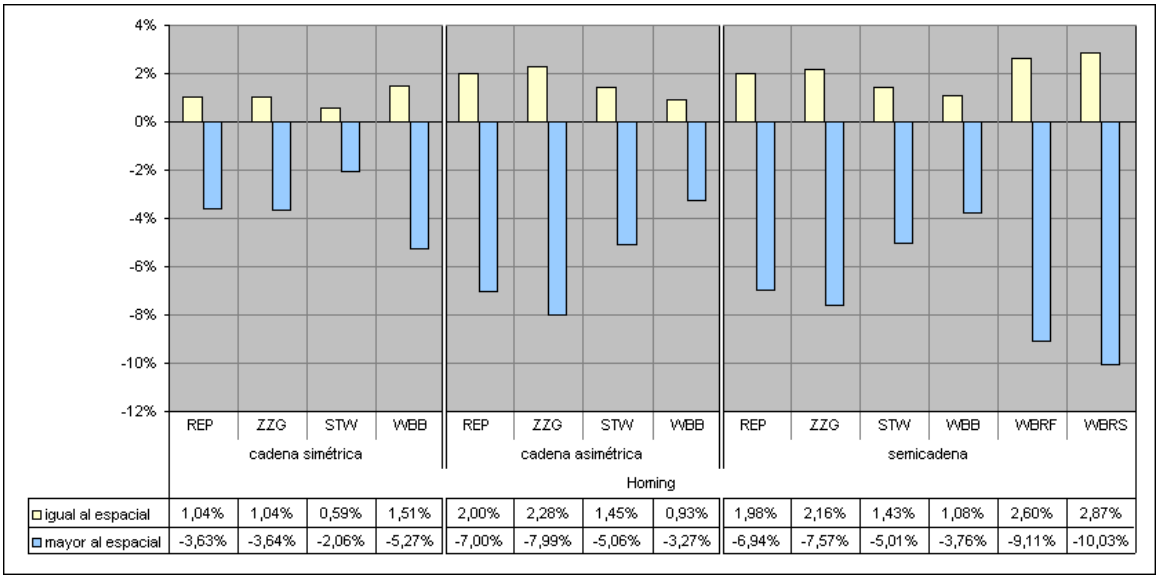
(a) En general, se observa mucho más impacto en la eficiencia de las estrategias al variar la *distribución espacial de requerimientos* que al variar la *distribución de vértices*.

(b) Los requerimientos acumulados en el origen benefician en ambas variantes a todas las estrategias, salvo en cadenas simétricas, donde la única estrategia que obtiene mejores resultados es `WBB`.

(c) Los requerimientos acumulados en el centro perjudican en ambas variantes a todas las estrategias, menos a `WBB` y a `WBR`, que en la variante Homing no se ven afectadas, y en la variante Nomadic `WBB` mejora levemente.

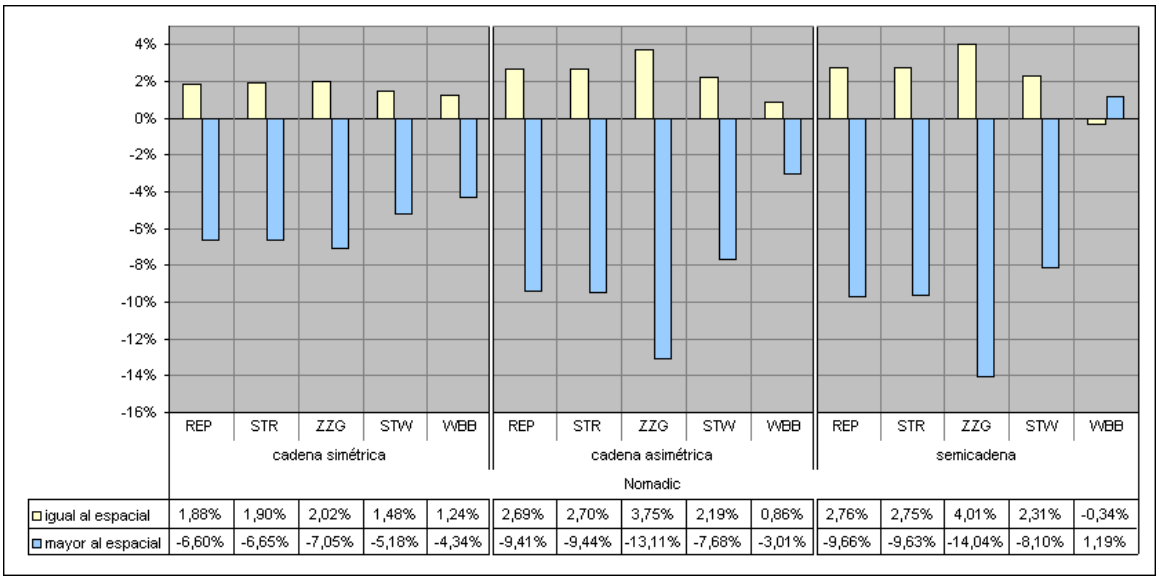
(d) Los requerimientos acumulados en ambos extremos generan buenos resultados en ambas variantes para `REP`, `STR` y `STW`. A la estrategia `ZZG`, esta distribución sólo la beneficia cuando se trata de cadenas simétricas. Finalmente, a las estrategias `WBB` y `WBR` las perjudica en todos los casos.

