Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

**Unidad Zacatenco**

**Departamento de Computación**

# Tratamiento eficiente del problema del reparador viajero con ventanas de tiempo

Tesis que presenta

**Oscar Yani Miguel Pilar**

para obtener el grado de

**Maestro en Ciencias en Computación**

Directores de la Tesis

**Dr. Guillermo Benito Morales Luna**
**Dr. Feliú Davino Sagols Troncoso**

México, D.F. Noviembre de 2015

# Resumen

El problema del reparador viajero con ventanas de tiempo (TRPTW por sus siglas en inglés: *Traveling Repairman Problem with Time Windows*) es un problema de calendarización de tareas, el cual es importante estudiar debido a su vasta aplicación en áreas como manufactura, logística, transportación o turismo. Este problema surge al agregar restricciones de tiempo al problema del agente viajero.

En este trabajo se estudia el caso especial de TRPTW en el cual las ventanas de tiempo de las tareas son de longitud unitaria. Se presentan algunos problemas relacionados y se analizan experimentalmente dos algoritmos de la literatura que ofrecen soluciones aproximadas para TRPTW.

Además, se propone un enfoque de programación lineal entera para resolver TRPTW en el mismo caso (ventanas de tiempo unitarias). Se expone un análisis experimental de este enfoque de programación lineal entera. El modelo se extiende para soportar el caso en que las ventanas de tiempo tienen longitud arbitraria. Las implementaciones de esta tesis conforman una plataforma computacional que resuelve instancias de TRPTW. La plataforma implementa dos algoritmos de aproximación de la literatura así como el enfoque de programación lineal entera que hemos propuesto.

# Abstract

The Traveling Repairman Problem with Time Windows (TRPTW) is a task scheduling problem that has become important due to its wide application in areas such as manufacturing, logistics, transportation, or tourism. This problem arises when adding time restrictions to the traveling salesman problem.

In this work, we study the special case of TRPTW in which time windows have unit length. We introduce some related problems and analyze two very well known algorithms to approximate TRPTW.

A new integer linear programming approach to solve TRPTW in this special case (unit time windows) is proposed. An experimental analysis of this integer linear programming approach is exposed. Later, this approach is extended to support the case of TRPTW with time windows for arbitrary length. Our implementations establish a computational platform to solve instances of TRPTW. They include the two algorithms analyzed as well as our new integer linear approach.

# Agradecimientos

Agradezco a México, que a través del Consejo Nacional de Ciencia y Tecnología (Conacyt) sustentó económicamente mis estudios de maestría. Gracias al Cinvestav por la formación académica, infraestructura y apoyos brindados.

Agradezco al Dr. Guillermo Morales por la oportunidad de asesorar mi trabajo de tesis; por su confianza y consejos. Al Dr. Feliú Sagols por co-asesorar la tesis, por su paciencia y observaciones relevantes. Al Dr. Francisco Zaragoza por sus comentarios siempre precisos que aportaron contenido sustancial en este trabajo. A mis sinodales de tesis por su tiempo y correcciones oportunas.

A los profesores del Departamento de Computación del Cinvestav Zacatenco por sus cursos impartidos; al Dr. Debrup Chakraborty por el magistral curso de *Design and Analysis of Algorithms*. También al Dr. Guillermo Morales por el minucioso formato y contenido de sus cursos de Complejidad Computacional. Al personal administrativo del departamento de Computación, a Sofía Reza, les agradezco el apoyo en trámites académicos.

Mi más franco agradecimiento y reconocimiento a mi madre, a mi padre, a mi hermana y hermano por su apoyo incondicional en este camino misterioso y desconocido que es la vida.

Mi agradecimiento a Helena.

A todas aquellas personas que divulgan el conocimiento.

# Índice general

# Índice de figuras

# Índice de tablas

# Capítulo 1

# Introducción

## 1.1. Motivación

El problema del reparador viajero es un problema de orientación. En los últimos años, los problemas de orientación han sido útiles para modelar un vasto número de aplicaciones en manufactura, logística, transporte, entre otros [Vansteenwegen et al., 2011]. Estos problemas, desde el enfoque computacional, son una combinación del problema de la mochila (KP: *Knapsack Problem*) y el problema del agente viajero (TSP: *Travelling Salesman Problem*). Es decir, dichos problemas pueden verse como una combinación del problema de selección de vértices con pesos y el problema de encontrar la trayectoria hamiltoniana[1] más corta entre esos vértices (*Shortest Hamiltonian Path*). El objetivo en un problema de orientación es maximizar la ganancia total recolectada, mientras que en TSP se busca minimizar la distancia recorrida [Vansteenwegen et al., 2011, p. 11].

En este trabajo nos enfocamos en una variante del problema del reparador viajero con ventanas de tiempo, en esta versión las ventanas de tiempo son unitarias. Es importante estudiar tal variante ya que es un problema computacionalmente difícil [Perez Perez et al., 2014] y tiene aplicaciones prácticas directas como la calendarización de tareas, el enrutamiento de repartidores, aplicaciones militares o de turismo, por mencionar algunas [Vansteenwegen et al., 2011].

## 1.2. Planteamiento del problema

A pesar de que en la literatura existen algoritmos para resolver el problema del reparador viajero con ventanas de tiempo, existen pocas implementaciones que comprueben experimentalmente la eficiencia de estos. En el caso particular del problema en el cual las ventanas de tiempo tienen longitud unitaria, existen algoritmos de aproximación que merece la pena implementar y comprobar su eficiencia. También es

[1] En la sección 2.1.1 se define el término *trayectoria hamiltoniana*.

importante abordar el caso en que las ventanas de tiempo tienen longitud arbitraria, pues tal variante se ha estudiado en menor grado que otras variantes en la literatura.

Aunque la variante del problema que estamos abordando se ha estudiado por varios científicos mediante diversas técnicas como teoría de grafos y programación dinámica, conviene analizar y proponer otros enfoques de solución para buscar mejorar la tasa de aproximación y complejidad computacional de los algoritmos actuales.

## 1.3. Objetivos de la tesis

En este trabajo se realiza un estudio del panorama actual del problema del reparador con ventanas de tiempo. Se implementan dos algoritmos de aproximación de la literatura que resuelven el caso particular del problema donde las ventanas de tiempo son unitarias; se comparan estos programas y se comprueba experimentalmente la eficiencia de aproximación de un programa respecto al otro. También se han implementado algunos algoritmos relacionados.

Además, se propone un enfoque de programación lineal entera para esa variante del problema. Posteriormente, basado en este enfoque, se aborda el caso en que las ventanas de tiempo son arbitrarias. Nuestro enfoque es analizado experimentalmente generando instancias aleatorias del problema para medir su eficiencia.

## 1.4. Organización de la tesis

En el capítulo 1 hemos presentado la motivación y objetivos de la tesis. En el capítulo 2 se introducen, en la primera parte, los términos y problemas fundamentales para abordar el problema del reparador viajero con ventanas de tiempo. En la segunda parte, se presenta el estado del arte del problema del reparador. Cuando se emplean conceptos de teoría de grafos a lo largo de esta tesis, se hace siguiendo la nomenclatura del texto *Graph Theory* de Reinhard Diestel (ver [Diestel, 2012]).

En la primera parte del capítulo 3 se exponen e implementan algunos problemas de calendarización de tareas, en la segunda parte se revisan dos algoritmos de la literatura que resuelven el problema del reparador viajero y se comparan experimentalmente los resultados generados por estos algoritmos.

En el capítulo 4 proponemos un nuevo enfoque de solución para el problema del reparador viajero con ventanas de tiempo unitarias. Este enfoque se basa en programación lineal entera, para resolver el modelo empleamos el *software* matemático *Gurobi Optimizer*; implementamos y presentamos un análisis experimental de este enfoque generando instancias de prueba aleatorias. Posteriormente extendemos el procedimiento para soportar ventanas de tiempo de longitud arbitraria.

Finalmente, en el capítulo 5 se presentan las conclusiones generales y el posible trabajo a futuro. Además, se encuentra un vínculo al sitio electrónico donde se puede obtener el código fuente, programas y manuales de usuario de las implementaciones realizadas.

# Capítulo 2

# Nociones fundamentales

En la primera parte de este capítulo se recopilan definiciones de términos que son utilizados con frecuencia en esta tesis para facilitar al lector la lectura de ésta. También se define matemáticamente a los problemas de orientación acorde a [Vansteenwegen et al., 2011]. Se define también el problema del reparador viajero con ventanas de tiempo. En la segunda parte se hace una revisión del estado del arte del problema del reparador viajero así como las diferentes variantes o casos especiales en que nos centramos en este trabajo.

## 2.1. Conceptos previos

Las siguientes subsecciones son una recopilación de definiciones utilizados con frecuencia en esta tesis. En la subsección 2.1.7 se definen los problemas de orientación y se presenta su definición matemática. En la subsección 2.1.8 se define el problema del reparador viajero con ventanas de tiempo. En la subsección 2.1.8 se describen las técnicas de solución empleadas para tratar este problema.

### 2.1.1. Trayectoria hamiltoniana

Sea $G = (V, E)$ un grafo con vértices $V = \{v_1, \ldots, v_n\}$ con $n = |V|$ y aristas $E$, una *trayectoria hamiltoniana* es una permutación de los vértices $< v_1, v_2, \ldots, v_n >$ de $G$ tal que $\{v_i, v_{i+1}\} \in E$ para todo $i$ tal que $1 \leq i < n$. Una *trayectoria hamiltoniana entre dos puntos* es una trayectoria hamiltoniana que comienza en un vértice $u$ y termina en un vértice $v$. [Garey and Johnson, 1990, p. 60].

### 2.1.2. Trayectoria monótonamente creciente

En este documento empleamos el término *trayectoria monótonamente creciente* para referirnos a una trayectoria formada por un conjunto de segmentos, los cuales siempre conservan el orden de ascendencia en sus coordenadas cartesianas, es decir, para todo segmento $i$, sus coordenadas cartesianas del punto final $q_i = (x_i, y_i)$

siempre son mayores que las coordenadas del punto inicial $p_1 = (x_i, y_i)$. Además, las coordenadas de un segmento $i$ siempre son mayores que las del segmento $i_{i-1}$.

### 2.1.3. Trayectoria $y$-monótonamente creciente

Cuando en este documento empleamos el término *trayectoria y-monótonamente creciente* nos referimos a una trayectoria, sobre el plano bidimensional, que inicia en el orígen y está compuesta por una secuencia de líneas que tienen inclinación en el intervalo $[45°, 135°]$. La trayectoria es monótonamente creciente respecto a las coordenadas $y$. La trayectoria termina en un punto predeterminado sobre el eje vertical [Bar-Yehuda et al., 2005].

### 2.1.4. Trayectoria $x(y)$-monótonamente creciente

Una trayectoria $x(y)$-monótonamente creciente, sobre el plano bidimensional, está compuesta por una secuencia de líneas que tienen inclinación en el intervalo $[0°, 90°]$. La trayectoria es monótonamente creciente respecto a las coordenadas $x$ y $y$. La trayectoria inicia en el orígen y termina en un punto predeterminado [Bar-Yehuda et al., 2005].

### 2.1.5. Otros términos empleados en este trabajo

#### Espacio métrico

Un espacio métrico es una pareja $(M, d)$ donde $M$ es un conjunto de puntos y $d$ es una función de distancia denominada métrica tal que $d : M \times M \to \mathbb{R}$. La función de distancia debe cumplir las siguientes propiedades $\forall x, y, z \in M$ [Papadopoulos, 2005]:

$$d(x, y) \geq 0 \qquad \textit{positividad} \tag{2.1}$$

$$d(x, y) = 0 \leftrightarrow x = y \qquad \textit{distancia 0 entre un mismo punto} \tag{2.2}$$

$$d(x, y) = d(y, x) \qquad \textit{simetría} \tag{2.3}$$

$$d(x, z) \leq d(x, y) + d(y, z) \qquad \textit{desigualdad triangular} \tag{2.4}$$

#### Tiempo de inicio y de vencimiento

En este documento empleamos frecuentemente los términos *tiempo de inicio* y *tiempo de vencimiento*. $r_i$ es el tiempo de inicio en que una tarea $i$ puede ser realizada por un agente reparador. $d_i$ es el tiempo de vencimiento en el que dicha tarea puede ser realizada.

#### Ventana de tiempo

Empleamos el término *ventana de tiempo* para referirnos al intervalo de tiempo $[r_i, d_i]$ en el cual la tarea $i$ está disponible para ser realizada. En algunas secciones de este trabajo se estudian casos del problema del reparador viajero en los cuales está presente sólo uno de los dos tiempos. Cuando nos referimos a una ventana de tiempo compuesta forzosamente por los dos tiempos, la llamamos *ventana de tiempo general*.

#### Ventanas de tiempo unitarias

Empleamos el término *ventanas de tiempo unitarias* para hacer referencia a ventanas de tiempo cuya longitud (intervalo) de tiempo es igual a 1. Además, empleamos el término *ventanas de tiempo arbitrarias* para referirnos a ventanas de tiempo que tienen longitud arbitraria es decir, valores que pueden ser diferentes a 1 incluyendo a aquellos iguales a 1.

#### Tiempo de procesamiento

El *tiempo de procesamiento* es el monto de tiempo requerido para realizar o ejecutar una tarea. En algunas secciones de la tesis aparece el término *tiempo de procesamiento cero* que significa que la tarea en cuestión se ejecuta inmediatamente sin requerir tiempo alguno. En algunas secciones de la tesis aparece el término *tiempo de procesamiento general*, el cual usamos para denotar que cierta tarea puede tener tiempo de ejecución mayor o igual a cero.

#### Movimiento del reparador a máxima velocidad

Se emplea el término *movimiento del reparador a máxima velocidad* para indicar que el reparador se transporta, desde una tarea $i$ hacia otra tarea $j$, en tiempo instantáneo.

#### Latencia

Empleamos el término *latencia* para referirnos a la suma de retardos en que el reparador incurre. Un retardo se refiere al tiempo que el reparador espera antes de poder atender a una tarea.

### 2.1.6. Problema de la trayectoria monótona más larga (*Max-Monotone-Tour*)

El problema de la trayectoria monótona más larga (*Max-Monotone-Tour*) es un problema geométrico. Este problema se describe a continuación, debido a que en los capítulos 3 y 4 se mencionan algoritmos que usan como parte de su proceso una reducción a este problema.

En *Max-Monotone-Tour* se tiene un conjunto de $n$ segmentos, cada segmento $s_i$ con los siguientes atributos:

- $x_i$ y $y_i$ son las coordenadas iniciales de cada segmento $s_i$. Los valores de estas coordenadas son números reales.
- $l_i$: longitud del segmento $s_i$.
- $c_i$ es la pendiente del segmento $s_i$. En este trabajo manejamos las inclinaciones de los segmentos a 45° respecto al eje de las abcisas.
- Un punto inicial $p_0 \in \mathbb{R}^2$ donde comienza la trayectoria.
- Un punto final $p_f \in \mathbb{R}^2$ donde termina la trayectoria.

El objetivo es encontrar una trayectoria $x(y)-$monótonamente creciente tal que ésta interseque al máximo número de segmentos. La trayectoria debe iniciar en el punto $p_0$ y atravesar el plano hasta el punto $q_0$.

En el problema del reparador con ventanas de tiempo, el punto inicial $p_0$ serán las coordenadas $(0, 0)$. El punto final $p_f$ será un punto localizado geométricamente arriba de todos los segmentos.

### 2.1.7. Problemas de orientación

El término *problema de orientación* proviene del juego de deporte de orientación donde competidores individuales comienzan en un punto de control específico, tratan de visitar tantos puestos de control como sea posible y vuelven al punto de control en un plazo determinado [Chao et al., 1996].

En un problema de orientación se da un conjunto $V = \{v_1, \ldots, v_N\}$ de $N$ vértices, cada uno con una ganancia *score* $S_i$, el punto inicial y el punto final. También se conoce el tiempo $t_{ij}$ necesario para recorrer del vértice $i$ al vértice $j$ para todas las parejas de vértices. No necesariamente todos los vértices deben visitarse ya que el tiempo disponible está limitado por un factor de tiempo $T_{max}$. El objetivo de un problema de orientación es determinar una ruta, limitada por $T_{max}$, que visite algunos de los vértices, de manera que maximice la ganancia total recolectada. Las ganancias son aditivas y cada vértice puede visitarse a lo más una vez [Vansteenwegen et al., 2011, p. 14].

#### Enfoque matemático del problema de orientación

Acorde a [Vansteenwegen et al., 2011], un problema de orientación puede ser definido con la ayuda de un grafo $G = (V, A)$ donde $V = \{v_1, \ldots, v_N\}$ es el conjunto de vértices y $A$ es el conjunto de aristas. La ganancia $S_i$ es asociada con cada vértice $v_i \in V$ y el tiempo de viaje $t_{ij}$ se asocia con cada arista $a_{ij} \in A$. Sea $G' \subset G$, el problema de orientación consiste en determinar una trayectoria hamiltoniana en $G'$ incluyendo los vértices preestablecidos de inicio y fin, $v_1$ y $v_N$, y teniendo una longitud que no exceda $T_{max}$ para que se maximice la ganancia recolectada [Vansteenwegen et al., 2011, p. 3].

Un problema de orientación se puede formular como un problema de programación entera como se muestra enseguida. Se usan las siguientes variables: $x_{ij}$ es una variable binaria que tendrá valor igual a 1 si tras una visita al vértice $i$ sigue una visita al vértice $j$, en otro caso tendrá valor igual a 0; $u_i$ es la posición del vértice $i$ en la ruta.