5. **¿Varía la eficiencia de las estrategias al modificar el *rango temporal*?**



igual al espacial	1,04%	1,04%	0,59%	1,51%	2,00%	2,28%	1,45%	0,93%	1,98%	2,16%	1,43%	1,08%	2,60%	2,87%
mayor al espacial	-3,63%	-3,64%	-2,06%	-5,27%	-7,00%	-7,99%	-5,06%	-3,27%	-6,94%	-7,57%	-5,01%	-3,76%	-9,11%	-10,03%

Impacto al modificar el *rango temporal* para la variante Homing



igual al espacial	1,88%	1,90%	2,02%	1,48%	1,24%	2,69%	2,70%	3,75%	2,19%	0,86%	2,76%	2,75%	4,01%	2,31%	-0,34%
mayor al espacial	-6,60%	-6,65%	-7,05%	-5,18%	-4,34%	-9,41%	-9,44%	-13,11%	-7,68%	-3,01%	-9,66%	-9,63%	-14,04%	-8,10%	1,19%

Impacto al modificar el *rango temporal* para la variante Nomadic

Observaciones:

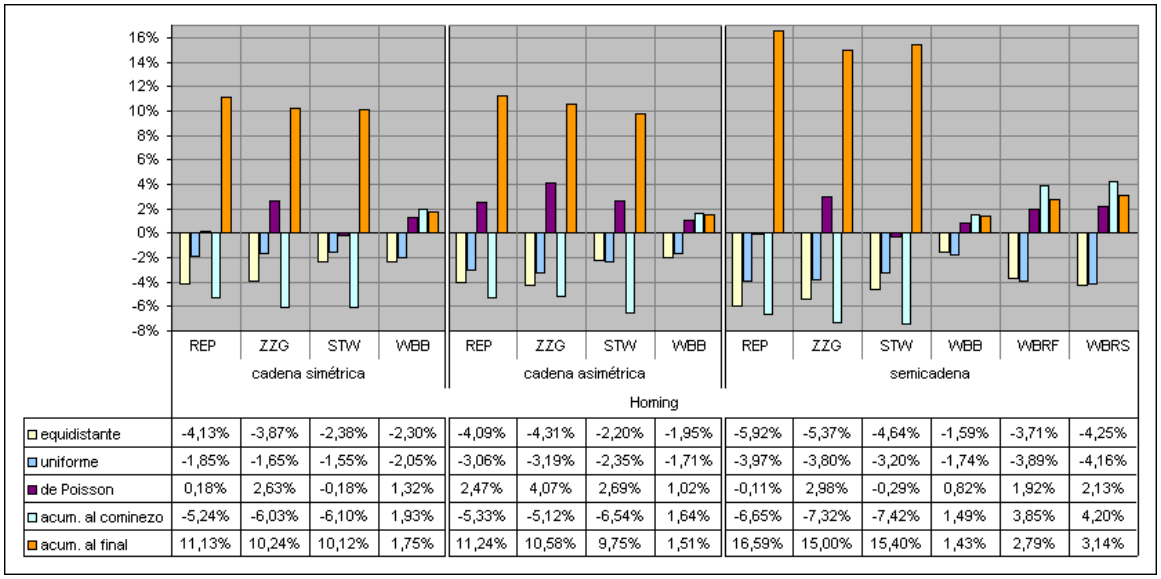
(a) Es claro que la eficiencia de las estrategias mejora notablemente cuando el *rango temporal* es 10 veces mayor que el *rango espacial*.

(b) El impacto es mayor para la variante Nomadic que para la variante Homing.

(c) Estos resultados confirman que, si medimos la eficiencia a partir del makespan, cuanto más tarde se presenta el último requerimiento, más fácil es para las estrategias obtener soluciones de buena calidad.