Calcular

$$max \sum_{i=2}^{N-1} \sum_{j=2}^{N} S_i x_{ij}, \tag{2.5}$$

sujetos a que

$$\begin{aligned}
& \sum_{j=2}^{N} x_{1j} = \sum_{i=1}^{N-1} x_{iN} = 1, \\
\forall k = 2, \cdots N-1 : \quad & \sum_{i=1}^{N-1} x_{ik} = \sum_{j=2}^{N} x_{kj} \leq 1, \\
& \sum_{i=1}^{N-1} \sum_{j=2}^{N} t_{ij} x_{ij} \leq T_{max}, \\
\forall i = 2, \cdots, N : \quad & u_i \in \{2, 3, \ldots, N\}; \\
\forall i, j = 2, \cdots, N : \quad & u_i - u_j + 1 \leq (N-1)(1 - x_{ij}); \\
\forall i, j = 1, \cdots N, x \neq j : \quad & x_{ij} \in \{0, 1\};
\end{aligned} \tag{2.6}$$

La función objetivo (2.5) busca que se maximice la ganancia recolectada sujeta a las restricciones en (2.6). La primera ecuación en las restricciones garantiza que la trayectoria comience en el vértice 1 y termine en el vértice $N$. La segunda ecuación garantiza que la trayectoria visite los vértices una sola vez. La tercera ecuación asegura que sólo se use el tiempo limitado disponible. Las restricciones cuarta y quinta son necesarias para evitar ciclos aislados. La última ecuación de las restricciones restringe los valores de las variables.

### Enfoque matemático del problema de orientación con ventanas de tiempo

Los problemas de orientación con ventanas de tiempo, OPTW (*Orienteering Problems with Time Windows*), se han estudiado con mayor atención en los últimos años debido a su aplicación directa en problemas de calendarización de tareas y enrutamiento de repartidores [Vansteenwegen et al., 2011, p. 6]. Es posible resolver problemas de orientación con técnicas que brinden buenos resultados, pero dichas técnicas no darán resultados eficientes en un OPTW donde existen ventanas de tiempo. El problema del reparador viajero con ventanas de tiempo es un tipo de problema OPTW.

En el OPTW, a cada vértice se le asigna una ventana de tiempo $[O_i, C_i]$ y una visita a un vértice sólo puede comenzar durante esta ventana de tiempo. El OPTW puede modelarse como un problema de programación entero. Se usan las siguientes variables: $x_{ij}$ es una variable binaria que tendrá valor igual a 1 si tras una visita al

vértice $i$ sigue una visita al vértice $j$, en otro caso tendrá valor igual a 0; $y_i$ toma el valor 1 si se visita el vértice $i$, en otro caso es igual a 0; $s_i$ indica el inicio del servicio en el vértice $i$ [Vansteenwegen et al., 2011, p. 7].

Calcular

$$max \sum_{i=2}^{N-1} \sum_{J=2}^{N} S_i x_{ij}, \tag{2.7}$$

sujetos a que

$$\begin{aligned}
& \sum_{j=2}^{N} x_{1j} = \sum_{i=1}^{N-1} x_{iN} = 1, \\
\forall k = 2, \cdots N-1 : \quad & \sum_{i=1}^{N-1} x_{ik} = \sum_{j=2}^{N} x_{kj} \leq 1, \\
& \sum_{i=1}^{N-1} \sum_{j=2}^{N} t_{ij} x_{ij} \leq T_{max}, \\
\forall i = 1, \cdots, N : \quad & O_i \leq s_i \\
\forall i = 1, \cdots, N : \quad & s_i \leq C_i \\
\forall i, j = 1, \cdots N, x \neq j : \quad & x_{ij} \in 0, 1;
\end{aligned} \tag{2.8}$$

La función objetivo (2.7) busca que se maximice la ganancia recolectada sujeta a las restricciones en (2.8). La primera ecuación de las restricciones garantiza que la ruta comience en el vértice 1 y termine en el vértice $N$. La segunda ecuación garantiza que la trayectoria visite los vértices una sola vez. La tercera ecuación asegura que sólo se use el tiempo disponible. Las ecuaciones cuarta y quinta restringen el inicio del servicio dentro de la ventana de tiempo.

### 2.1.8. Problema del reparador viajero con ventanas de tiempo

El problema del reparador con ventanas de tiempo (TRPTW: *Traveling Repairman Problem with Time Windows*) es un problema de orientación con ligeras diferencias. La función objetivo de TRPTW es diferente a la función objetivo de un problema de orientación.

A su vez, existen diversas variantes de TRPTW. El caso especial de TRPTW que estudiamos en este trabajo consiste en un reparador que debe atender tareas o servicios, las cuales están ubicadas sobre una línea. Una instancia de TRPTW sobre una línea[1] consiste de:

- Un conjunto $Y = \{y_1, \ldots, y_n\}$ de $n$ tareas.
- Cada tarea $y_i$ tiene un valor $x_i \in \mathbb{R}$ indicando su ubicación.

[1]El término *sobre una línea* indica que tiene valores en la recta real.

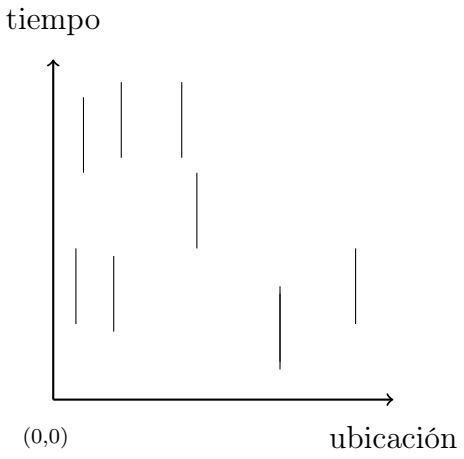


Figura 2-1: Instancia de TRPTW sobre un plano bidimensional, cada segmento vertical tiene longitud unitaria.

- Cada tarea tiene asociada una ventana de tiempo $w_i \in \mathbb{R}$ indicando su disponibilidad de tiempo. En este trabajo nos enfocamos en el caso particular donde todas las ventanas de tiempo son iguales a 1 y cada tarea es realizada en tiempo de procesamiento igual a 0.

Un reparador inicia su recorrido en una tarea inicial en tiempo 0, y arriba en tiempo $t_i$ para completar la tarea $y_i$. El reparador ocupa un monto de tiempo dado por una función $move(x_i, x_j) = |x_i - x_j|$ para moverse de la tarea $y_i$ a la tarea $y_j$. El reparador busca maximizar el número de tareas completadas. En [Bar-Yehuda et al., 2005] proponen una representación de instancias de TRPTW sobre un plano bidimensional. El eje de las abcisas representa la ubicación de cada tarea, el eje de las ordenadas representa el tiempo. La figura 2-1 es una instancia de TRPTW representada en un plano bidimensional.

Una solución óptima de TRPTW es una calendarización $\tau$ de tareas. $\tau$ es un subconjunto maximal de $Y$ tal que $\tau = \{x_1, \cdots, x_h\}$, $h$ es la cardinalidad del subconjunto. Además, el subconjunto satisface lo siguiente:

$$\forall i \in \mathbb{N} \cap [0, h] : \ t_{i+1} \geq t_i + move(x_i, x_{i+1}).$$

La figura 2-2 es una solución de TRPTW. Cualquier trayectoria monótonamente creciente respecto al eje vertical, con pendiente en cualquier momento en el intervalo $[-1, 1] = [\tan(135°), \tan(45°)]$ (y que capture el mayor número de segmentos), es una trayectoria factible [Bar-Yehuda et al., 2005].

### Posibles enfoques de solución

En las subsecciones siguientes se describen brevemente algunas técnicas mediante las cuales es posible abordar el problema del reparador viajero: algoritmos voraces y

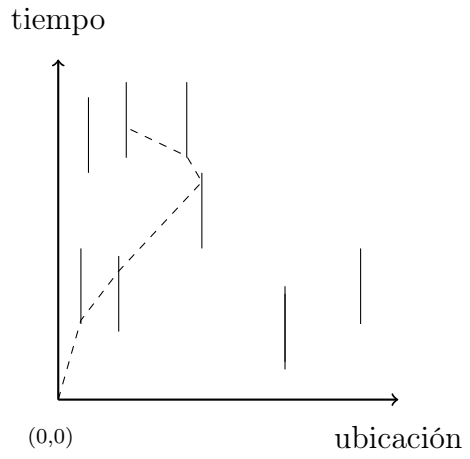


Figura 2-2: Solución óptima para una instancia de TRPTW. Los segmentos verticales son el conjunto de tareas $Y = \{y_1, \ldots, y_n\}$. La línea discontinua indica las tareas que deben ser capturadas (conjunto $\tau$). Para esta instancia son capturados 5 segmentos.

programación lineal entera.

### Algoritmos voraces

La técnica voraz consiste en resolver un problema incrementalmente por etapas, tomando decisiones o realizando acciones que permitan construir progresivamente la solución del problema. Esta técnica se utiliza en problemas de optimización, en los cuales hay que buscar la solución óptima en un espacio de búsqueda, el cual suele estructurarse en forma de árbol. En cada punto de decisión, un algoritmo voraz toma la decisión que le parezca mejor en ese momento. Los algoritmos voraces obtienen soluciones óptimas en la mayoría de las veces respecto a otros tipos de algoritmos. Un problema que se pretenda resolver con este enfoque, debe cumplir las propiedades de *Greedy-choice* y de sub estructura óptima [Cormen et al., 2009, p. 423].

En la sección 16.3 del texto [Cormen et al., 2009] se presenta una estrategia voraz para un problema de calendarización de tareas donde se requiere calendarizar $n$ tareas unitarias en un procesador que las ejecuta inmediatamente. El enfoque de solución presentado en [Cormen et al., 2009] reduce las instancias del problema de calendarización de tareas a una representación de matroides. Un matroide es una pareja ordenada $M = (S, \mathcal{I})$ que satisface las siguientes tres propiedades [Cormen et al., 2009]:

1. $S$ es un conjunto finito.

2. $\mathcal{I}$ es una familia de subconjuntos de $S$ no vacíos, denominados conjuntos independientes de $S$ que cumplen la propiedad de herencia. La propiedad de herencia se define a continuación: sean $A$, $B$ dos conjuntos de elementos, si $B \in \mathcal{I}$ y $A \in B$, entonces $A \in \mathcal{I}$.

3. $M$ satisface la propiedad de intercambio. Si $A \in \mathcal{I}$, $B \in \mathcal{I}$ y $|A| < |B|$, entonces existe algún elemento $x \in B - A$ tal que $A \cup \{x\} \in \mathcal{I}$.

**Programación Lineal Entera**

El problema del reparador viajero con ventanas de tiempo es un problema combinatorio, el cual puede ser abordado mediante programación lineal entera (ILP por sus siglas en inglés). Por esta razón, a continuación damos una breve descripción de esta técnica de resolución de problemas.

La programación lineal es una técnica que permite modelar problemas mediante ecuaciones lineales. Los modelos de programación entera son una extensión de los modelos de programación lineal en los cuales algunas variables toman valores enteros. Cuando todas las variables deben tomar valores enteros, se trata de un problema de programación lineal entera *puro*, en caso contrario se dice que es un problema *mixto*. Se dice que una variable es *binaria* si puede tomar los valores 0 y 1.

Un problema de programación lineal entera se compone de:

- Una función objetivo a optimizar, ésta puede ser de maximización o minimización.
- Un conjunto de restricciones que deben ser satisfechas para que la solución sea óptima.
- Un intervalo de valores admisibles para cada variable del modelo.

Esta técnica puede ser usada en el diseño de algoritmos de aproximación ya que la mayoría de los problemas de optimización pueden ser formulados como problemas de programación lineal entera. Esta formulación equivalente no simplifica tales problemas, simplemente permite diseñar algoritmos que ofrecen soluciones con una tasa de aproximación adecuada [Ausiello et al., 1999].

Si se busca plantear un enfoque de programación lineal entera para el problema del reparador viajero con ventanas de tiempo, la función objetivo sería una función de maximización ya que se busca atender el mayor número de tareas acotadas por las ventanas de tiempo.

## 2.2. Estado de conocimiento del problema del reparador viajero con ventanas de tiempo

La subsección 2.2.1 expone el orígen de TRPTW como problema derivado del agente viajero. En la subsección 2.2.2 se revisa la literatura de TRPTW; se presentan diversos enfoques de solución y algoritmos propuestos por varios científicos así como algunos problemas que aún están abiertos. La subsección 2.2.3 presenta una clasificación de TRPTW que hemos propuesto con base a los algoritmos revisados en la literatura. En la subsección 2.2.4 se describe los casos del problema que son tratados en este trabajo.

| | Tiempo de procesamiento cero | Tiempo de procesamiento general |
|---|---|---|
| Sin tiempo de inicio ni tiempo de vencimiento | trivial | trivial |
| Sólo tiempo de inicio | $O(n^2)$ | NP-Completo |
| Sólo tiempo de vencimiento | $O(n^2)$ | **?** |
| Ventana de tiempo general | NP-Completo | NP-Completo |

Tabla 2.1: Complejidad de casos especiales de Line-TSPTW (*Traveling Salesman Problem with Time Windows*) ($n$ es el número de trabajos).

### 2.2.1. Orígen

TRPTW se deriva del problema del agente viajero (TSP); análogamente en TRPTW las ciudades son tareas que deben ser completadas por un reparador, además, se agregan restricciones de tiempo, lo cual aumenta la complejidad computacional. TRPTW en su versión general, es decir, cuando existen tiempos de inicio y de vencimiento de las tareas, es un problema NP-Completo [Afrati et al., 1986].

En [Tsitsiklis, 1992] se estudian diversas variantes del problema del agente viajero. La primera de esas variantes tiene restricciones de tiempo, se denomina Line-TSPTW: *Traveling Salesman Problem with Time Windows on a line*. En Line-TSPTW el objetivo es minimizar el tiempo total que ocupa el reparador en completar todas las tareas. En otra variante, denominada Line-TRPTW: *Traveling Repairman Problem with Time Windows on a line* el objetivo es minimizar la sumatoria de tiempos de espera de las tareas, es decir, que la suma de tiempos de retraso en alcanzar cada tarea sea mínima.

### 2.2.2. Estudios del problema del reparador viajero con ventanas de tiempo

En [Tsitsiklis, 1992] se estudia la complejidad computacional de diversas variantes de TRPTW, por ejemplo, cuando hay tiempos de inicio o de vencimiento para cada tarea, o cuando las tareas tienen tiempos de procesamiento. Sin embargo, aún existen variantes del problema para los cuales no se ha determinado la complejidad computacional.

Las tablas 2.1 y 2.2 muestran la complejidad computacional de dos variantes de TRPTW, tales variantes son Line-TSPTW y Line-TRPTW, respectivamente, las cuales fueron descritas en la sección anterior. La primer columna de las tablas indica el tipo de ventana de tiempo de las tareas. La fila superior indica si las tareas tienen tiempo de procesamiento. Aparece un signo de interrogación en la celda de aquellos problemas para los cuales no se ha estimado la complejidad computacional y por tanto aún están abiertos [Tsitsiklis, 1992].

| | Tiempos de procesamiento cero | Tiempos de procesamiento general |
|---|---|---|
| Sin tiempos de inicio ni Tiempos de vencimiento | $O(n^2)$ | **?** |
| Sólo tiempos de inicio | **?** | NP-Completo |
| Sólo tiempos de vencimiento | NP-Completo seudo polinomial | NP-Completo |
| Ventanas de tiempo generales | NP-Completo | NP-Completo |

Tabla 2.2: Complejidad de casos especiales de Line-TRPTW ($n$ es el número de trabajos).

En [Bar-Yehuda et al., 2005] se abordan dos variantes de TRPTW[2]:

1. TRPTW con métrica. En esta variante las tareas tienen una ubicación en la recta real [3], no contemplan tiempos de procesamiento y tienen tiempos de inicio y finalización. La métrica, o función de distancia, modela el monto de tiempo requerido para viajar de una tarea a otra.

2. Versión no métrica de TRPTW. En esta variante la función de distancia no es una métrica. Esta variante no es de nuestro interés, sin embargo, hacemos mención de esta.

En la primera variante, los autores proponen un algoritmo para resolver TRPTW en el caso particular en que las ventanas de tiempo son unitarias. Esto lo consiguen transformando TRPTW al problema geométrico de la trayectoria monótona más larga (*Max-Monotone-Tour*). En primer lugar, se representa geométricamente la instancia de TRPTW en un plano bidimensional. El eje $x$ corresponde al valor de un punto, el eje $y$ al tiempo. Una ventana de tiempo $[r(v), d(v)]$ de un punto $v$ es representada como un segmento vertical; los puntos de cada segmento son $(v, r(v)), (v, d(v))$. Posteriormente, se rota el plano 45 grados, el objetivo es encontrar una curva monótona respecto al eje $x$ que inicie en el orígen y que interseque tantos segmentos como sea posible. Los autores resuelven *Max-Monotone-Tour* mediante un algoritmo con garantía de $O(\log n)$-aproximación donde $n$ es el número de tareas. El enfoque propuesto emplea una cuadrícula para resolver *Max-Monotone-Tour* (en la sección 3.2.1 se detalla este proceso), resultando en un algoritmo que resuelve TRPTW con garantía de 8-aproximación [Bar-Yehuda et al., 2005].

En la segunda variante, se aborda un caso general de distancias sin métrica entre ubicaciones. Se define un parámetro $d$ de densidad que limita el número de visitas de ida y vuelta entre ubicaciones y se propone un algoritmo de programación dinámica que encuentra una trayectoria que visita al menos $\frac{k}{d}$ ubicaciones donde $k$ es el número

[2]En el texto los autores se refieren al problema como TSPTW.
[3]las ubicaciones toman valores del conjunto de los números reales.

de soluciones óptimas para esa variante del problema. Este algoritmo puede extenderse a problemas con ventanas de tiempo arbitrarias y que contemplan tiempos de procesamiento [Bar-Yehuda et al., 2005].