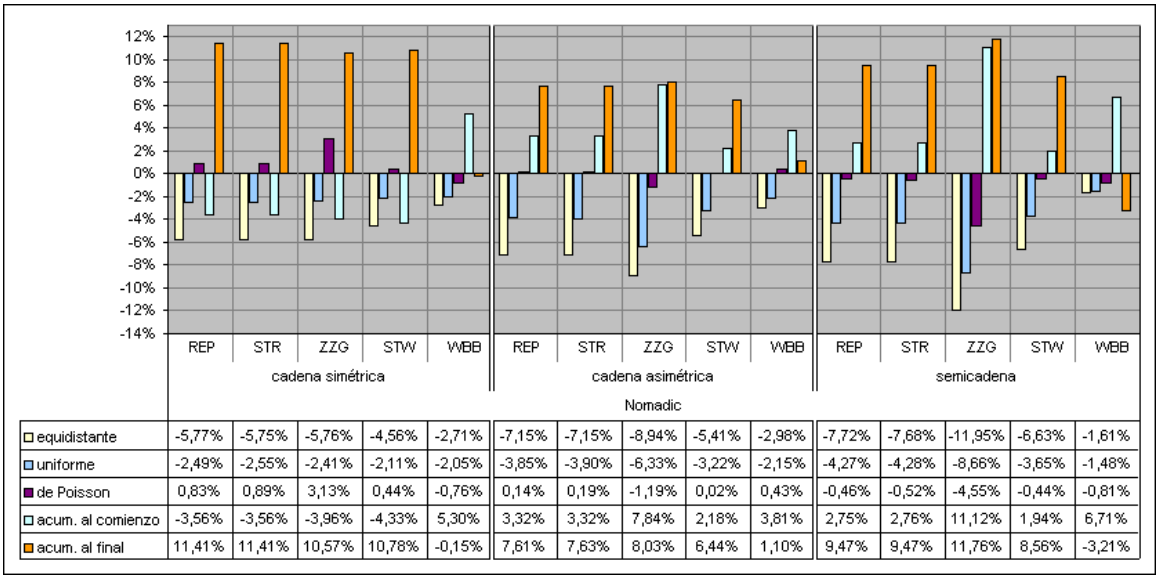
6. **¿Varía la eficiencia de las estrategias al modificar la *distribución temporal de requerimientos*?**

Aquí sólo analizamos el comportamiento de las estrategias cuando el *rango temporal* es igual al *rango espacial*, ya que en el otro caso la eficiencia de las estrategias no varía al modificar la *distribución temporal de requerimientos*. Al igual que en la Sección 6.2, como en la *distribución temporal de requerimientos* el valor *sin acumulación* generó distintos resultados de acuerdo a si se utilizó una distribución equidistante, uniforme o de Poisson, este valor aparece desglosado.



	REP	ZZG	STW	WBB	REP	ZZG	STW	WBB	REP	ZZG	STW	WBB	WBRF	WBRS
equidistante	-4,13%	-3,87%	-2,38%	-2,30%	-4,09%	-4,31%	-2,20%	-1,95%	-5,92%	-5,37%	-4,64%	-1,59%	-3,71%	-4,25%
uniforme	-1,85%	-1,65%	-1,55%	-2,05%	-3,06%	-3,19%	-2,35%	-1,71%	-3,97%	-3,80%	-3,20%	-1,74%	-3,89%	-4,16%
de Poisson	0,18%	2,63%	-0,18%	1,32%	2,47%	4,07%	2,69%	1,02%	-0,11%	2,98%	-0,29%	0,82%	1,92%	2,13%
acum. al cominezo	-5,24%	-6,03%	-6,10%	1,93%	-5,33%	-5,12%	-6,54%	1,64%	-6,65%	-7,32%	-7,42%	1,49%	3,85%	4,20%
acum. al final	11,13%	10,24%	10,12%	1,75%	11,24%	10,58%	9,75%	1,51%	16,59%	15,00%	15,40%	1,43%	2,79%	3,14%

Impacto al modificar la *distribución temporal de requerimientos* para la variante Homing



	REP	STR	ZZG	STW	WBB	REP	STR	ZZG	STW	WBB	REP	STR	ZZG	STW	WBB
equidistante	-5,77%	-5,75%	-5,76%	-4,56%	-2,71%	-7,15%	-7,15%	-8,94%	-5,41%	-2,98%	-7,72%	-7,68%	-11,95%	-6,63%	-1,61%
uniforme	-2,49%	-2,55%	-2,41%	-2,11%	-2,05%	-3,85%	-3,90%	-6,33%	-3,22%	-2,15%	-4,27%	-4,28%	-8,66%	-3,65%	-1,48%
de Poisson	0,83%	0,89%	3,13%	0,44%	-0,76%	0,14%	0,19%	-1,19%	0,02%	0,43%	-0,46%	-0,52%	-4,55%	-0,44%	-0,81%
acum. al comienzo	-3,56%	-3,56%	-3,96%	-4,33%	5,30%	3,32%	3,32%	7,84%	2,18%	3,81%	2,75%	2,76%	11,12%	1,94%	6,71%
acum. al final	11,41%	11,41%	10,57%	10,78%	-0,15%	7,61%	7,63%	8,03%	6,44%	1,10%	9,47%	9,47%	11,76%	8,56%	-3,21%

Impacto al modificar la *distribución temporal de requerimientos* para la variante Nomadic

Observaciones:

(a) Este atributo fue el que más impacto tuvo en la eficiencia de las estrategias.

(b) La distribución temporal de requerimientos que más perjudicó a `REP`, `STR`, `ZZG` y `STW` fue la acumulación de requerimientos al final del rango temporal. Pero esta distribución no significó mayor problema para `WBB` y `WBR`. Esto era esperable, puesto que esta forma de presentar los requerimientos potencia la ventaja del adversario.

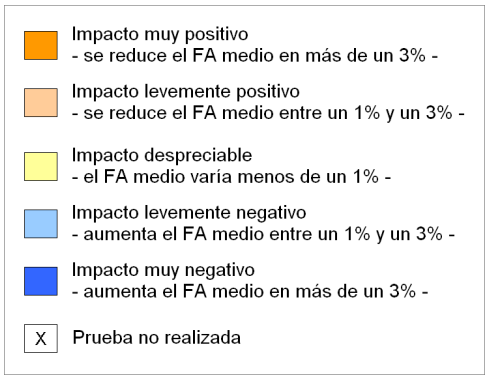
(c) En la variante Nomadic, `WBB` tuvo sus peores resultados cuando los requerimientos se presentaron al comienzo. Sucedió lo mismo con `WBR` en la variante Homing. Esto es lógico, ya que conociendo todos los requerimientos al comienzo del rango temporal, estas estrategias pueden estimar mejor el costo óptimo offline y producir así recorridos con un FA similar a su competitividad.

(d) Para `REP`, `STR`, `ZZG` y `STW`, la acumulación de requerimientos al comienzo produjo buenos resultados en la variante Homing. Pero en la variante Nomadic sólo para cadenas simétricas. Para cadenas asimétricas y semicadenas tuvo el efecto contrario, sobre todo en la estrategia `ZZG`.

(e) En general, cuando no hubo acumulación temporal de requerimientos, la distribución de Poisson fue la más difícil, seguida por la uniforme y finalmente por la equidistante. Nuevamente, estas diferencias fueron más acentuadas para la estrategia `ZZG`.

7. Resumen del impacto de los atributos

Para finalizar este análisis, presentamos dos cuadros que resumen el impacto que tuvo cada atributo en la eficiencia de las estrategias. Representamos el impacto con la siguiente escala cromática:

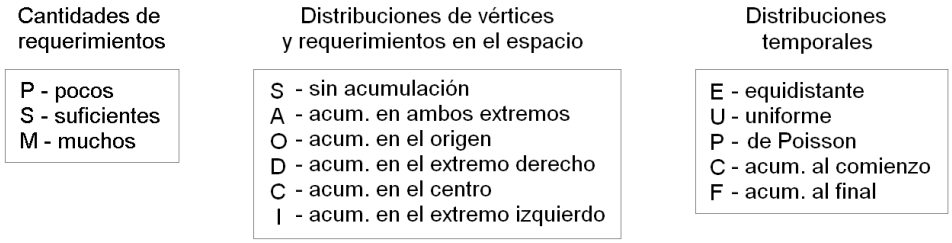


variante	estrategia	tipo de cadena	vértices										requerimientos													
			cantidad			distribución							cantidad			distrbución espacial						r.temp		distrib. temporal				
			10	30	100	S	A	O	D	C	I		P	S	M	S	A	O	D	C	I	=	>	E	U	P	C	F
Homing	REP	cadena simétrica								X	X									X	X							
		cadena asimétrica																										
		semicadena									X										X							
	ZZG	cadena simétrica								X	X									X	X							
		cadena asimétrica																										
		semicadena									X										X							
	STW	cadena simétrica								X	X									X	X							
		cadena asimétrica																										
		semicadena									X										X							
	WBB	cadena simétrica								X	X									X	X							
		cadena asimétrica																										
		semicadena									X										X							
	WBRF	semicadena									X										X							
	WBRS	semicadena									X										X							