Otras variantes de TRPTW también son estudiadas en [Frederickson and Wittman, 2007] donde proponen algoritmos de aproximación que tienen complejidad de tiempo polinomial para dos variantes del problema:

1. El objetivo es encontrar una trayectoria que visite el máximo número posible de lugares durante la vigencia de sus ventanas de tiempo.

2. El objetivo es encontrar una trayectoria que consuma el mínimo de velocidad posible para visitar todos los lugares durante sus ventanas de tiempo. Esta variante es denominada el problema del reparador veloz (*Speeding Deliveryman Problem*).

Para ambas variantes, las ventanas de tiempo son de longitud unitaria y la métrica (función distancia) se basa en un grafo con pesos. Para la primera variante, se propone un algoritmo de 3-aproximación y de complejidad $O(n^4)$ para el caso en que el reparador trabaja sobre un árbol (es decir, las tareas no están sobre una línea recta). Se propone un algoritmo de $6 + \epsilon$-aproximación y de complejidad $O(n^4)$ para el caso del reparador que trabaja sobre un grafo [Frederickson and Wittman, 2007].

Para la segunda variante, el problema del reparador veloz, también se proponen algoritmos de aproximación. Los algoritmos propuestos pueden extenderse a problemas con ventanas de tiempo arbitrarias, donde la longitud esté en un rango acotado. [Frederickson and Wittman, 2007]. Sin embargo, no abordamos estas variantes en este trabajo.

En [Perez Perez et al., 2014] se estudia el caso especial de TRPTW donde las ubicaciones están sobre una línea, el tiempo de procesamiento de cada tarea es igual a cero y la longitud de las ventanas de tiempo de las tareas son unitarias. Aunque el problema del reparador viajero es NP-difícil, la complejidad del caso con ventanas unitarias sigue siendo desconocido[Perez Perez et al., 2014] . Los autores presentan un algoritmo de 4-aproximación de complejidad $O(n^2)$ basado en el algoritmo propuesto por [Bar-Yehuda et al., 2005] de 8-aproximación (también de complejidad $O(n^2)$). La mejora en la tasa de aproximación se debe a que en [Perez Perez et al., 2014] se evita el problema de doble conteo[4].

La tabla 2.3 resume algunos algoritmos, descritos en la literatura antes revisada, que resuelven el problema del reparador viajero con ventanas de tiempo.

### 2.2.3. Variantes de TRPTW

En la literatura hay estudios de diferentes casos de TRPTW. Para clasificar los tipos de problemas de TRPTW, proponemos hacerlo de acuerdo a su espacio de métrica como se observa a continuación.

[4]El problema de doble conteo se explica en la subsección 3.2.2. La figura 3-3 muestra un ejemplo de este problema.

| Garantía | Complejidad | Referencia |
|---|---|---|
| 8-aproximación (sobre línea) | $O(n^2)$ | [Bar-Yehuda et al., 2005] |
| 4-aproximación (sobre línea) | $O(n^2)$ | [Perez Perez et al., 2014] |
| 3-aproximación (sobre árbol) | $O(n^4)$ | [Frederickson and Wittman, 2007] |
| $(6+\epsilon)$-aproximación (sobre grafo) | $O(n^4)$ | [Frederickson and Wittman, 2007] |

Tabla 2.3: Algunos algoritmos de aproximación existentes para el problema del reparador viajero con ventanas de tiempo.

**Métrica**

Por el espacio de métrica de la función objetivo, se puede catalogar las siguientes tres variantes de TRPTW:

- Por la distancia: se busca minimizar la distancia total recorrida por el agente reparador. Algunos estudios de esta variante están en [Afrati et al., 1986], [Wu et al., 2004], [Sitters, 2013], [Fakcharoenphol et al., 2003].
- Por la latencia del tiempo: se busca minimizar el tiempo total consumido en atender los servicios por el reparador. Un estudio de esta variante está en [Arora and Karakostas, 1999].
- Por el número de tareas a completar: se busca maximizar el número de tareas completadas por el agente reparador. Identificamos, a su vez, los siguientes casos.
  - Las tareas (segmentos) son de longitud unitaria. Ver [Bar-Yehuda et al., 2005] y [Perez Perez et al., 2014].
  - Las tareas tienen una función de peso (beneficio) por ser capturadas. Ver [Dewilde et al., 2010].
  - Las tareas unitarias comienzan en el inicio. Ver [Tsitsiklis, 1992].
  - Las tareas tienen tiempo de inicio, pero terminan en tiempo infinito. Ver [Tsitsiklis, 1992].
  - Las tareas son de longitud arbitraria. Ver [Tsitsiklis, 1992].

### 2.2.4. Variantes del problema tratadas en este trabajo

En este trabajo se aborda el caso especial de TRPTW en el cual la función objetivo consiste en que se maximice el número de segmentos (tareas) que el reparador debe completar. Sin embargo, hay casos especiales dentro de esta variante, que son abordados en este trabajo. Estos casos son:

1. Ventanas de tiempo con longitud unitaria (el caso de TRPTW que abordan en [Perez Perez et al., 2014]).

2. Ventanas de tiempo con longitud arbitraria.

Adicionalmente, se analizan y comparan los algoritmos propuestos por [Bar-Yehuda et al., 2005] y [Perez Perez et al., 2014].

# Capítulo 3

# Implementación de algoritmos conocidos

En la primera parte de este capítulo presentamos algunos problemas de calendarización de tareas y el enfoque de solución de éstos. El objetivo es mostrar que algunas técnicas, como los algoritmos voraces, resultan no aptos para modelar problemas simples de calendarización de tareas. En la segunda sección de este capítulo se presentan implementaciones de dos algoritmos de la literatura, los cuales abordan TRPTW con ventanas de tiempo unitarias. Además, se presenta una breve comparación de estos algoritmos.

La sección 3.2 de este capítulo da la pauta para proponer un enfoque de programación entera, el cual es explicado en el capítulo 4.

## 3.1. Problemas relacionados de calendarización de tareas

En esta sección presentamos tres problemas de calendarización de tareas que merece la pena mencionar para comprender que algunas técnicas, como los algoritmos voraces, quedan limitados para abordar tales problemas, y por consecuencia a TRPTW. Estos problemas son:

- Problema de calendarización de tareas unitarias con vigencia y penalizaciones para un procesador.
- Problema de calendarización de intervalos de tiempo.
- Problema de calendarización de intervalos con peso.

### 3.1.1. Problema de calendarización de tareas unitarias con vigencia y penalizaciones para un procesador

En la sección 16.5 del texto de [Cormen et al., 2009] se describe el problema de calendarización de tareas unitarias con vigencia y penalizaciones para un procesador *(scheduling unit-time tasks with deadlines and penalties for a single processor)*, el cual consiste de:

- un conjunto $S = \{a_1, a_2, \ldots, a_n\}$ de $n$ tareas unitarias.
- un conjunto de $n$ tiempos de vencimiento $\{d_1, d_2, \ldots, d_n\} \in \mathbb{N}$ asociados a las tareas, tal que $d_i$ satisface $1 \leq d_i \leq n$ y cada tarea $a_i$ debe terminar antes del tiempo $d_i$.
- un conjunto de pesos o penalidades no negativas $\{w_1, w_2, \ldots, w_n\} \in \mathbb{R}^+$ tal que se incurre en una penalidad $w_i$ si la tarea $a_i$ no se realiza antes del tiempo $d_i$. A esto se le denomina tarea vencida.

El objetivo del problema es encontrar una calendarización para $S$ que minimize la penalidad total derivada de las tareas vencidas.

#### Algoritmo voraz

Para resolver el problema en cuestión, se emplea el algoritmo 1 de tipo voraz propuesto en [Cormen et al., 2009], el cual emplea el concepto de *matroide*. Un *matroide* es una pareja ordenada de elementos $M = (S, \mathcal{I})$ que satisface las tres propiedades siguientes [Cormen et al., 2009, p. 437]:

1. $S$ es un conjunto finito.
2. $\mathcal{I}$ es una familia de subconjuntos de $S$ no vacíos, denominados conjuntos independientes de $S$ y cumplen la propiedad de herencia.
3. $M$ satisface la propiedad de intercambio.

En [Cormen et al., 2009] enuncian un teorema que establece que si $M = (S, \mathcal{I})$ es un matroide con una función de peso $w$, entonces el algoritmo 1 obtiene un subconjunto óptimo. La demostración de dicho teorema puede consultarse en [Cormen et al., 2009, p. 442]

**Algorithm 1** Calendarización de tareas

**input**: $M(S, I), w$: estructura de matroide donde $S$ es el conjunto de tareas. $w$ es una función de peso asociada a cada tarea.
**output**: $A$: conjunto de tareas aceptadas.
$A = \emptyset$.
Ordena $M.S$ en orden monótonamente decreciente por peso $w_i$.
**for** cada $x \in M.S$ tomada en orden decreciente por peso $w(x)$ **do**
  **if** $(A \cup \{x\} \in M.\mathcal{I})$.
  **then** $A \cup \{x\}$.
**end for**
**return** A

Se implementó el algoritmo 1. El código fuente de esta implementación puede obtenerse siguiendo las instrucciones de la sección 5.2.

### 3.1.2. Problema de calendarización de intervalos de tiempo

El problema de calendarización de intervalos (*Interval Scheduling Problem*) consiste de:

- Un conjunto de solicitudes $R = \{1, 2, \ldots, n\}$ donde la $i$-ésima solicitud corresponde a un intervalo de tiempo.
- Cada solicitud comienza en un tiempo de inicio $s_i$ y finaliza en un tiempo de finalización $f_i$. Ambos son números naturales.

Un conjunto de solicitudes es compatible si no hay dos de éstas traslapandose en el mismo tiempo. El objetivo es aceptar el mayor número de solicitudes compatibles, es decir, el conjunto compatible de tamaño máximo.

#### Implementación mediante algoritmo voraz

Se implementó el algoritmo 2, propuesto en [Kleinberg and Tardos, 2005, p. 118]. Es un algoritmo voraz que consiste en aceptar, en primer lugar, a la solicitud que termina primero, es decir aquella solicitud $i$ para la cual $f(i)$ tiene el valor más pequeño, esto es una idea natural, pues se asegura que el recurso se libera tan pronto como sea posible sin dejar de cumplir su petición, lo que permite maximizar el tiempo restante para satisfacer las otras peticiones [Kleinberg and Tardos, 2005].

La complejidad del algoritmo es $O(n \log n)$ [Kleinberg and Tardos, 2005]. Se realizaron pruebas de la implementación con instancias del problema cuyos valores fueron generados aleatoriamente. El código fuente de esta implementación puede obtenerse siguiendo las instrucciones de la sección 5.2.

**Algorithm 2** Calendarización de intervalos

**input**: $R$: conjunto de solicitudes.
**output**: $A$: conjunto de solicitudes aceptadas.
$A = \emptyset$.
**while** $R \neq \emptyset$ **do**
    Escoger la solicitud $i \in R$ con el menor tiempo de finalización.
    $A = A \cup i$.
    Borrar las solicitudes en $R$ que no son compatibles con $i$.
**end while**
**return** A.

### 3.1.3. Problema de calendarización de intervalos de tiempo con peso

El problema de calendarización de intervalos con peso (*Weighted Interval Scheduling*) se extiende del problema de calendarización de intervalos de tiempo. En este problema cada intervalo tiene asignado un valor (peso), y se busca aceptar un conjunto de peso máximo. La complejidad aumenta considerablemente al agregar pesos, lo que hace que no sea posible resolver el problema mediante un algoritmo voraz, pero sí mediante otras técnicas, como programación dinámica [Kleinberg and Tardos, 2005].

Una instancia del problema de calendarización de intervalos de tiempo con peso consiste de:

- Un conjunto $R = \{1, \ldots, n\}$ de $n$ solicitudes (intervalos de tiempo).
- Cada solicitud tiene un tiempo de inicio $s_i$ y un tiempo de finalización $f_i$. Ambos son números naturales.
- Cada solicitud $i$ tiene asignado un peso $v_i \in \mathbb{R}^+$.

El objetivo en este problema es seleccionar un subconjunto $S \in \{1, \ldots, n\}$ de intervalos mutuamente compatibles que maximice la suma de valores de los intervalos seleccionados $\sum_{i \in S} v_i$. Dos intervalos son compatibles si éstos no se traslapan.

#### Implementación mediante programación dinámica

Para resolver este problema, implementamos el algoritmo 3 propuesto por [Kleinberg and Tardos, 2005]. Mencionaremos a continuación una breve descripción de cómo invocar a este algoritmo, sin embargo, no es la idea explicar su funcionamiento. Una descripción detallada puede consultarse en la fuente [Kleinberg and Tardos, 2005, p. 256].

El primer paso es ordernar los intervalos de tiempo por sus tiempos de finalización en orden ascendente $f_1 \leq f_2 \leq \cdots \leq f_n$. La solicitud $i$ está antes de la solicitud $j$ si $i \leq j$. Para una solicitud $j$, se define $p(j)$ como el índice más grande $i \leq j$ tal que los

intervalos de tiempo $i$ y $j$ son disjuntos, es decir, $i$ termina antes de que $j$ comience [Kleinberg and Tardos, 2005].

Acorde a [Kleinberg and Tardos, 2005, p. 256], el algoritmo 3 de programación dinámica, resuelve el problema en cuestión. Para determinar la solución óptima para el conjunto de solicitudes, se invoca a CALCULATE-OPT(n) donde $n$ es el $n$-ésimo intervalo. El algoritmo emplea un arreglo global[1] $M[0, \ldots, n]$. La primera vez que se invoca el algoritmo, $M[j]$ está vacío, entonces se calcula el valor de su solución óptima y la almacena en $M[j]$ para que el valor de esa solución esté accesible en posteriores llamadas y no se consuma recursos volviendo a calcularla, lo cual es una característica típica de un algoritmo de programación dinámica.

**Algorithm 3** Calculate-Opt

```
input: j: intervalo.
output: solución óptima para ese intervalo.
function CALCULATE-OPT(j)
    if j = 0 then
        return 0
    else
        if M[j] ≠ ∅ then
            return M[j]
        else
            M[j] = max(v_j + CALCULATE-OPT(p(j)), CALCULATE-OPT(j − 1))
            return M[j]
        end if
    end if
end function
```

Como el arreglo global tiene $n+1$ elementos, el número de veces que se invoca a la función CALCULATE-OPT es a lo más $n$, por lo tanto, la complejidad computacional del algoritmo es $O(n)$ [Kleinberg and Tardos, 2005].

La implementación se probó con instancias de valores generados aleatoriamente. El código fuente de esta implementación puede obtenerse siguiendo las instrucciones de la sección 5.2.

## 3.2. Problema del reparador viajero con ventanas de tiempo sobre una línea

En contraste con la sección anterior en la que se presentan implementaciones de problemas relacionados, y que de alguna forma son el preámbulo de TRPTW, en esta

[1] $M$ es un arreglo que almacena valores calculados de subproblemas. El uso de este arreglo es una característica de un algoritmo de programación dinámica para evitar la recursividad.

sección presentamos las implementaciones de 2 algoritmos existentes en la literatura para TRPTW sobre una línea. El primer algoritmo, propuesto por [Bar-Yehuda et al., 2005], encuentra soluciones aproximadas dentro de un factor de 8 de la solución óptima, mientras que el segundo algoritmo, de [Perez Perez et al., 2014], tiene un factor de 4. Ambos algoritmos tienen complejidad cuadrática. Las siguientes dos subsecciones describen la implementación de estos algoritmos.

### 3.2.1. Algoritmo de 8-aproximación con ventanas unitarias

En esta subsección presentamos el algoritmo de 8-aproximación de [Bar-Yehuda et al., 2005] que aborda el caso especial de TRPTW sobre una línea en el cual las ventanas de tiempo son de longitud unitaria y explicamos la implementación basada en este algoritmo. Los autores reducen el problema del reparador viajero a una instancia del problema *Max-Monotone-Tour*, esto lo consiguen de la siguiente forma.

Primero, la instancia de TRPTW sobre una línea se representa en un diagrama bidimensional; el eje $x$ representa las ubicaciones de las tareas, el eje $y$ representa el tiempo disponible de cada tarea. De esta forma cada tarea está representada por medio de un segmento vertical.

El plano se rota 45° en dirección de las manecillas del reloj, esto ocasiona que los segmentos verticales se conviertan en segmentos inclinados a 45 grados y que una trayectoria factible sea cualquier trayectoria que es monótonamente creciente con respecto a los ejes vertical y horizontal. Esto es una diferencia considerable respecto al plano original, en donde la trayectoria estaba caracterizada solamente con respecto al eje vertical. Los detalles de esta rotación pueden verse en la sección 4.1.

Después se coloca una cuadrícula sobre el plano, teniendo cuidado que los extremos de los segmentos inclinados no toquen a las aristas de la cuadrícula. Cada cuadrado de la cuadrícula mide $\frac{1}{\sqrt{2}} \times \frac{1}{\sqrt{2}}$ por lado, la razón de esta medida es que cada segmento inclinado, cuya longitud es igual a 1, interseque a lo más a un segmento horizontal y un segmento vertical de la cuadrícula.

#### Modelación mediante un grafo dirigido acíclico

Después de que la cuadrícula se colocó sobre el plano rotado, se construye un grafo dirigido acíclico $G = (V, A)$ donde los vértices $V$ son las intersecciones de los segmentos internos de la cuadrícula, las aristas $A$ son los segmentos verticales y horizontales de la cuadrícula. Los segmentos horizontales (paralelos con en el eje $x$) son aristas dirigidas hacia la derecha, mientras que los segmentos verticales (segmentos paralelos con el eje $y$) son aristas dirigidas hacia arriba. A cada arista $a$ tal que $a \in A$, se le asigna un peso $w(a)$ igual al número de segmentos inclinados que intersecan a $a$.