Impacto por atributo para la variante Homing

variante	estrategia	tipo de cadena	vértices										requerimientos													
			cantidad			distribución							cantidad			distrbución espacial						r.temp		distrib. temporal				
			10	30	100	S	A	O	D	C	I		P	S	M	S	A	O	D	C	I	=	>	E	U	P	C	F
Nomadic	REP	cadena simétrica								X	X									X	X							
		cadena asimétrica																										
		semicadena									X										X							
	STR	cadena simétrica								X	X									X	X							
		cadena asimétrica																										
		semicadena									X										X							
	ZZG	cadena simétrica								X	X									X	X							
		cadena asimétrica																										
		semicadena									X										X							
	STW	cadena simétrica								X	X									X	X							
		cadena asimétrica																										
		semicadena									X										X							
	WBB	cadena simétrica								X	X									X	X							
		cadena asimétrica																										
		semicadena									X										X							

Impacto por atributo para la variante Nomadic



Observaciones:

(a) La variante Nomadic resulto ser más sensible a la variación de los atributos que la variante Homing, produciendo mayores impactos para todas las estrategias. Esto está relacionado con el grado de libertad extra que hay en la variante Nomadic, al ser desconocido el vértice donde deben terminar los recorridos.

(b) Como habíamos intuido, las estrategias que replanifican el recorrido en cada vértice (`REP`, `STR` y `STW`) se vieron beneficiadas en instancias con mayor número de vértices. No ocurrió lo mismo con las otras estrategias (`ZZG`, `WBB` y `WBR`).

(c) Al aumentar la cantidad de requerimientos también encontramos que la mayoría de las estrategias mejoran su comportamiento, con un impacto incluso mayor que al aumentar la cantidad de vértices. Esto puede deberse a que cuando hay muchos requerimientos queda poco espacio para las "sorpresas" que suelen provocar desaciertos en las estrategias online.

(d) Si medimos la eficiencia a partir del makespan, cuanto más tarde se presenta el último requerimiento, más fácil es para las estrategias obtener soluciones de buena calidad.

(e) De todos los atributos que analizamos, el que tuvo mayor impacto para las estrategias que no persiguen una competitividad (`REP`, `STR`, `ZZG` y `STW`), fue *distribución temporal de requerimientos*. En particular, los FA obtenidos cuando los requerimientos se presentaron al final del rango temporal fueron entre un 16% y un 25% más elevados que los obtenidos cuando éstos se presentaron al comienzo del rango temporal. Esto se debe a que cuanto más tarde aparece el primer requerimiento, mayor es la ventaja que tiene el adversario al poder comenzar un recorrido cuando las estrategias online aún no tienen información. En cambio, las estrategias `WBR` y `WBB` no se vieron tan impactadas por la *distribución temporal de requerimientos* porque toman sus decisiones acotando el costo óptimo offline.

Capítulo 7

Conclusiones

A lo largo de esta Tesis hemos analizado en profundidad un problema de ruteo online que denominamos DOLTSP. En este problema, un servidor se desplaza a través de un grafo para atender requerimientos que van apareciendo en distintos vértices a medida que transcurre el tiempo y debe finalizar su trabajo lo antes posible. El servidor se mueve a una velocidad constante y cuando está atravesando un eje no se le permite cambiar de dirección ni detenerse. En la variante Homing, los recorridos deben finalizar en el mismo vértice donde comenzaron. En la variante Nomadic, pueden finalizar en cualquier vértice.

Los algoritmos o estrategias online para este problema deben guiar al servidor sin conocer de antemano los momentos ni los vértices en que se presentan los requerimientos. Llamamos algoritmos *entusiastas* a aquellos que no dejan quieto al servidor si tiene trabajo por realizar, y algoritmos *cautelosos* a aquellos que utilizan la espera para aumentar las posibilidades de modificar el recorrido planificado, en caso de que aparezcan nuevos requerimientos mientras el servidor está detenido en un vértice.