El último paso de este algoritmo consiste en resolver el problema del camino más largo (*longest path algorithm*) en $G$ para encontrar un camino $c$ sobre la cuadrícula que interseque el máximo número de segmentos inclinados. El camino $c$ es la trayectoria que el reparador debe seguir para completar el mayor número de tareas. El algoritmo 4

resume este procedimiento.

### Tasa de aproximación

En [Bar-Yehuda et al., 2005], los autores demuestran un teorema que enuncia que el algoritmo 4 encuentra soluciones con un factor de aproximación de 8 de la solución óptima, tal demostración sigue un procedimiento teórico que puede consultarse en [Bar-Yehuda et al., 2005, p. 80-83]. Es importante mencionar que el factor de aproximación del algoritmo es respecto a las soluciones óptimas de la instancia original de TRPTW, es decir, sea $p'$ una trayectoria en la cuadrícula (encontrada por el algoritmo), sea $p$ una trayectoria óptima en la instancia original de TRPTW (que no involucra a la cuadrícula), el factor de aproximación $p'$ está dentro de un factor de 8 de la solución de $p$.

### Implementación

Hemos implementado un programa que traduce instancias de TRPTW a instancias del problema *Max-Monotone-Tour* y posteriormente encuentra la trayectoria que captura el mayor número de segmentos acorde al algoritmo de 8-aproximación. La implementación fue realizada en el lenguaje de programación Java debido a que empleamos la biblioteca `algs4`, disponible para este lenguaje. Esta biblioteca es desarrollada por los autores del texto [Sedgewick and Wayne, 2011], y provee estructuras de datos para representación de grafos, así como funciones para hacer operaciones sobre grafos. La función que encuentra el camino más largo en el DAG consume tiempo $E + V$, donde $E$ es el número de aristas y $V$ el número de vértices del grafo [Sedgewick and Wayne, 2011, p. 661]. La implementación genera archivos de solución que son graficados mediante la herramienta `Gnuplot`.

**Algorithm 4** 8-aproximación para TRPTW

**input**: Conjunto $X$ de $n$ ubicaciones sobre una línea.

**output**: Camino que captura el mayor número de localidades sobre la cuadrícula resultante.

**function** TRP-8APPROXIMATION

1. Representar las ubicaciones en dos dimensiones; agregar el eje $y$ para representar el tiempo.

2. Rotar el plano 45° en dirección de las manecillas del reloj.

3. Colocar nuevos ejes cartesianos para trabajar en el primer cuadrante del plano rotado.

4. Colocar cuadrícula sobre el primer cuadrante. Los lados de cada cuadrado de la cuadrícula miden $\frac{1}{\sqrt{2}}$.

5. Construir el grafo dirigido acíclico con pesos $G = (V, A)$.

6. Asignar a cada arista de $A$ un peso igual al número de segmentos inclinados que interseca en la cuadrícula.

7. Resolver el problema *Max-Monotone-Tour*, con esto se obtiene el camino maximal monótono de $G$.

**return** Camino maximal monótono de $G$.

**end function**

Un aspecto importante en la implementación de este algoritmo, es el relacionado a la complejidad computacional del mismo. Mientras que los primeros pasos (1 al 4) consumen tiempo $O(n)$, la construcción del DAG (paso 5) consume tiempo $O(n^2)$, la asignación de aristas (paso 6) consume tiempo $O(n)$. El cálculo del camino más largo en el grafo $G$ (paso 7) se realiza en tiempo cuadrático gracias al algoritmo de programación dinámica de [Sedgewick and Wayne, 2011, p. 660-662]. Este algoritmo utiliza fundamentalmente el método para encontrar caminos más cortos en digrafos acíclicos, presentado también en [Sedgewick and Wayne, 2011], para ello multiplica el peso de las aristas del grafo por $-1$, de esta forma todas las aristas tienen peso negativo, y una solución del camino más corto en el grafo de aristas de peso negativo es equivalente a una solución del camino más largo en el grafo original. Es conveniente mencionar que este algoritmo es eficiente en contraste con los algoritmos más populares para encontrar caminos más largos en digrafos ponderados con pesos negativos, los cuales consumen tiempo exponencial en el peor de los casos [Sedgewick and Wayne, 2011, p. 661].

### Pruebas realizadas

Exponemos a continuación soluciones generadas por nuestra implementación para dos instancias de TRPTW. La figura 3-1 es la solución para una instancia de 4 tareas. El camino que completa el mayor número de tareas es un conjunto de aristas de la cuadrícula capturando a 3 tareas. La trayectoria inicia en el vértice $(0, 0)$ y atraviesa

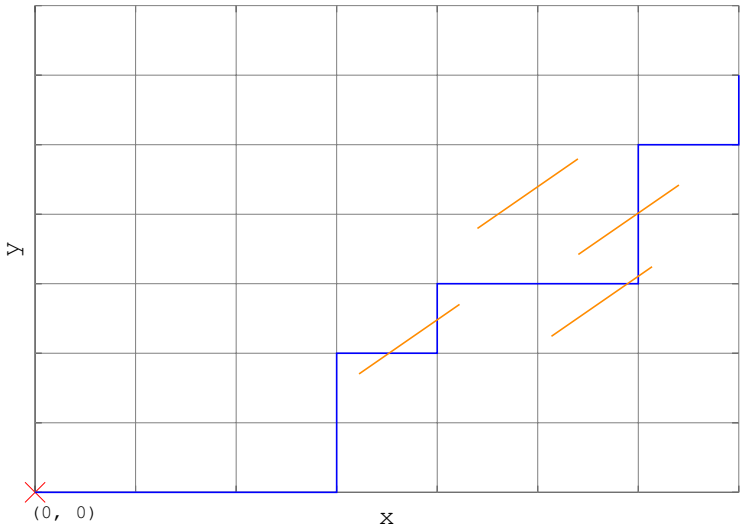


Figura 3-1: Solución obtenida mediante la implementación del algoritmo de 8 aproximación para una instancia de 4 tareas. La cuadrícula necesaria que cubre a los segmentos mide 9 vértices por lado. La mejor ruta completa 3 tareas.

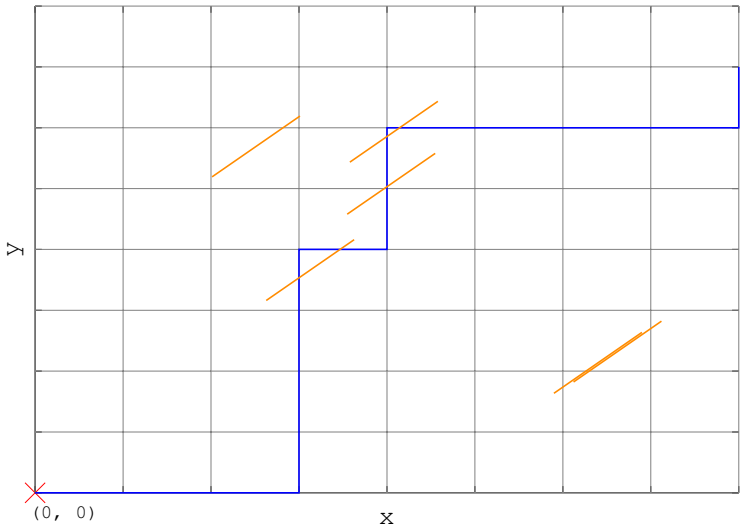


Figura 3-2: Solución obtenida mediante la implementación del algoritmo de 8 aproximación para una instancia de 6 tareas. La mejor ruta completa 3 tareas.

el plano[2], es decir, finaliza en la esquina superior derecha de la cuadrícula. La figura 3-2 es la solución para una instancia de 6 tareas. La mejor ruta que el reparador debe seguir completa sólo 3 tareas.

[2]Considerando una cuadrícula de tamaño $m \times m$ vértices, *atravesar el plano* indica que la trayectoria comienza en el orígen y finaliza en el vértice $(m, m)$.

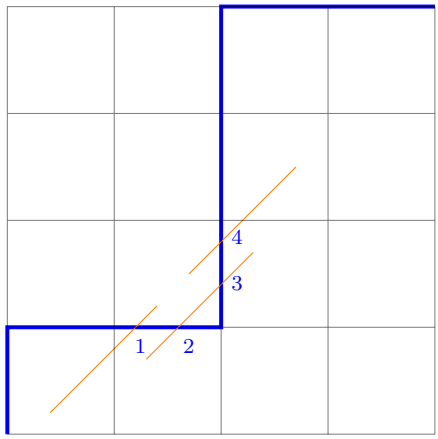


Figura 3-3: Problema de doble conteo. Uno de los segmentos inclinados es contado dos veces.

### 3.2.2. Algoritmo de 4-aproximación con ventanas unitarias

En esta sección explicamos el algoritmo de 4-aproximación de [Perez Perez et al., 2014] cuya tasa de aproximación es mejor y la complejidad computacional es la misma ($O(n^2)$).

#### Problema de doble conteo

En el algoritmo de 8-aproximación de la sección anterior, cada segmento inclinado interseca exáctamente una arista horizontal y una arista vertical de los cuadrados de la cuadrícula, por esta razón, un segmento inclinado, algunas veces, puede ser contado dos veces. Lo anterior se conoce como problema de doble conteo y se presenta cuando el camino que el reparador debe seguir en la cuadrícula contiene una vuelta en un cuadrado donde hay un segmento inclinado que interseca a dos aristas de ese cuadrado (ver la figura 3-3).

El algoritmo de [Perez Perez et al., 2014], algoritmo 5, evita el problema de doble conteo mediante el uso de un grafo dirigido acíclico, el cual es distinto al grafo del algoritmo de 8-aproximación. La construcción de este grafo la explicaremos a continuación acorde a los autores del algoritmo.

#### Construcción del grafo dirido acíclico

En la cuadrícula sobre el plano rotado a 45°, se agrega un nuevo vértice a la mitad de cada arista de los cuadrados que conforman la cuadrícula. Con estos vértices (en la cuadrícula y en la mitad de cada segmento) se construye un grafo dirigido acíclico. En este nuevo grafo $G' = (V', A')$, $V'$ es el conjunto de vértices tal que $V' = V'_h \cup V'_v$ y

$A'$ es el conjunto de aristas tal que $A' = A'_h \cup A'_v \cup A'_h v \cup A'_v h$. Enseguida explicamos la notación de cada variable acorde a los autores:

- $V'_h$ y $V'_v$ son los vértices horizontales y verticales correspondientes a los segmentos horizontales y verticales, respectivamente, en la cuadrícula original.

- $A'_h$ es el conjunto de aristas que unen dos vértices adyacentes de $V'_h$. Estas aristas son horizontales y tienen dirección hacia el este (derecha).

- $A'_v$ es el conjunto de aristas que unen dos vértices adyacentes de $V'_v$. Estas aristas son verticales y tienen dirección hacia el norte (arriba).

- $A'_{hv}$ es el conjunto de aristas que unen un vértice de $V'_h$ con un vértice de $V'_v$ y cuyos segmentos inciden en la cuadrícula original. Estas aristas están inclinadas a 45° y tienen dirección hacia el noreste.

- $A'_{vh}$ es el conjunto de aristas que unen un vértice de $V'_v$ con un vértice de $V'_h$ y cuyos segmentos inciden en la cuadrícula original. Estas aristas están inclinadas a 45° y tienen dirección hacia el noreste.

### Implementación

Implementamos un programa que reduce instancias de TRPTW a instancias del problema *Max-Monotone-Tour* y posteriormente encuentra la trayectoria que captura el mayor número de segmentos acorde al algoritmo de 4-aproximación. La implementación fue realizada en el lenguaje de programación Java debido a que empleamos la biblioteca `algs4`, disponible para este lenguaje. Esta biblioteca fue desarrollada por los autores del texto [Sedgewick and Wayne, 2011], y provee estructuras de datos para representación de grafos, así como funciones para hacer operaciones sobre grafos. La función que encuentra el camino más largo en un grafo consume tiempo $E+V$, donde $E$ es el número de aristas y $V$ el número de vértices del grafo [Sedgewick and Wayne, 2011, p. 661]. El programa grafica las soluciones en la herramienta `Gnuplot`.

**Algorithm 5** 4-aproximación para TRPTW

**input**: Conjunto $X$ de $n$ ubicaciones sobre una línea.

**output**: Camino que captura el mayor número de localidades sobre la cuadrícula resultante.

**function** TRPTW-4APPROXIMATION

1. Representar las ubicaciones en dos dimensiones; agregar el eje $y$ para representar el tiempo.

2. Rotar el plano 45° en dirección de las manecillas del reloj.

3. Colocar nuevos ejes cartesianos para trabajar en el primer cuadrante del plano rotado.

4. Colocar cuadrícula sobre el primer cuadrante. Los lados de cada cuadrado de la cuadrícula miden $\frac{1}{\sqrt{2}}$.

5. Construir el grafo dirigido acíclico con pesos $G' = (V', A')$.

6. Asignar a cada arista de $A'$ un peso igual al número de segmentos inclinados que interseca en la cuadrícula original.

7. Resolver el problema *Max-Monotone-Tour*, con esto se obtiene el camino maximal monótono de $G'$.

**return** Camino maximal monótono en $G'$.

**end function**

La implementación de este algoritmo consume tiempo cuadrático debido a que, al igual que el algoritmo de 8-aproximación de la sección anterior, usa el método de programación dinámica de [Sedgewick and Wayne, 2011] para encontrar el camino más largo.

#### Pruebas realizadas

Exponemos a continuación soluciones generadas por nuestra implementación para dos instancias de TRPTW. La figura 3-4 es la solución para una instancia de 4 tareas donde la mejor ruta captura 3 tareas. El camino que completa el mayor número de tareas consiste en una colección de segmentos sobre la cuadrícula. La figura 3-5 es la solución para una instancia de 6 tareas. La mejor ruta completa sólo 3 tareas.

### 3.2.3. Comparación de algoritmos

El algoritmo de [Perez Perez et al., 2014] evita el problema de doble conteo y teóricamente su tasa de aproximación es mejor respecto al algoritmo de [Bar-Yehuda et al., 2005]. Para comprobar esto experimentalmente, generamos instancias de prueba aleatorias del problema TRPTW[3], y posteriormente resolvemas estas instancias mediante los dos algoritmos. A continuación describimos cómo generamos las instancias aleatorias.

[3]Para ver las propiedades de una instancia de TRPTW vea la sección 2.1.8

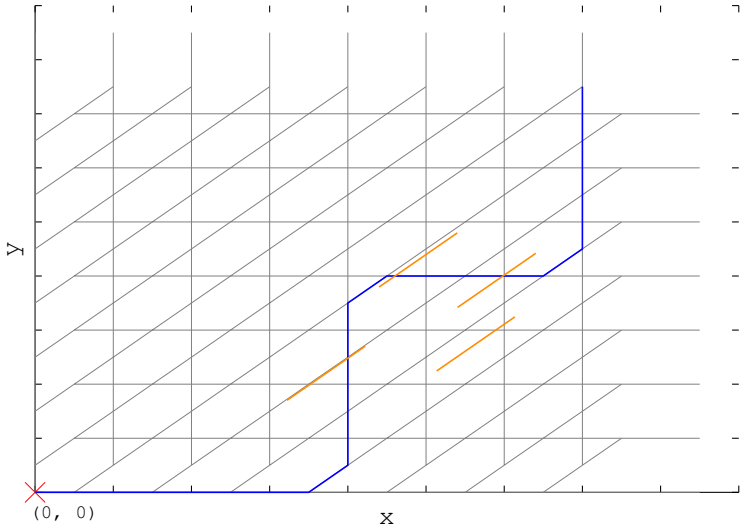


Figura 3-4: Solución obtenida mediante nuestra implementación del algoritmo de 4 aproximación para una instancia de 4 tareas. La mejor ruta completa a 3 tareas.

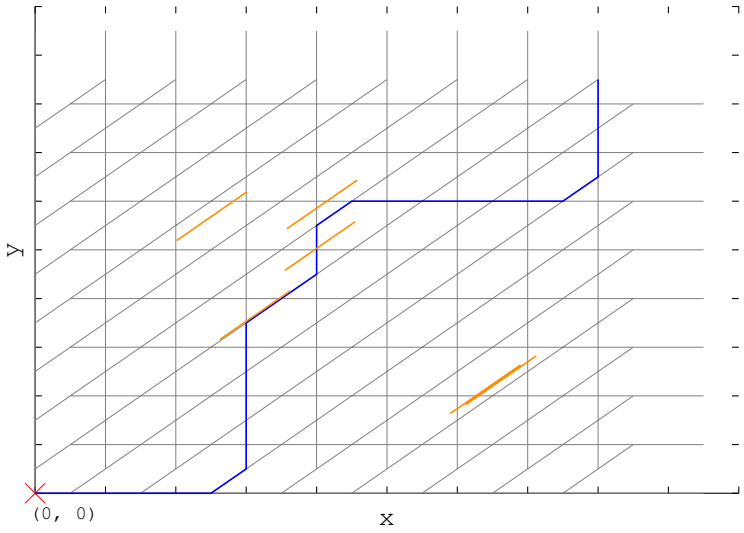


Figura 3-5: Solución obtenida mediante nuestra implementación del algoritmo de 4 aproximación para una instancia de 6 tareas. La mejor ruta completa 3 tareas. Aunque gráficamente la tercera tarea parece que no es intersecada, en realidad si es intersecada ya que la trayectoria en la imágen es el camino más largo en el DAG y las vueltas en diagonal representan vueltas completas en la cuadrícula original.