Hemos presentado distintas estrategias, que analizamos en forma teórica y práctica. El análisis teórico consistió en determinar la *competitividad* de cada estrategia. Para esto comparamos el costo de las soluciones producidas por cada algoritmo online contra la de un *adversario offline* que conoce de antemano la secuencia completa de requerimientos y puede usar este conocimiento para generar una solución óptima. Consideramos dos modelos de adversario: uno *standard*, cuyo servidor puede llegar a cualquier vértice tan pronto como su velocidad se lo permita, y uno *fair*, cuyo servidor sólo puede moverse dentro del espacio delimitado por los requerimientos ya presentados. El análisis práctico consistió en definir un conjunto representativo de instancias de DOLTSP y observar la calidad de las soluciones generadas por cada estrategia, comparando los resultados en distintos aspectos.

7.1 Resumen de las estrategias presentadas

Las estrategias que presentamos fueron diseñadas para *cadenas*, es decir, grafos en los que el servidor sólo puede moverse en dos direcciones. Una de ellas funciona sólo en *semicadenas*, que son cadenas en las que el origen es un vértice con un único vecino. La siguiente tabla detalla en qué variante y grafo es válida cada una de las estrategias que definimos. Más abajo presentamos un resumen del comportamientos de cada una. Las primeras tres son entusiastas y las últimas tres son cautelosas.

ID	estrategia	variante		grafo	
		Homing	Nomadic	semicadena	cadena
ZZG	Zig-Zag	X	X	X	X
REP	Replan	X	X	X	X
STR	Statistic Replan		X	X	X
WBR	Wait Before Return	X		X	
WBB	Wait Before Begin	X	X	X	X
STW	Statistic WBB	X	X	X	X

- `Zig-Zag`: aquí el servidor realiza sucesivos tramos a izquierda y derecha, siguiendo en cada tramo un mismo sentido hasta atender a todos los requerimientos que tiene el servidor por delante. Si se trata de la variante Homing, finalmente regresa al origen.
- `Replan`: es una estrategia que ajusta la solución cada vez que se presenta un requerimiento. Con cada replanificación computa un nuevo recorrido que le permite al servidor online completar el servicio en el menor tiempo posible, recorriendo la menor distancia.
- `Statistic Replan`: mientras quedan requerimientos pendientes, el servidor se comporta del mismo modo que `Replan`. Cuando no quedan más por servir, en lugar de quedarse quieto como sugiere `Replan` en la variante Nomadic, se dirige hacia un vértice donde esperamos minimizar el costo de atender eventuales nuevos requerimientos.
- `Wait Before Return`: consiste en atender primero los requerimientos más alejados del origen, dejando para el final aquellos que pueden atenderse al regresar. Antes de emprender cada regreso, el servidor espera en su posición buscando mantener cierta competitividad.
- `Wait Before Begin`: cuando el servidor está detenido, elige un vértice destino entre todos aquellos donde hay requerimientos pendientes, pero espera un tiempo antes de comenzar a moverse hacia allí. Con esta espera busca alcanzar una competitividad preestablecida.
- `Statistic WBB`: mantiene la idea de esperar antes de ejecutar cada movimiento, pero en lugar de calcular la espera para lograr una competitividad predeterminada, el servidor comienza a moverse recién cuando el riesgo de realizar el próximo movimiento está por debajo de cierto umbral.

7.2 Principales logros

Las principales conclusiones y contribuciones que logramos con nuestro trabajo son las siguientes:

- Mostramos que al restringir en DOLTSP la posibilidad que tiene el servidor de cambiar de dirección entre dos vértices, única diferencia con OLTSP, encontramos cotas inferiores más elevadas para la competitividad. Más aún, las cotas inferiores de DOLTSP se parecen mucho a las de OLDARP, un problema más complejo que OLTSP en el que el servidor debe trasladar objetos de un punto a otro del espacio.

- Extendimos el concepto de *adversario fair*, definiéndolo para que sea válido en grafos. También adaptamos el concepto de *algoritmos entusiastas* e indrodujimos el de *algoritmos cautelosos*.

- Mostramos que la categoría de algoritmos entusiastas no garantiza ninguna competitividad, a pesar de que el servidor online se mantiene enfocado en terminar su trabajo pendiente.

- Explicitamos un esquema de espera general para estrategias que pretenden alcanzar una competitividad preestablecida y presentamos dos estrategias que lo implementan: `WBR` y `WBB`. Algunas estrategias propuestas en trabajos anteriores se ajustan a este mismo esquema, pero en forma implícita.

- Obtuvimos cotas inferiores y superiores coincidentes para DOLTSP en las variantes Homing y Nomadic, contra adversarios *fair* y *standard* y para estrategias entusiastas y cautelosas.

- Concluimos, tanto en forma teórica como empírica, que la variante Nomadic es más difícil que la Homing, porque las estrategias desconocen el vértice donde deben terminar los recorridos.