### Generación de pruebas aleatorias

Para generar instancias aleatorias de TRPTW empleamos el generador de números pseudo aleatorios que provee cada lenguaje de programación (función `srand(int`

`seed)` en C++). El generador se inicializa mediante el argumento enviado como parámetro (semilla). Para generar valores aleatorios, el generador se inicializa con un valor diferente en tiempo de ejecución, en este caso empleamos el valor devuelto por la función `time`, ésto genera valores suficientemente distintos para nuestras necesidades de aleatoriedad.

En cada instancia, generamos $n$ tareas donde la ubicación y tiempo de cada tarea tienen valores aleatorios (acorde al procedimiento descrito en el párrafo anterior). Los valores de las ubicaciones y el tiempo tienen valores entre 1.0 y 60.0.

### Comparación

Resolvimos cada instancia mediante los dos programas comparando el valor de la función objetivo (el número de tareas completadas). La tabla 3.1 muestra la comparación de las funciones objetivo calculadas por los dos algoritmos para varias instancias de TRPTW con diversos números de tareas. La primer columna indica el número de tareas $n$. La segunda y tercera columna son el resultado de la función objetivo del algoritmo de 8-aproximación y 4-aproximación, respectivamente. La cuarta columna de la tabla es la razón de variación de la función objetivo, esto es, el cociente de dividir el valor de la función objetivo del algoritmo de 8-aproximación y el valor de la función objetivo del algoritmo de 4-aproximación. Un valor igual o cercano a 1 indica que los resultados de ambos algoritmos son iguales, sin embargo, en ningún caso obtienen el mismo resultado, por lo tanto, se comprueba que el algoritmo de 4-aproximación obtiene mejores soluciones.

Para ejemplificar gráficamente la comparación de algoritmos, mostramos las soluciones de ambos algoritmos para una instancia de TRPTW con $5,000$ tareas. La figura 3-6 es una solución mediante el algoritmo de 8-aproximación donde el valor de la función objetivo es igual a 204 tareas completadas, mientras que la figura 3-7 es la solución para esa misma instancia mediante el algoritmo de 4-aproximación, la función objetivo en este caso es igual a 224, siendo éste el número correcto de tareas completadas, por tanto, la tasa de variación de ambos programas es igual a $\frac{204}{224} = 0.91$.

### Verificación de implementaciones

Las implementaciones del algoritmo de 8-aproximación y de 4-aproximación se verificaron empleando un programa de programación entera, ésto se detalla en la sección 4.6 del siguiente capítulo. El programa entero obtiene soluciones con la misma garantía de aproximación. Cada instancia (renglón) de la tabla 3.1 fue verificada con el programa entero de 8-aproximación y con el programa entero de 4-aproximación.

Tabla 3.1: Razón de variación entre el valor de la función objetivo del algoritmo de 8-aproximación y el algoritmo de 4-aproximación. Un valor igual o cercano a 1, en la cuarta columna, indica el grado de similitud de las soluciones.

| Núm. Tareas | A: Tareas completadas (8-aproximación) | B: Tareas completadas (4-aproximación) | C=A/B. Razón de variación |
|---|---|---|---|
| 50 | 14 | 16 | 0.875 |
| 100 | 18 | 19 | 0.947 |
| 150 | 22 | 22 | 1.000 |
| 200 | 26 | 30 | 0.867 |
| 250 | 28 | 32 | 0.875 |
| 300 | 32 | 35 | 0.914 |
| 350 | 41 | 43 | 0.953 |
| 400 | 36 | 43 | 0.837 |
| 450 | 42 | 47 | 0.894 |
| 500 | 46 | 51 | 0.902 |
| 550 | 46 | 50 | 0.920 |
| 600 | 56 | 59 | 0.949 |
| 650 | 51 | 56 | 0.911 |
| 700 | 51 | 61 | 0.836 |
| 750 | 58 | 64 | 0.906 |
| 800 | 58 | 66 | 0.879 |
| 850 | 67 | 77 | 0.870 |
| 900 | 71 | 76 | 0.934 |
| 950 | 73 | 83 | 0.880 |
| 1000 | 77 | 86 | 0.895 |
| 1500 | 78 | 90 | 0.867 |
| 2000 | 107 | 117 | 0.915 |
| 2500 | 127 | 144 | 0.882 |
| 3000 | 141 | 154 | 0.916 |
| 3500 | 160 | 178 | 0.899 |
| 4000 | 161 | 189 | 0.852 |
| 4500 | 187 | 215 | 0.870 |
| 5000 | 204 | 229 | 0.891 |
| 5500 | 202 | 228 | 0.886 |
| 6000 | 228 | 260 | 0.877 |
| 6500 | 236 | 266 | 0.887 |
| 7000 | 258 | 293 | 0.881 |
| 7500 | 274 | 312 | 0.878 |
| 8000 | 290 | 332 | 0.873 |
| 8500 | 287 | 345 | 0.832 |
| 9000 | 297 | 359 | 0.827 |
| 9500 | 317 | 372 | 0.852 |
| 10000 | 324 | 385 | 0.842 |

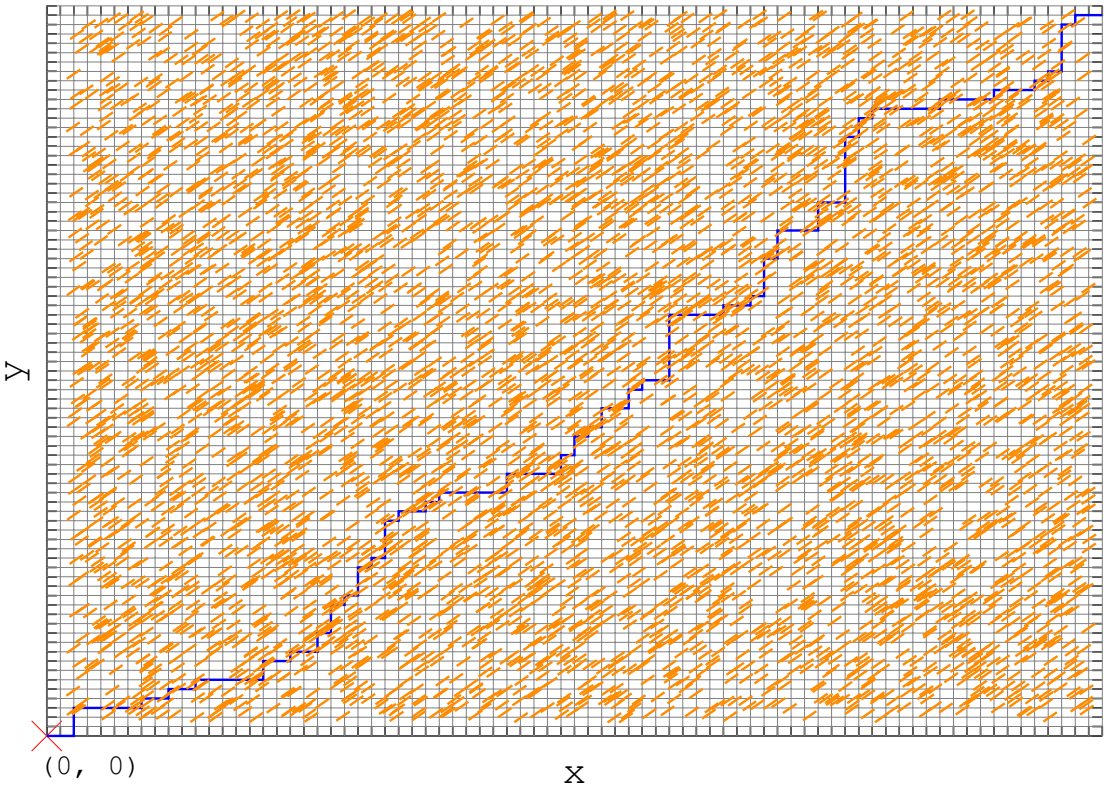


Figura 3-6: Solución obtenida mediante nuestra implementación del algoritmo de 8 aproximación para una instancia de 5000 tareas. La función objetivo es igual a 204 tareas.

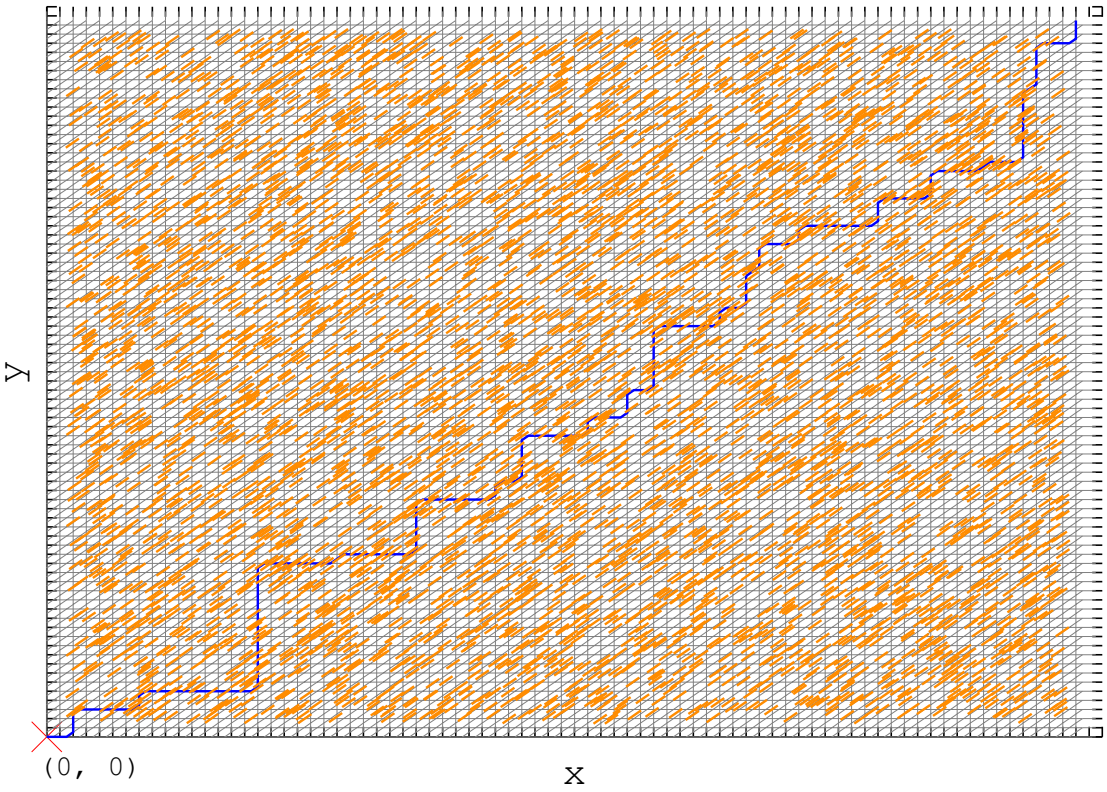


Figura 3-7: Solución obtenida mediante nuestra implementación del algoritmo de 4 aproximación para la misma instancia de 5000 tareas de la figura 3-6. La función objetivo es igual a 224 tareas.

# Capítulo 4

# Enfoque del problema del reparador viajero mediante programación entera

En este capítulo presentamos un enfoque mediante programación lineal entera para el problema del reparador viajero con ventanas de tiempo (TRPTW). Este enfoque resuelve instancias de TRPTW cuya función objetivo está dentro de un factor de 4 de la solución óptima. En la primera parte de este capítulo se detalla la construcción del modelo de programación entera (secciones 4.1 y 4.2). En la segunda parte (sección 4.3) presentamos un análisis empírico del tiempo necesario para resolver dicho modelo con la ayuda de un *solver* matemático, en este caso empleamos *Gurobi Optimizer*. Finalmente, mostramos cómo extender este enfoque para el caso en que las ventanas de tiempo son de longitud arbitraria.

## 4.1. Reducción al problema *Max-Monotone-Tour*

Comenzamos siguiendo el enfoque de [Bar-Yehuda et al., 2005] para reducir el problema original (TRPTW) al problema *Max-Monotone-Tour*. Esta reducción es un paso importante debido a que es más fácil trabajar con instancias del problema reducido que con instancias del problema original, como se verá a continuación.

En primer lugar representamos el conjunto de $n$ tareas en el plano bidimensional; el eje $x$ representa las ubicaciones de cada tarea, el eje $y$ representa el tiempo disponible de cada tarea (ventana de tiempo), como se ve en la figura 4-1.

Una trayectoria factible en este sistema de coordenadas, que captura a las ventanas de tiempo, es cualquier trayectoria monótonamente creciente respecto al eje vertical, con pendiente en cualquier momento en el intervalo $[-1, 1] = [\tan(135°), \tan(45°)]$ (ver figura 2-2 del capítulo 2). Las inclinaciones cercanas a 45° representan movimientos del reparador hacia la derecha a máxima velocidad, mientras que las inclinaciones cercanas a 135° representan movimientos hacia la izquierda, a máxima velocidad.

El plano se rota 45° respecto al orígen en dirección de las manecillas del reloj (figura 4-2). En el plano rotado, los segmentos inclinados que caen en el primer cuadrante son los únicos alcanzables por el reparador. Esto se debe a que el primer cuadrante

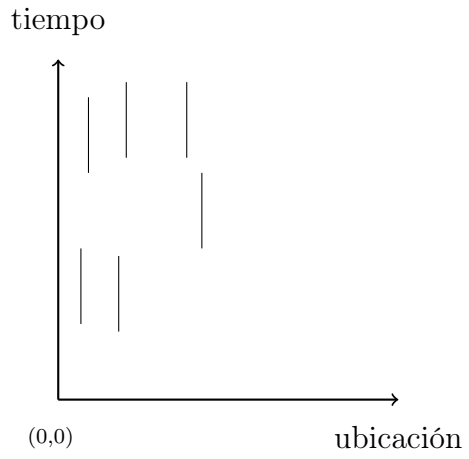


Figura 4-1: Instancia de TRPTW sobre un plano bidimensional, cada segmento vertical tiene longitud unitaria.

del plano rotado es en realidad el espacio de búsqueda de la trayectoria factible en el plano original (intervalo $[45°, 135°]$). Se puede ver que los segmentos verticales se convierten en segmentos inclinados con pendiente $1 = \tan(45°)$ respecto al nuevo eje $x$ (figura 4-3)[Bar-Yehuda et al., 2005].

Una solución factible en este nuevo plano es una trayectoria compuesta por líneas cuya inclinación está en el intervalo $[0°, 90°]$, es decir, una trayectoria que es monótonamente creciente con respecto a los ejes vertical y horizontal. Esto es una diferencia considerable respecto al plano original, en donde la trayectoria estaba caracterizada solamente con respecto al eje vertical [Bar-Yehuda et al., 2005]. Con base en lo anterior, la figura 4-4 muestra dos imágenes; la primera (izquierda) es el área de captura factible cuando la trayectoria es $y-$monótonamente creciente. La segunda imágen (derecha) es el área de captura factible cuando la trayectoria es $x(y)-$monótonamente creciente.

Esta rotación reduce el problema original, que era encontrar una trayectoria máxima en el plano original con segmentos verticales, al problema *Max-Monotone-Tour* con segmentos inclinados a 45°. El objetivo en *Max-Monotone-Tour* es encontrar una trayectoria $x(y)-$monótonamente creciente tal que ésta interseque el máximo número de segmentos. La ruta comienza en un punto $p_0$ (localizado en el orígen del plano cartesiano) y atraviesa el plano diagonalmente hasta un punto final predeterminado.

Obsérvese que una trayectoria factible en *Max-Monotone-Tour* debe ser monótonamente creciente respecto al eje $x$ y al eje $y$, en contraste con la trayectoria factible en el plano original, donde la trayectoria es monótonamente creciente sólo respecto al eje $y$.

Los movimientos del reparador hacia la derecha en el plano original, ahora, en *Max-Monotone-Tour* son equivalentes a desplazamientos paralelos al eje $x$, y los movimientos hacia la izquierda son ahora equivalentes a desplazamientos paralelos al eje $y$. Por lo tanto, la reducción al problema *Max-Monotone-Tour* resulta adecuada.

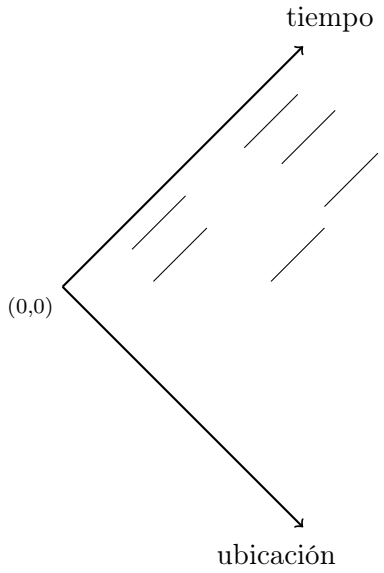


Figura 4-2: El plano se rota 45° en dirección de las manecillas del reloj respecto al orígen.

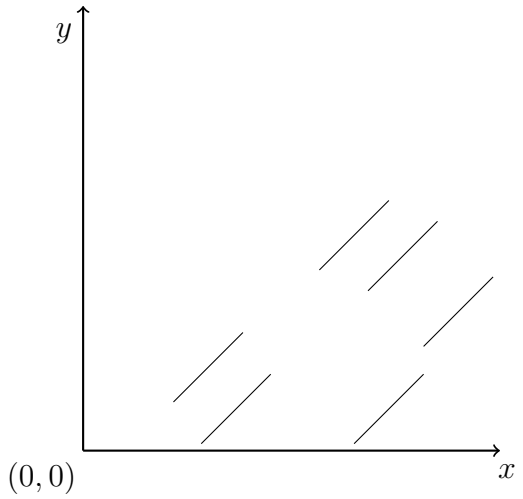


Figura 4-3: El plano rotado con el nuevo sistema de ejes coordenados. Los segmentos inclinados tienen pendiente $1 = \tan(45°)$.