- Comprobamos que una estrategia cautelosa puede obtener una mejor competitividad que una entusiasta.

- Encontramos que, cuando se trata de algoritmos entusiastas para cadenas, el análisis de competitividad con un adversario *fair* produce los mismos resultados que con un adversario *standard*. Respecto a los algoritmos generales, comprobamos que la mejor competitividad posible en la variante Homing con un adversario *standard* es un 5% más elevada que con uno *fair*. Esta diferencia es pequeña si la comparamos contra la que se produce en la versión continua de OLTSP: en la variante Homing en semirrectas, la mejor competitividad posible con un adversario *standard* es un 17% más elevada que con uno *fair*.

- Descubrimos que, en la práctica, las estrategias que tienen la mejor competitividad (`WBR` y `WBB`) generan soluciones mucho más costosas que las estrategias con peor competitividad (`Zig-Zag` y `Replan`).

- Mostramos que la espera no sólo es útil para mejorar la competitividad, sino también para mejorar los resultados en la práctica (como el caso de `Statistic WBB`).

- Verificamos que el análisis de competitividad es poco significativo cuando se consideran rangos temporales demasiado grandes en relación al tamaño del espacio.

- Generamos evidencia empírica que resume la dificultad inherente a un problema online: la falta de información. Cuando los primeros requerimientos se presentaron en momentos tardíos, las estrategias se comportaron mucho peor que cuando todos los pedidos aparecieron al comienzo.

- Construimos una herramienta que permite simular diversos problemas de ruteo online, proveyendo una interfaz muy simple para la implementación de nuevas estrategias online. Entre sus funcionalidades más destacadas está la posibilidad de definir grandes conjuntos de instancias a partir de meta-instancias y la de visualizar gráficamente las soluciones generadas por cada estrategia.

7.3 Resumen de resultados teóricos

Los resultados teóricos que obtuvimos para las distintas variantes de DOLTSP, hasta donde hemos investigado, no figuran en trabajos anteriores. Hemos establecido cotas inferiores para algoritmos entusiastas que son válidas en cualquier grafo no trivial y para adversarios *fair* y *standard*. Los algoritmos `Zig-Zag` y `Replan` son entusiastas y alcanzan esas cotas en cadenas, tanto para la variante Homing como para la variante Nomadic de DOLTSP. En el siguiente cuadro aparece el valor de cada cota y, entre paréntesis, los teoremas en los cuales demostramos esa cota.

	cota inferior	cota superior
Homing	2 (4.2)	2 (4.4) (4.7)
Nomadic	3 (4.3)	3 (4.5) (4.8)

Cotas inferiores y superiores para algoritmos entusiastas

Para algoritmos cautelosos establecimos cotas inferiores generales que son válidas para cualquier grafo no trivial al competir contra un adversario *fair* y para cierta familia de grafos al competir contra un adversario *standard*. Las estrategias `WBR` y `WBB` aprovechan la posibilidad de esperar demorando al máximo la decisión de comenzar determinados movimientos y así logran mejores competitividades que las estrategias entusiastas.

	fair		*standard*	
	cota inferior	cota superior	cota inferior	cota superior
Homing	$\frac{1+\sqrt{5}}{2} \approx 1.618$ (3.1)	$\frac{1+\sqrt{5}}{2} \approx 1.618$ (4.19)	$\frac{2+\sqrt{2}}{2} \approx 1.707$ (3.3)	$\frac{2+\sqrt{2}}{2} \approx 1.707$ (4.20)
Nomadic	≈ 1.839 (3.4)	2 (4.29)	2 (3.6)	2 (4.29)

Cotas inferiores y superiores para algoritmos generales

Como puede observarse, en todos los casos coinciden las cotas inferiores con las respectivas superiores, salvo para la variante Nomadic contra un adversario *fair*, donde hay una pequeña brecha.

7.4 Trabajo futuro

Para finalizar, proponemos algunas extensiones que complementarían nuestra investigación.