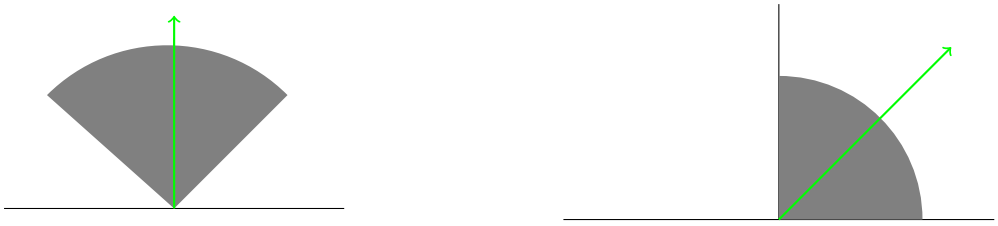

Figura 4-4: Area de captura de segmentos acorde al tipo de trayectoria. $y-$monótonamente creciente (izquierda). $x(y)-$monótonamente creciente (derecha).

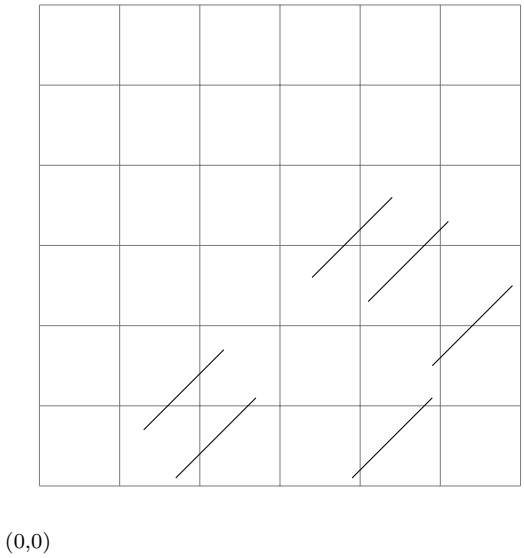


Figura 4-5: Una cuadrícula se coloca sobre el plano para cubrir los segmentos inclinados en su totalidad. Cada arista de los cuadrados mide $\frac{1}{\sqrt{2}}$.

Continuando con el enfoque propuesto en [Bar-Yehuda et al., 2005], se coloca una cuadrícula sobre el plano rotado teniendo cuidado que los extremos de los segmentos inclinados (de longitud unitaria) no toquen algún segmento de la cuadrícula. Esto se consigue haciendo que el tamaño de cada lado de los cuadrados en la cuadrícula sea igual a $\frac{1}{\sqrt{2}}$, por lo tanto su diagonal es igual a 1. El tamaño óptimo de la cuadrícula es $m \times m$ donde $m$ es el número de cuadrados que se requieren como mínimo, para cubrir en su totalidad a todos los segmentos inclinados (figura 4-5).

Hasta aquí se ha reducido el problema original a una instancia de *Max-Monotone-Tour*, en la siguiente sección vemos cómo construir el modelo de programación lineal entera.

## 4.2. Construcción del modelo de programación lineal entera

Explicamos a continuación cómo modelar el problema en un enfoque de programación lineal entera (ILP por sus siglas en inglés: *Integer Linear Programming*). Asignamos una variable $y_i$ a cada segmento inclinado en el plano, donde $1 \leq i \leq n$. Todas las variables tendrán valores binarios. La función objectivo es $\sum_{i=1}^{n} y_i$ donde $y_i$ es igual a 1 si el $i$-ésimo segmento es alcanzado, y 0 en otro caso. Las restricciones son modeladas a continuación. Sea $m$ el número de vértices en cada lado de la cuadrícula, hay $m^2$ vértices. En cada vértice, el reparador tiene dos movimientos legales: se mueve hacia el norte (segmento vertical), o bien, hacia el este (segmento horizontal). Asignamos 1 a los segmentos atrapados por el reparador y 0 a los demás segmentos.

Los movimientos permitidos por el reparador pueden ser formulados a través de un sistema de ecuaciones lineales. Aquí surgen casos independientes dados por movimientos en el inicio de la trayectoria, en el final de la trayectoria y sobre vértices internos de la cuadrícula. Además, los movimientos en las orillas de la cuadrídula dependen de sus posiciones.

Sea $x_{ij0}$ la variable que denota al segmento horizontal con dirección hacia el este proveniente del vértice $(i, j)$. Similarmente, sea $x_{ij1}$ la variable que denota el movimiento vertical hacia el norte. El reparador comienza su ruta en el vértice $(0, 0)$ y escoge solamente uno de los aristas $x_{000}$ ó $x_{001}$, es decir, la ecuación (4.1) se cumple:

$$x_{000} + x_{001} = 1. \tag{4.1}$$

En el último vértice de la cuadrícula, el cual está en las coordenadas $(m, m)$, se cumple la ecuación (4.2). Esta ecuación indica que el reparador puede proceder solamente de uno de los aristas adyacentes, $x_{m-1,m,0}$ ó $x_{m,m-1,1}$:

$$x_{m-1,m,0} + x_{m,m-1,1} = 1. \tag{4.2}$$

Para ilustrar el uso de estas ecuaciones, veamos un ejemplo gráfico. En la figura 4-6 cada arista es etiquetada con su correspondiente variable. Las Figuras 4-7.a y 4-7.b ilustran el cumplimiento de las ecuaciones (4.1) y (4.2) respectivamente.

En cada vértice interno, la ecuación (4.3) debe cumplirse, $\forall (i, j) \in [1, m-1]^2$:

$$x_{i-1,j,0} + x_{i,j-1,1} = x_{ij0} + x_{ij1}, \tag{4.3}$$

Esta ecuación mantiene la idea principal de que en cada vértice entran y salen dos segmentos, lo cual se cumple claramente porque el reparador escoge aristas de su trayectoria que comienza en $(0, 0)$ y termina en $(m, m)$ (ver figura 4-8).

De la misma forma, en los vértices no internos de la cuadrícula se deben cumplir condiciones similares. Sin embargo, en algunos de estos vértices sólo uno de las dos aristas entrantes (o salientes) está presente. Los bordes de la cuadrícula que contienen tales vértices deben ser identificados, de esta forma, las restricciones (4.4) y (4.5)

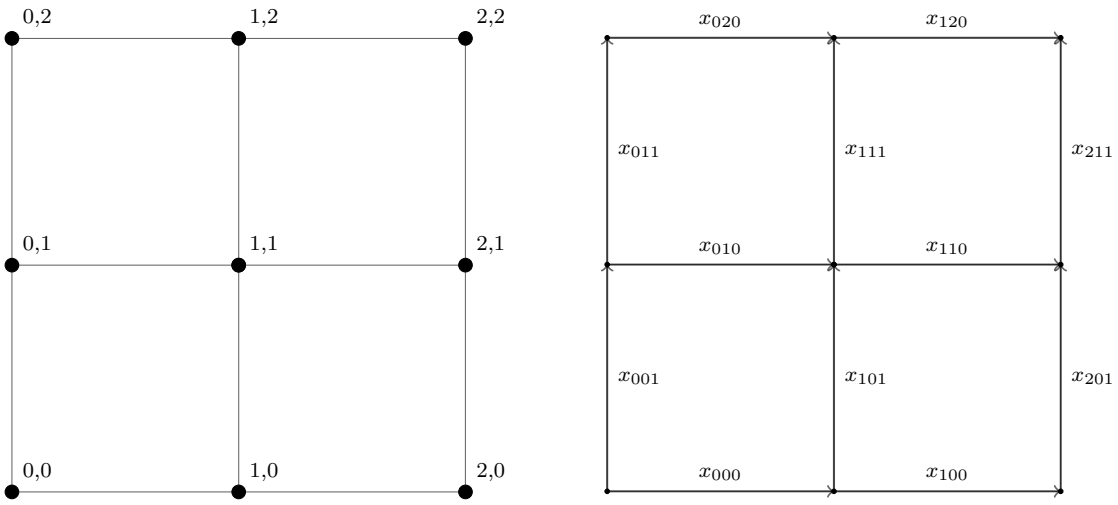


Figura 4-6: Una cuadrícula de $3 \times 3$ con sus vértices etiquetados, aparece en la izquierda. La misma cuadrícula con sus aristas etiquetados, se muestra a la derecha.

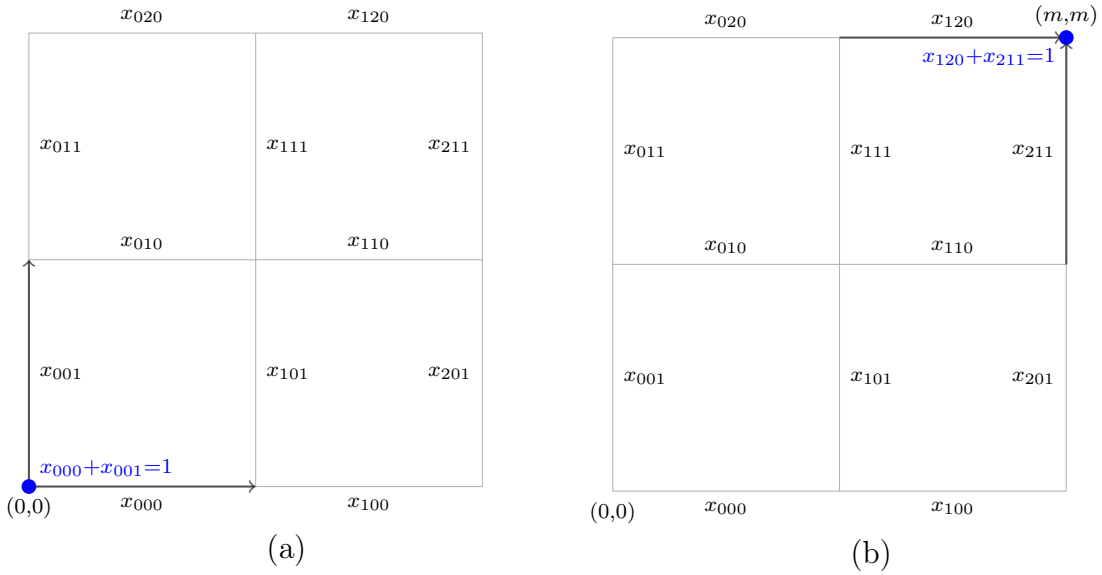


Figura 4-7: Movimientos permitidos del reparador. (a) En el vértice inicial. (b) En el último vértice.

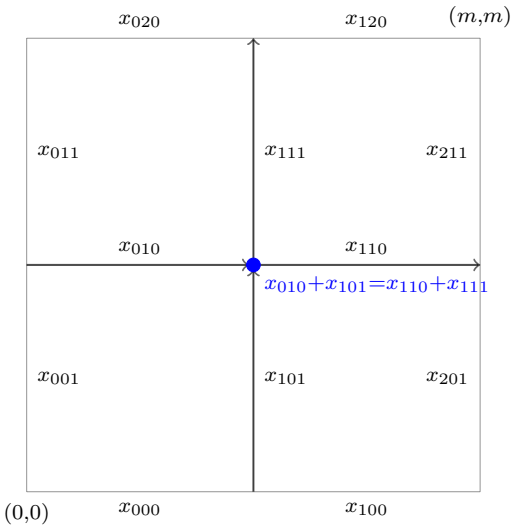


Figura 4-8: Ecuación lineal que modela los movimientos permitidos en vértices internos de la cuadrícula.

deben ser satisfechas en las esquinas restantes (ver figura 4-9.a).

$$x_{0,m-1,1} = x_{0m0}, \tag{4.4}$$
$$x_{m-100} = x_{m01}. \tag{4.5}$$

Las restricciones (4.6), (4.7), (4.8) y (4.9) deben cumplirse en los vértices restantes de las orillas de la cuadrícula (ver figura 4-9.b).

$$x_{0,j-1,1} = x_{0j0} + x_{ij1}, \quad 1 \leq j < m, \tag{4.6}$$
$$x_{m,j-1,1} + x_{m-1,j,0} = x_{mj1}, \quad 1 \leq j < m, \tag{4.7}$$
$$x_{i-100} = x_{i00} + x_{i01}, \quad 1 \leq i < m, \tag{4.8}$$
$$x_{i-1,m,0} + x_{i,m-1,1} = x_{im0}, \quad 1 \leq i < m. \tag{4.9}$$

### 4.2.1. Modelación de captura de ventanas con longitud unitaria

Hasta aquí hemos modelado los movimientos permitidos del reparador sobre la cuadrícula. Ahora nos enfocaremos de las restricciones relacionadas a la captura de los segmentos inclinados en el caso particular donde éstos tienen longitud unitaria. Sean $x_{i_0j_00}$ y $x_{i_1j_11}$ los segmentos horizontales y verticales (de la cuadrícula) intersecados por el $i$-ésimo segmento inclinado, $1 \leq i \leq n$. Entonces, las Desigualdades (4.10)

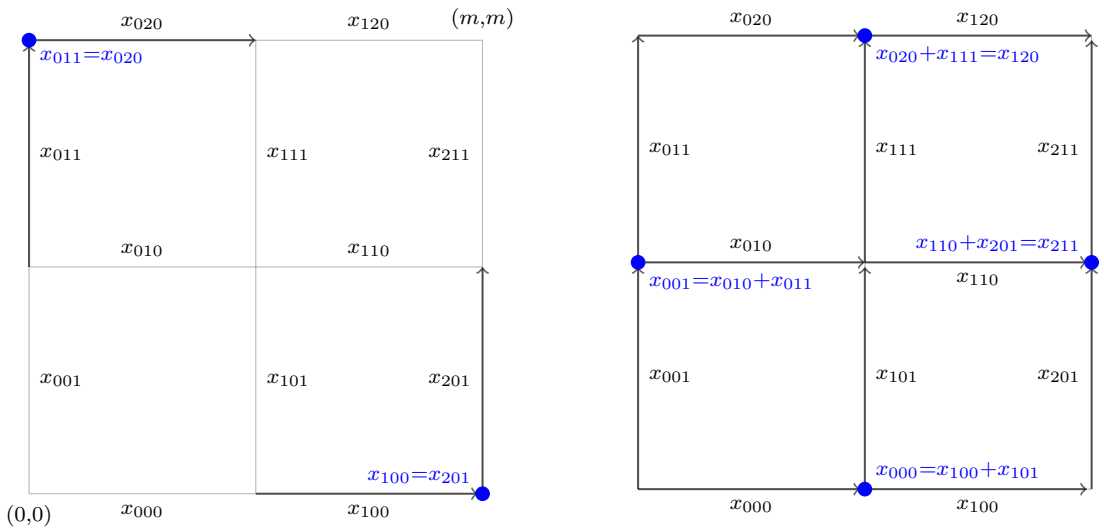


Figura 4-9: Restricciones para movimientos del reparador en vértices no-internos de la cuadrícula. (a). En las esquinas. (b). En las orillas

y (4.11) deben ser agregadas al conjunto de restricciones.

$$y_i \leq x_{i_0 j_0 0} + x_{i_1 j_1 1}, \tag{4.10}$$

$$y_i \leq 1, \tag{4.11}$$

La desigualdad (4.10) indica que cuando el reparador alcanza al menos uno de los segmentos de la cuadrícula intersecados por el $i$-ésimo segmento inclinado, entonces $y_i$ es capturado y puede tener un valor mayor que cero. En otro caso el único valor posible para $y_i$ es cero. La desigualdad (4.11) indica que la captura de un segmento es contado a lo más una vez a pesar de que la captura fue hecha por un segmento de cuadrícula horizontal o vertical. De esta forma, evitamos el *problema de doble conteo*, el cual es una limitante del algoritmo de 8-aproximación presentado por [Bar-Yehuda et al., 2005]. Un ejemplo de las desigualdades para dos segmentos inclinados, se muestra en la figura 4-10.

Las restricciones anteriores constituyen la reducción de programación lineal entera. Dado que las aristas de la cuadrícula y los segmentos inclinados han sido modelados como variables en la reducción de ILP, la resolución de esta reducción indicará cuáles aristas de la cuadrícula deben ser visitadas para obtener la mejor ruta.

## 4.3. Implementación y análisis

Implementamos un método que transforma instancias de TRPTW al modelo de programación lineal entera, acorde al procedimiento descrito en la sección 4.1. El

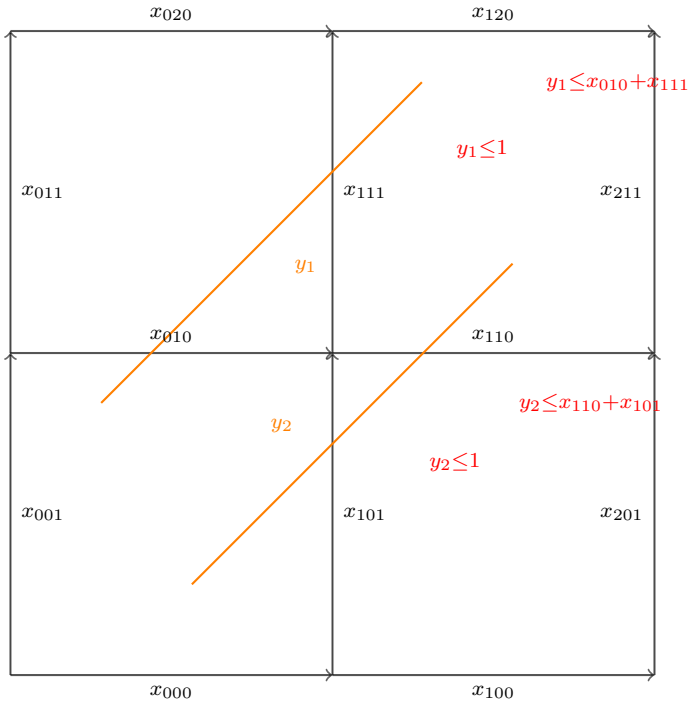


Figura 4-10: Desigualdades lineales que modelan la captura de los segmentos inclinados.

programa fue construido en `C++`. Una vez que se obtiene el modelo de programación entera, resulta necesario resolver dicho modelo con la ayuda de un *solver* matemático, en este caso empleamos *Gurobi Optimizer* (ver [Gurobi Optimization, 2015]). Cuando *Gurobi Optimizer* ha encontrado la mejor solución, el programa grafica la solución empleando `Gnuplot`.