- Como puede apreciarse en el resumen de resultados de algoritmos generales, en la variante Nomadic de DOLTSP contra un adversario *fair* no coinciden las cotas inferior y superior. Queda como pregunta abierta determinar si existe una cota inferior más elevada, o si puede definirse un algoritmo online con una mejor competitividad que `WBB`.
- Dejamos para un trabajo futuro el análisis de competitividad de `WBB` en cadenas completas, el de `WBB` para la variante Homing y el de las estrategias `Statistic WBB` y `Statistic Replan`.
- En nuestro trabajo analizamos el problema sobre cadenas y semicadenas. Una primera extensión consistiría en analizarlo sobre árboles en general. En este caso el problema resulta más complicado porque el servidor debe optar entre más de dos direcciones posibles. Un siguiente paso en este sentido podría incluir grafos circulares y finalmente grafos generales con circuitos, en los que hay más de un camino para llegar a cada vértice.
- Al igual que con OLTSP, podrían analizarse otros problemas de ruteo online restringiendo las posibilidades de los servidores para que éstos sólo puedan modificar sus recorridos en puntos específicos del espacio.
- Respecto a la herramienta de simulación que desarrollamos, sería interesante extenderla para que permita definir instancias en forma gráfica y modificarlas en tiempo real, de manera interactiva.
- Por último, quizás utilizando programación dinámica pueda reducirse la complejidad de nuestro algoritmo recursivo para calcular el costo óptimo offline.

Bibliografía

[ABF96] B. Awerbuch, Y. Bartal, and A. Fiat. Distributed paging for general networks. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 574–583, 1996.

[ACP+86] F. N. Afrati, S. S. Cosmadakis, C. H. Papadimitriou, G. Papageorgiou, and N. Papakostantinou. The complexity of the travelling repairman problem. *Informatique Théorique et Applications*, 20(1):79–87, 1986.

[AFL+01] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Algorithms for the on-line travelling salesman. *Algorithmica*, 2001.

[BCGL06] G. Berbeglia, J. F. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems: A classification scheme and survey. 2006.

[BEY98] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[BKdPS00] M. Blom, S. Krumke, W. E. de Paepe, and L. Stougie. The online tsp against fair adversaries. *Lecture Notes in Computer Science*, 1767:137–??, 2000.

[DB84] I. Deif and L. Bodin. Extension of the clarke and wright algorithm for solving the vehicle routing problem with backhauling. In *Babson Conference on Software Uses in Transportation and Logistics Management*, Babson Park, MA, 1984.

[Dev98] J. L. Devore. *Probabilidad para ingeniería y ciencias*, pages 125–126, 682–683. International Thomson Editores, fourth edition, 1998.

[dP02] W. E. de Paepe. *Complexity Results and Competitive Analysis for Vehicle Routing Problems*. PhD thesis, Technische Universiteit Eindhoven, 2002.

[Feu95] E. Feuerstein. *On-line Paging of Structured Data and Multi-threaded Paging*. PhD thesis, Dipartamento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1995.

[FW98] A. Fiat and G. J. Woeginger. chapter 14. Number 1442 in Springer LNCS "State-of-the-Art Survey". Fiat and Woeginger, 1998.

[JOPW86] J. J. Jaw, A. R. Odoni, H. N. Psaraftis, and N. H. M. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research*, 20B(3):243–257, 1986.

[KP94] E. Koutsoupias and C. Papadimitriou. Beyond competitive analysis. In *IEEE Symposium on Foundations of Computer Science*, pages 394–400, 1994.

[Kru56] J. B. Kruskal. On the shortest spanning subtree and the traveling salesman problem. In *American Mathematical Society*, volume 7, pages 48–50, 1956.

[Kru01] S. Krumke. *Online Optimization, Competitive Analysis and Beyond*. Habilitationsschrift Technische Universität Berlin, 2001.

[Lin65] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44:2245–2269, 1965.

[Lip03] M. Lipmann. *On-line Routing*. PhD thesis, Technische Universiteit Eindhoven, 2003.

[Mul90] H. M. Mulder. *The Expansion Procedure for Graphs*, pages 459–477. R. Bodendiek, 1990.

[PK95] K. Pruhs and B. Kalyanasundaram. Speed is as powerful as clairvoyance. In *IEEE Symposium on the Foundations of Computer Science*, pages 214–221, 1995.

[Psa80] H. N. Psaraftis. A dynamic programming approach to the single-vehicle, many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14:130–154, 1980.

[PSM90] H. N. Psaraftis, M. M. Solomon, and T. L. Magnanti. Routing and scheduling on a shoreline with release times. *Tai-Up Kim Management Science*, 36(2):212–223, 1990.

[PSTW97] C. Philips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *29th Annual ACM Symposium on the Theory of Computing*, pages 140–149, 1997.

[Rud76] W. Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, third edition, 1976.

[Sav85] M. W. P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4(1-4):285–305, 1985.

[ST85] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[Ste78] D. Stein. An asymptotic, probabilistic analysis of a routing problem. *Mathematics of Operations Research*, 3(2):89–101, 1978.