En la figura 4-11 aparece una solución para la instancia de TRPTW tratada en la figura 4-2. La mejor trayectoria del reparador que maximiza el número de tareas completadas es la ruta $x_{000}, x_{101}, x_{110}, x_{210}, x_{310}, x_{410}, x_{511}, x_{521}, x_{531}, x_{540}, x_{641}, x_{651}$. En esta solución, el reparador completa un total de 5 tareas.

La figura 4-12 es una solución para una instancia de 20 segmentos inclinados donde el número de segmentos atrapados es 8. La figura 4-13 es una solución para una instancia de 1000 segmentos inclinados donde el número de segmentos atrapados es 70. Ambas son soluciones generadas por nuestro programa.

### 4.3.1. Crecimiento del modelo de programación entera

El crecimiento del modelo propuesto, en términos del número de variables y número de restricciones (desigualdades y ecuaciones), es diréctamente proporcional al número de vértices por lado de la cuadrícula y del número de segmentos inclinados de una instancia dada. A continuación presentamos dos fórmulas (Lema 1 y 2) que indican el número de variables y restricciones necesarias para modelar mediante programación

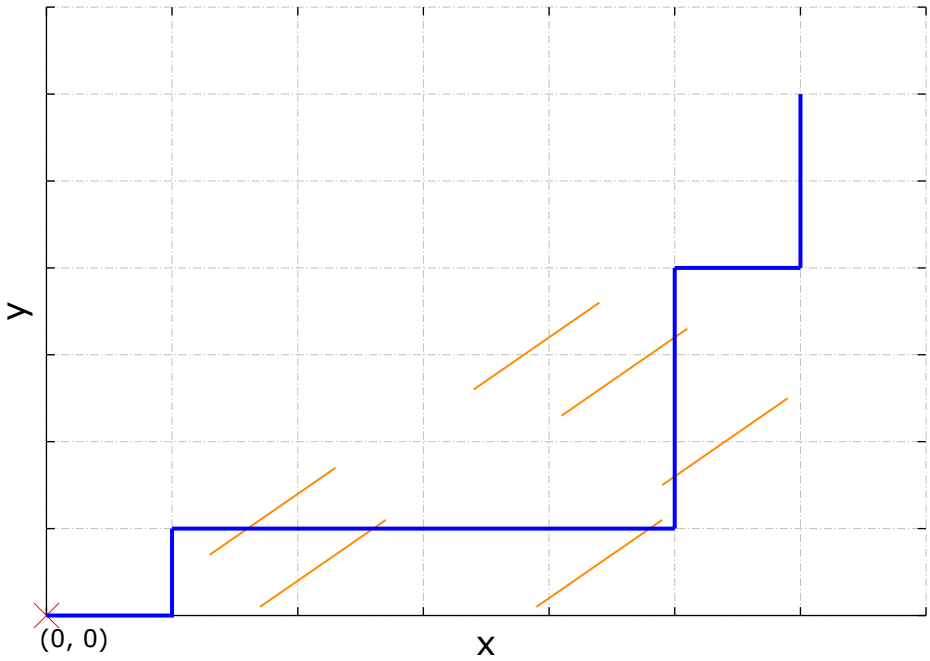


Figura 4-11: Solución para la instancia rotada de TRPTW presentada en la figura 4-2.

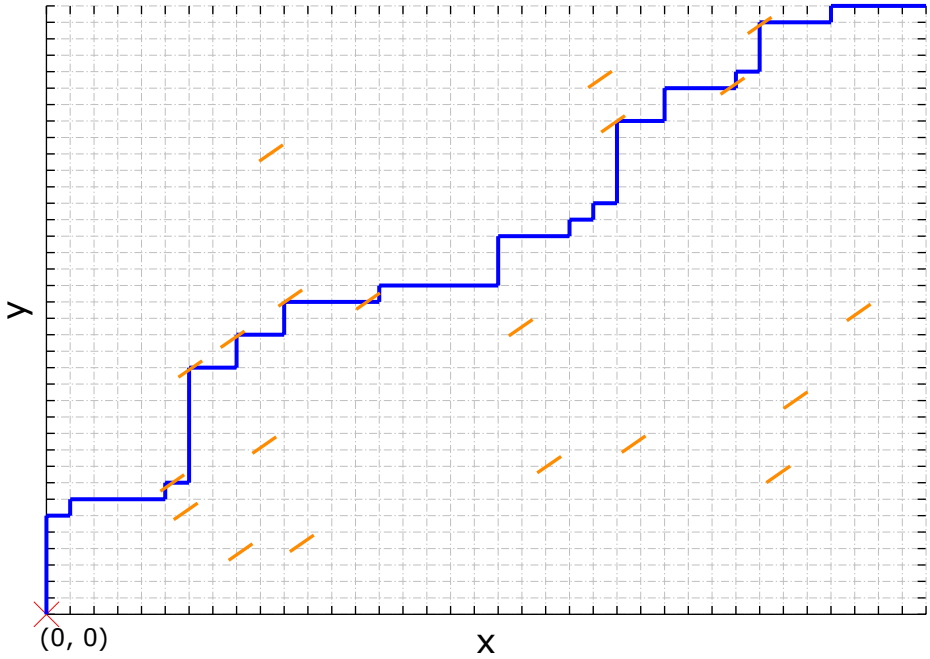


Figura 4-12: Solución para una instancia de 20 segmentos inclinados, 8 segmentos son alcanzados.

lineal entera cualquier instancia de TRPTW.

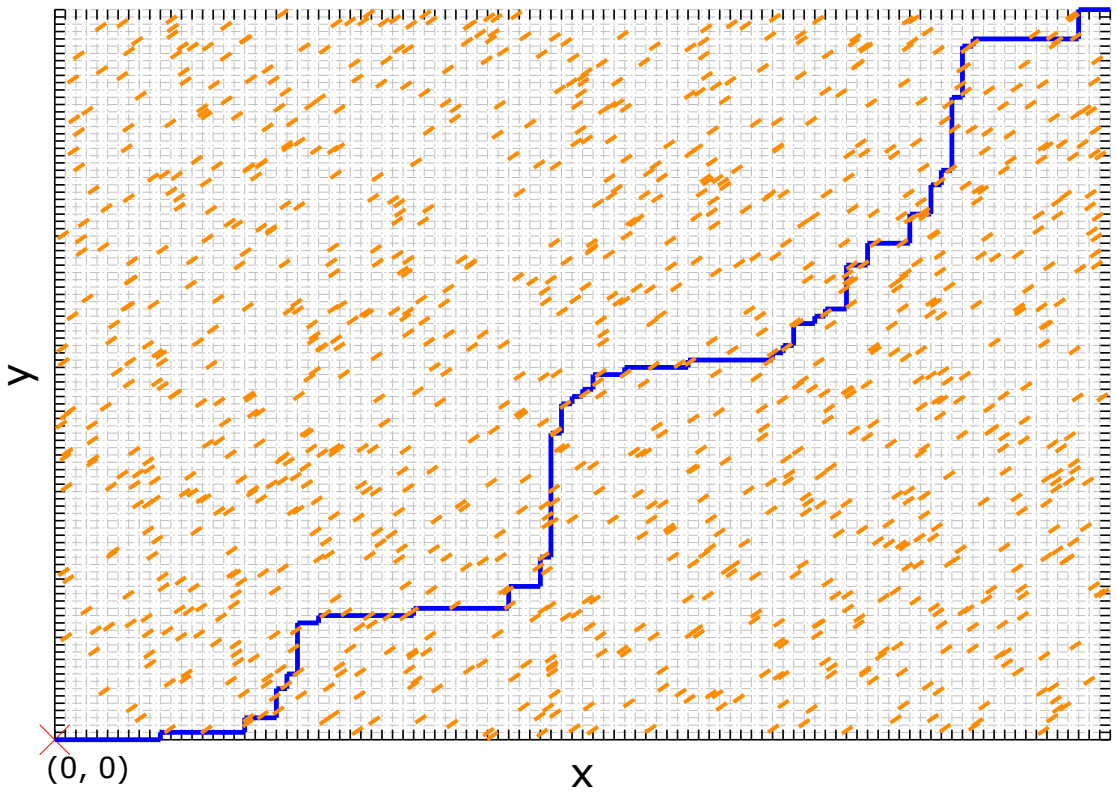


Figura 4-13: Solución para una instancia de 1000 segmentos inclinados. 70 segmentos son alcanzados.

**Proposición 1** *Sea $n$ el número de segmentos inclinados (tareas) de una instancia de TRPTW, y sea $m$ el número de vértices por lado de la cuadrícula que cubre todos los segmentos inclinados. El número de variables del modelo de programación lineal entera está dado por $v = 2m(m-1) + n$.*

**Proposición 2** *Sea $n$ el número de segmentos inclinados (tareas) de una instancia de TRPTW, y sea $m$ el número de vértices por lado de la cuadrícula que cubre todos los segmentos inclinados. El número de restricciones del modelo de programación lineal entera está dado por $r = m^2 + 2n$.*

### 4.3.2. Análisis

Para una instancia de TRPTW, sea $n$ el número de tareas (segmentos inclinados), sea $m$ el número de vértices por lado de la cuadrícula necesaria para cubrir todos los segmentos inclinados. La transformación desde una instancia de TRPTW a la reducción del modelo de ILP se lleva a cabo en tiempo $O(nm)$. El tiempo requerido para resolver el modelo de ILP, es el tiempo que consume *Gurobi Optimizer* en resolver tal modelo, por lo tanto, no podemos hacer un análisis asintótico del tiempo requerido

para resolver el modelo de ILP. Para dar una estimación experimental del tiempo que *Gurobi Optimizer* tarda en resolver tal reducción, hemos generado 4569 instancias de TRPTW para probar nuestro enfoque con diferentes valores de $n$ y $m$. Para valores de $(n, m)$ con $1 \leq n \leq 500$ y $1 \leq m \leq 80$ generamos cinco instancias aleatorias de TRPTW. Estas instancias fueron ejecutadas con nuestro programa y calculamos el tiempo promedio de ejecución. Los valores del conjunto de datos fueron ajustados a varias funciones polinomiales de grado $1, 2, \cdots, 8$ para determinar cuál grado de función es el adecuado. Las funciones de grado 2 y 3 tuvieron un mejor ajuste. Se hizo análisis de regresión de ambos ajustes y se obtuvo el factor *Adjusted R-Squared Statistics* con un valor de 0.928 para el ajuste cuadrático, y 0.943 para el ajuste cúbico.

La figura 4-14 es una gráfica del tiempo ocupado por el conjunto de datos (puntos en color azul) y la curva del ajuste cúbico polinomial. El polinomio de ajuste es:

$$\begin{aligned} p_3(n,m) \;=\; & 0.319 \times 10^{-1} \\ & -0.441 \times 10^{-3} n + 0.721 \times 10^{-2} m \\ & +0.325 \times 10^{-5} n^2 - 0.280 \times 10^{-4} nm \\ & -0.362 \times 10^{-3} m^2 \\ & -0.231 \times 10^{-9} n^3 - 0.145 \times 10^{-6} n^2 m \\ & +0.304 \times 10^{-5} nm^2 + 0.622 \times 10^{-5} m^3. \end{aligned}$$

Ciertamente, un polinomio cúbico es un mejor ajuste para el conjunto de valores de tiempo de las instancias de prueba comparado con un ajuste cuadrático. El polinomio $p_3(n, m)$ ofrece una buena interpolación de los valores de tiempo recabados, pero evidentemente está lejos de ofrecer alguna extrapolación. Sin embargo, el valor absoluto pequeño de los valores de los coeficientes de los monomios cúbicos en el ajuste polinomial puede sugerir que hay un crecimiento polinomial cuadrático en el tiempo de ejecución experimental empleado en resolver el modelo de ILP para este conjunto de instancias.

Las soluciones calculadas por este enfoque están dentro de un factor de 4 de la solución óptima, esto se debe a que nuestro enfoque emplea la rotación del plano con la cuadrícula sobrepuesta tal como se hace en [Bar-Yehuda et al., 2005] en donde se demuestra que si existe un algoritmo que encuentre un camino maximal en esta cuadrícula, este algoritmo será una 8-aproximación (ver [Bar-Yehuda et al., 2005]). La tasa de aproximación se reduce a la mitad gracias a que nuestro enfoque evita el problema de doble conteo. En síntesis, el algoritmo desarrollado es una 4-aproximación, debido esencialmente a la eliminación del doble conteo.

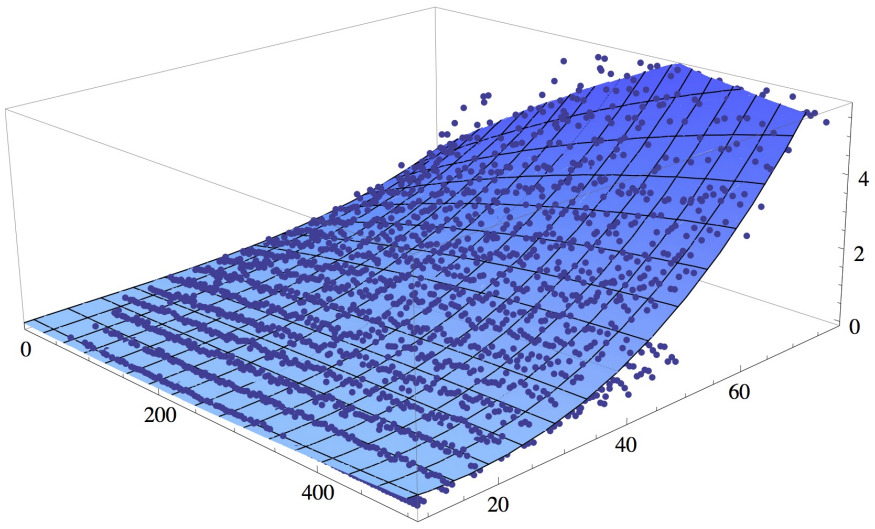


Figura 4-14: Tiempos de ejecución del conjunto de pruebas (puntos en azul) y la curva del ajuste polinomial de grado 3. El lado izquierdo de la superficie es el número de segmentos, el lado derecho de la superficie es el número de vértices por lado de la cuadrícula. La altura es el tiempo en segundos.

## 4.4. Extensión del enfoque para ventanas con longitud arbitraria

Es fácil modificar nuestro método propuesto para soportar ventanas de tiempo con longitud arbitraria. Para ello, regresamos a la construcción del modelo de ILP y modificaremos las desigualdades relacionadas a la captura de segmentos inclinados. Dado que ahora cada segmento inclinado puede tener una longitud mayor que 1, entonces éste podrá intersecar más de dos segmentos de la cuadrícula, en consecuencia, la primera desigualdad que modela dicha captura contendrá la suma de términos asociados a los segmentos de cuadrícula que intersecan al segmento inclinado. La segunda desigualdad, la cual restringe el valor de $y_i$ a 1, se mantiene sin cambio alguno. Un ejemplo de esta reducción aparece en la figura 4-15. La figura 4-16 es una solución generada con nuestro programa para una instancia de TRPTW con ventanas de tiempo de longitud arbitraria.

Cuando se trabaja la variante de TRPTW con ventanas de tiempo de longitud arbitraria, el valor de las ventanas está en el intervalo $[1, 2)$ debido a que es un valor recomendado en la literatura (véase [Bar-Yehuda et al., 2005]) obtenido tras particionar la longitud original. La figura 4-17 es una solución de nuestro enfoque para una instancia de TRPTW con 1000 tareas cuyas ventanas de tiempo tienen longitud en el intervalo $[1, 2)$. La cuadrícula resultante para cubrir los segmentos inclinados tiene 110 vértices por lado y la función objetivo indica que 93 tareas son completadas.

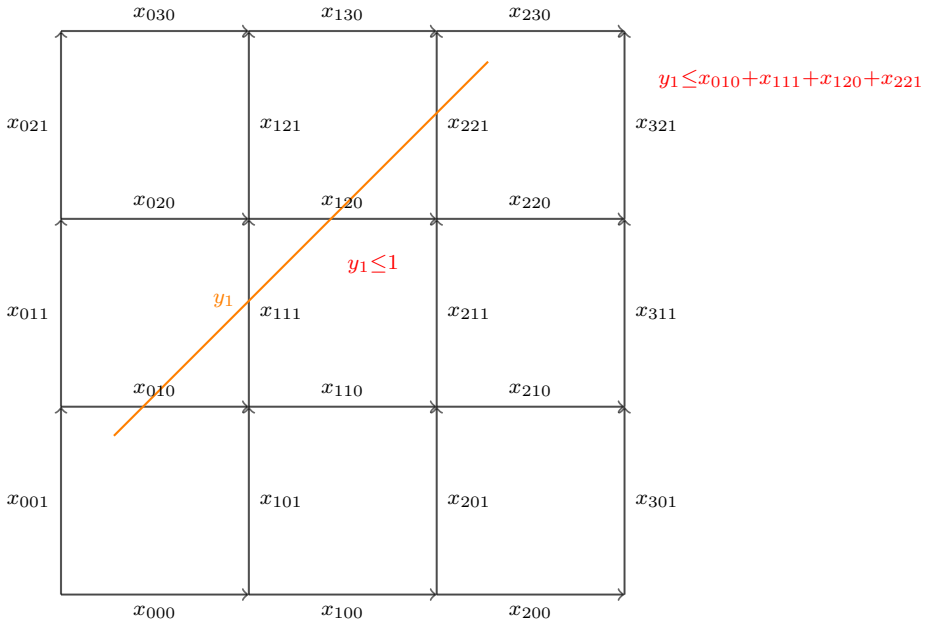


Figura 4-15: Desigualdades lineales (en rojo) que modelan la intersección entre los segmentos inclinados de longitud igual a 2 y los segmentos de la cuadrícula que son intersecados.

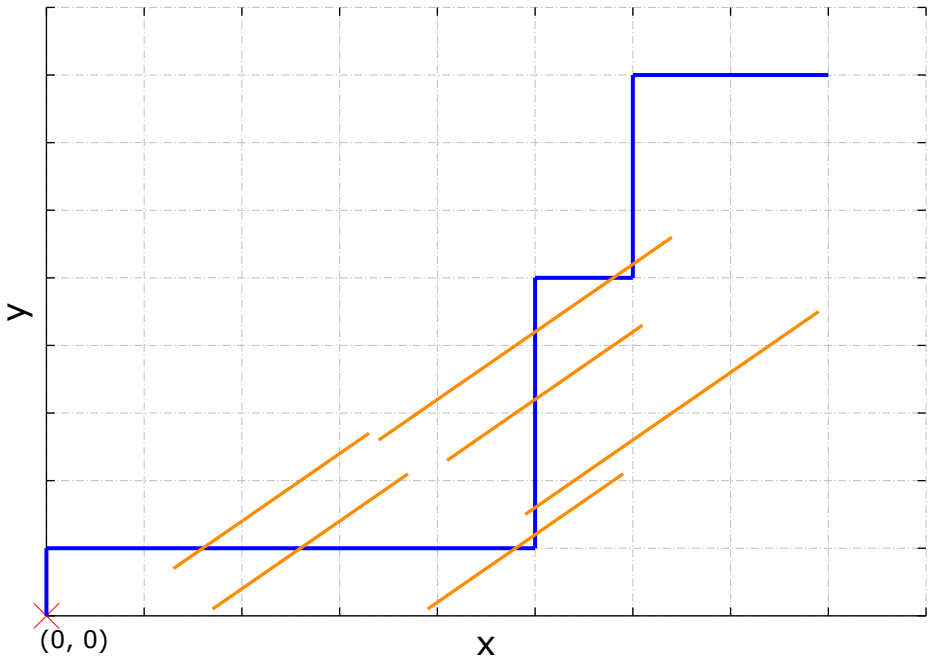


Figura 4-16: Segmentos inclinados de la figura 4-2 ahora tienen longitud arbitraria. La mejor trayectoria captura a todos estos.

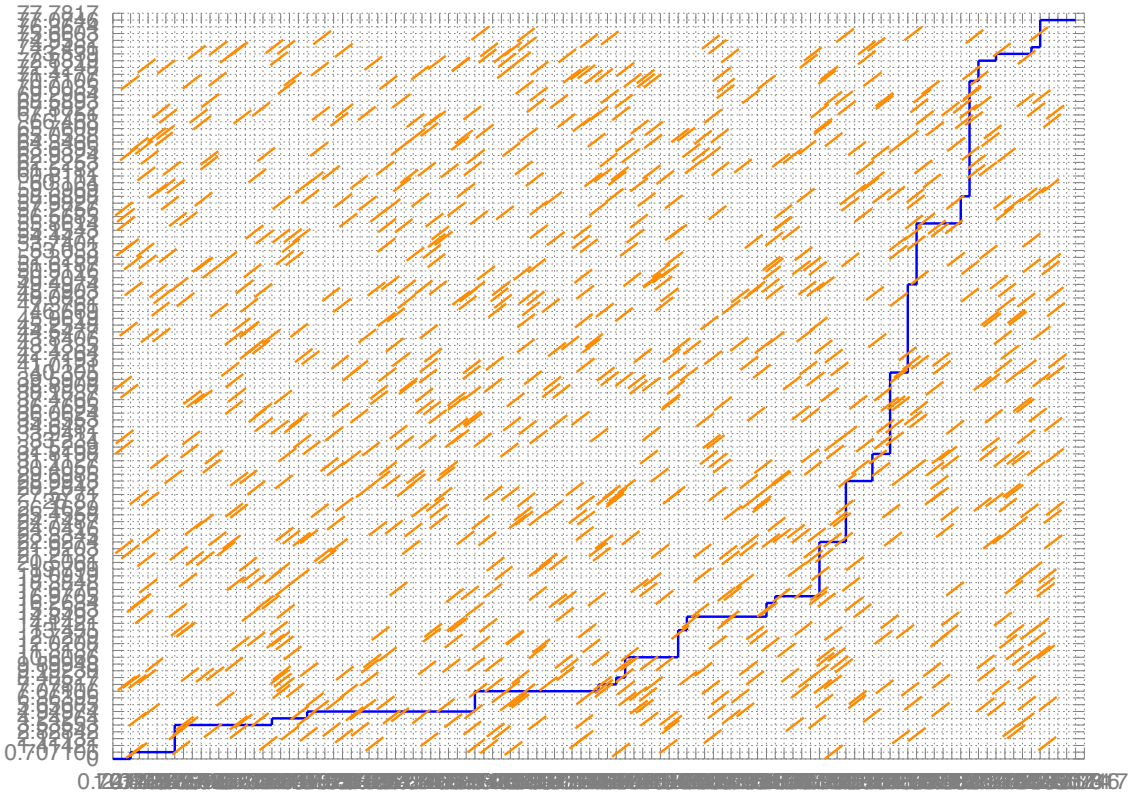

Figura 4-17: Solución para una instancia de TRPTW con 1000 tareas cuyas ventanas de tiempo tienen longitud arbitraria en el intervalo $[1, 2)$. La mejor trayectoria completa 93 tareas.

## 4.5. Código fuente y ambiente de prueba

Las mediciones del conjunto de datos, los reportes de análisis de regresión y el código fuente del enfoque de ILP están disponibles en Gitlab. En la sección 5.2 se ofrecen instrucciones para descargar el material.

Las pruebas fueron realizadas en una computadora con procesador Intel i5 dual-core con velocidad de procesador de 1.3 GHz en cada core. La memoria RAM del sistema es 4 GB.

## 4.6. Uso del modelo entero para verificación de algoritmos conocidos

Como el enfoque de programación entera que proponemos encuentra soluciones con una garantía de 4-aproximación, ya que evita el problema de doble conteo, tal implementación puede usarse como un verificador de la implementación del algoritmo de 4-aproximación de [Perez Perez et al., 2014] que presentamos en 3.2.2. Por lo tanto, cualquier instancia de TRPTW que sea resuelta mediante la implementación del algoritmo de 4-aproximación y mediante la implementación de programación entera tendrá soluciones iguales, es decir, el valor de la función objetivo en ambos programas será el mismo.

Análogamente, el programa entero también puede usarse como verificador para la implementación del algoritmo de 8-aproximación de [Bar-Yehuda et al., 2005] que presentamos en 3.2.1, sin embargo, requiere una ligera modificación debido a que por sí solo genera soluciones con garantía de 4-aproximación. Para que el programa entero genere soluciones con garantía de 8-aproximación es necesario hacer una modificación a la función objetivo del modelo; la función a maximizar actualmente está en términos de las variables que representan los segmentos inclinados $y_1, \ldots, y_n$, tales variables deben eliminarse del modelo y en su lugar se escribe la función objetivo en términos de las variables de aristas de la cuadrícula que intersecan a los segmentos inclinados. Como las variables $y_1, \ldots, y_n$ ya no están en el modelo, también se eliminan las desigualdades que restringían que sus valores sean menores o iguales que 1 ya que estas desigualdades evitaban el problema de doble conteo. De este modo, el modelo ya no evita el problema del doble conteo y obtiene soluciones con garantía de 8-aproximación al igual que el algoritmo de [Bar-Yehuda et al., 2005].

# Capítulo 5

# Conclusiones

Este capítulo contiene tres secciones. En la primer sección se presentan nuestras conclusiones tras la realización de este trabajo, lo que el lector encuentra en este trabajo, así como cuáles fueron las principales contribuciones y los resultados positivos o negativos de la tesis. En la segunda sección presentamos un vínculo al sitio electrónico donde el lector puede obtener el código fuente de los programas desarrollados en esta tesis. En la tercera sección se discute el posible trabajo a futuro.

## 5.1. Relevancia del trabajo

En contraste con otros estudios teóricos que han estudiado TRPTW, este estudio sigue un enfoque más experimental que teórico. A lo largo de este documento se ha presentado una breve introducción y estudio del problema del reparador viajero con ventanas de tiempo. Se presentaron algunos problemas de calendarización de tareas que resultan necesarios para abordar el problema, tales como el problema de calendarización de tareas unitarias con penalizaciones, problema de calendarización de intervalos de tiempo y el problema de calendarización de intervalos con peso.

Nos enfocamos en el caso especial de TRPTW donde las ventanas de tiempo son de longitud unitaria y estudiamos dos algoritmos de la literatura que abordan este caso. Implementamos estos algoritmos y comprobamos experimentalmente que las soluciones obtenidas por el algoritmo de 4-aproximación propuesto por [Perez Perez et al., 2014] respecto al algoritmo de 8-aproximación de [Bar-Yehuda et al., 2005], son mejores gracias a que el algoritmo de [Perez Perez et al., 2014] evita el problema de dobleo conteo con la ayuda de un nuevo digrafo acíclico ponderado que modela más vértices y restricciones del problema.

En el capítulo 4, hemos propuesto e implementado un enfoque de solución mediante programación lineal entera para solucionar instancias de TRPTW donde la función es maximizar el número de tareas que son completadas. Este enfoque trabaja para el caso especial en que las ventanas tienen longitud unitaria. Las soluciones que calcula nuestro enfoque tienen un factor de aproximación de 4 respecto a la solución óptima. Esto se debe a que la función objetivo de nuestro enfoque de ILP coincide con el

valor de la función objetivo del enfoque de programación dinámica en [Bar-Yehuda et al., 2005]. Además, nuestro programa es fácilmente extendible al caso en que las ventanas de tiempo son de longitud arbitraria, tal como lo hemos comprobado experimentalmente al resolver instancias de longitud arbitrarias de TRPTW. Debido a que es necesario emplear un *solver* matemático para solucionar el modelo de programación entera, resulta imposible dar una estimación de la complejidad computacional del programa en términos asintóticos, sin embargo, hemos recabado mediciones experimentales de un conjunto de instancias de prueba y estimamos que la función que describe el comportamiento del consumo de tiempo, para ese conjunto de datos, podría ser cuadrática o cúbica. Sin embargo, para instancias grandes el comportamiento podría ser exponencial debido al alto costo computacional que implica resolver el modelo de ILP.

En resumen, además de encontrar en este trabajo un breve estudio del problema del reparador viajero, el lector encuentra un conjunto de implementaciones de algoritmos de aproximación de TRPTW incluyendo nuestro enfoque de programación entera. Este conjunto de programas están disponibles para su descarga.

## 5.2. Plataforma computacional

Las implementaciones de algoritmos y resultados presentados en esta tesis conforman una plataforma computacional que resuelve instancias de TRPTW. El material se ha colocado en el siguiente sitio electrónico para ser descargado públicamente:

`https://computacion.cs.cinvestav.mx/~omiguel/plataformaTRP.html`

En esa ubicación es posible descargar los siguientes archivos:

- *Tarball* de archivos de código fuente.
- Manual de instalación.
- Manual de usuario.

## 5.3. Trabajo futuro

En el capítulo 3 se han comparado dos algoritmos principales de la literatura para el caso de TRPTW en que las ventanas de tiempo son unitarias. Esta comparación consitió en medir las soluciones de la función objetivo de ambos programas, sin embargo, los autores del algoritmo de 4-aproximación nos han comentado que sería un aporte considerable comprobar experimentalmente la tasa de aproximación de ese algoritmo. Para comprobar ésto, será necesario implementar un algoritmo exhaustivo que encuentre soluciones óptimas en instancias de TRPTW, sin embargo, como se trata de un problema computacionalmente difícil, el algoritmo exhaustivo ocupará

tiempo exponencial, así que se podrían resolver instancias con un número pequeño de tareas, que se traduce en una prueba con poca información, por tanto, aún queda abierto esta comprobación experimental. El cálculo de la complejidad computacional para ciertos casos de TRPTW (tablas 2.1 y 2.2) son aún problemas abiertos.

En el programa de programación entera, hemos analizado experimentalmente un conjunto de datos de tamaño limitado debido al alto costo computacional de la programación lineal entera ya que la reducción de ILP genera un modelo con un número alto de restricciones y variables que resulta adecuado resolver sólo con la ayuda de un resolvedor matemático, sin embargo, para instancias de TRPTW demasiado grandes, ello puede consumir tiempo considerable, incluso exponencial. Por lo tanto, otras técnicas como programación dinámica, algoritmos de grafos o incluso algoritmos aleatorios pueden ser útiles para proponer nuevas estrategias para TRPTW que tiendan a mejorar la tasa de aproximación de algoritmos actuales en tiempo polinomial.

# Bibliografía


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

[Arora and Karakostas, 1999] S. Arora and G. Karakostas. Approximation schemes for minimum latency problems. In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, pages 688–693, New York, NY, USA, 1999. ACM. ISBN 1-58113-067-8. doi: 10.1145/301250.301432. URL `http://doi.acm.org/10.1145/301250.301432`.

[Ausiello et al., 1999] G. Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi, and V. Kann. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1999. ISBN 3540654313.

[Bar-Yehuda et al., 2005] R. Bar-Yehuda, G. Even, and S. Shahar. On approximating a geometric prize-collecting traveling salesman problem with time windows. *J. Algorithms*, 55(1):76–92, 2005.

[Chao et al., 1996] I.-M. Chao, B. L. Golden, and E. A. Wasil. The team orienteering problem. *European Journal of Operational Research*, 88(3):464 – 474, 1996. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/0377-2217(94)00289-4. URL `http://www.sciencedirect.com/science/article/pii/0377221794002894`.

[Cormen et al., 2009] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. ISBN 0262033844, 9780262033848.

[Dewilde et al., 2010] T. Dewilde, D. Cattrysse, S. Coene, F. C. R. Spieksma, and P. Vansteenwegen. Heuristics for the traveling repairman problem with profits. In *ATMOS 2010 - 10th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, Liverpool, United Kingdom, September 6-10, 2010*, pages 34–44, 2010. doi: 10.4230/OASIcs.ATMOS.2010.34. URL `http://dx.doi.org/10.4230/OASIcs.ATMOS.2010.34`.

[Diestel, 2012] R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012. ISBN 978-3-642-14278-9.

[Fakcharoenphol et al., 2003] J. Fakcharoenphol, C. Harrelson, and S. Rao. The k-traveling repairman problem. In *Proceedings of the 14th annual ACM-SIAM symposium on Discrete algorithms*, pages 655–664, 2003.

[Frederickson and Wittman, 2007] G. N. Frederickson and B. Wittman. Approximation algorithms for the traveling repairman and speeding deliveryman problems with unit-time windows. In *Proceedings of the 10th International Workshop on Approximation and the 11th International Workshop on Randomization, and Combinatorial Optimization. Algorithms and Techniques*, APPROX '07/RANDOM '07, pages 119–133, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74207-4.

[Garey and Johnson, 1990] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. ISBN 0716710455.

[Gurobi Optimization, 2015] I. Gurobi Optimization. Gurobi optimizer reference manual, 2015. URL `http://www.gurobi.com`.

[Kleinberg and Tardos, 2005] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005. ISBN 0321295358.

[Papadopoulos, 2005] A. Papadopoulos. *Metric Spaces, Convexity and Nonpositive Curvature*. European Mathematical Society, 2005.

[Perez Perez et al., 2014] S. L. Perez Perez, L. E. Urban Rivero, R. Lopez Bracho, and F. J. Zaragoza Martinez. A fast 4-approximation algorithm for the traveling repairman problem on a line. In *Electrical Engineering, Computing Science and Automatic Control (CCE), 2014 11th International Conference on*, pages 1–4, Sept 2014. doi: 10.1109/ICEEE.2014.6978272.

[Sedgewick and Wayne, 2011] R. Sedgewick and K. Wayne. *Algorithms*. Pearson Education, 2011. ISBN 9780132762564. URL `https://books.google.fi/books?id=idUdqdDXqnAC`.

[Sitters, 2013] R. Sitters. Polynomial time approximation schemes for the traveling repairman and other minimum latency problems. *CoRR*, abs/1307.4289, 2013. URL `http://arxiv.org/abs/1307.4289`.

[Tsitsiklis, 1992] J. N. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22:263–282, 1992.

[Vansteenwegen et al., 2011] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1 – 10, 2011. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2010.03.045. URL `http://www.sciencedirect.com/science/article/pii/S0377221710002973`.

[Wu et al., 2004] B. Y. Wu, Z.-N. Huang, and F.-J. Zhan. Exact algorithms for the minimum latency problem. *Inf. Process. Lett.*, 92(6):303–309, Dec. 2004. ISSN 0020-0190. doi: 10.1016/j.ipl.2004.09.009. URL `http://dx.doi.org/10.1016/j.ipl.2004.09.009`.