UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

FACULTAD DE INGENIERÍA

MINIMIZACIÓN DE COSTOS Y RETRASOS EN LA PROGRAMACIÓN DE LA SECUENCIA DE DESPEGUES Y ATERRIZAJES EN UNA PISTA

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA

INGENIERÍA EN SISTEMAS - INVESTIGACIÓN DE OPERACIONES

P R E S E N T A:

ING. MARIANA BÁRCENA DIEZ

TUTOR:

DRA. MAYRA ELIZONDO CORTÉS

2011

JURADO ASIGNADO

Presidente: Dr. Aceves García Ricardo

Secretario: Dra. Flores de la Mota Idalia

Vocal: Dra. Elizondo Cortés Mayra

1er Suplente: Dr. Ramírez Rodriguez Javier

2do Suplente: Dra. Larraga Ramirez María Elena

Lugar o lugares donde se realizó la tesis:

POSGRADO DE INGENIERÍA

TUTOR DE TESIS

DRA. MAYRA ELIZONDO CORTÉS

_____

FIRMA

MINIMIZACIÓN DE COSTOS Y RETRASOS EN LA PROGRAMACIÓN DE LA SECUENCIA DE DESPEGUES Y ATERRIZAJES EN UNA PISTA

Mariana Bárcena Diez

Resumen

Los aterrizajes y despegues de los aviones en las pistas de los aeropuertos se han convertido en un recurso crítico en el sistema de tráfico aéreo. Los retrasos en estos vuelos tienen un gran impacto en las operaciones y costos de las aerolíneas. Es por esto que una planeación más estructurada, tanto en las llegadas como en las salidas de los vuelos, ha llegado a tomar mucha importancia.

En este trabajo se explicará primeramente un modelo matemático para la minimización de costos y retrasos describiendo la función objetivo y sus restricciones. Dado que el problema es combinatorio y la complejidad computacional inherente que esto conlleva, es necesario utilizar un método de aproximación como lo son los métodos heurísticos. En este caso se utiliza la metaheurística *Búsqueda Tabú*. Para poder tener una solución de arranque para utilizar este método y ya que no se cuenta con información real suficiente, se programó una simulación del sistema de despegues y aterrizajes en una pista a lo largo de un día. Posteriormente se presentan los resultados y las disminuciones obtenidas tanto en retrasos como en costos.

Palabras clave:

- Aterrizajes
- Despegues
- Vuelos
- Aerolíneas
- Pistas
- Retrasos
- Función de costos
- Programación de actividades *(Scheduling)*
- Métodos exactos
- Métodos heurísticos
- Búsqueda Tabú
- Complejidad computacional

ÍNDICE

ÍNDICE DE FIGURAS

FIGURA

ÍNDICE DE TABLAS

TABLAS

INTRODUCCIÓN

En la actualidad, nos encontramos inmersos en una realidad que se mueve con mucha rapidez. Todos los servicios y productos necesitan estar en el momento adecuado, en el lugar adecuado y al costo adecuado. Es por eso que todo gira alrededor del esfuerzo por minimizar costos, tiempos, retrasos, etc., para así poder satisfacer a los clientes, dar un mejor servicio y ser más competitivos. En el transporte aéreo no es la excepción. En particular se aplica con los vuelos de las aerolíneas. Éstas pretenden proporcionar a los clientes su servicio de la mejor manera posible y esto incluye ofrecer buenos precios mediante la reducción de sus costos, y que los vuelos salgan y lleguen a tiempo mediante la minimización de los retrasos. Sin embargo, la asignación de vuelos a las pistas es controlada por el aeropuerto en el que se encuentren y no por las aerolíneas. Éste se encarga de la asignación de pistas, cuando existe más de una, y de la secuencia en cada una de ellas tanto de despegues como de aterrizajes.

Es frecuente que los vuelos tengan retrasos, ya sea de unos cuantos minutos o incluso de varias horas. Esto genera un gran descontento en los clientes y una penalización monetaria bastante considerable para las aerolíneas. Sobre todo, estos costos se incrementan mucho en los vuelos que tienen conexiones porque ocasionan el retraso de esa o todas las conexiones que ese vuelo en específico precede.

Es por eso que es de suma importancia tener en cuenta que la secuencia de aterrizajes y despegues en la pista debería tener una mejor organización que la actual. Por el momento funciona con un sistema de cola llamado FCFS (first come first served), más comúnmente denominado FIFO (first in first out).

Se propone modificar la secuencia de vuelos (aterrizajes y despegues) de tal manera que se minimicen los retrasos y los costos asociados a ellos. El análisis se hará de un solo día, de tal manera que esta reorganización de secuencia se deberá hacer diariamente.

Scheduling (o programación de actividades) es un proceso de toma de decisión, el cual se utiliza frecuentemente en el área de manufactura y servicio. Se enfrenta a problemas de asignación de tareas a recursos dentro de un tiempo determinado. La meta de esta programación de actividades puede ser minimizar uno o más objetivos (Baker, 2009).

Los recursos y tareas (o trabajos) pueden tomar diferentes formas dependiendo de la empresa en cuestión. Los recursos pueden ser máquinas en una fábrica, pistas en un aeropuerto, personal en una constructora, etc. Los trabajos pueden ser operaciones en un trabajo de producción, despegues y aterrizajes en un aeropuerto, estaciones en un proyecto de construcción, etc. Todo trabajo tiene un cierto nivel de prioridad, un tiempo en el que puede empezarse y un tiempo en el que debe terminarse.

A continuación se presentan dos diagramas para ejemplificar la fase de *scheduling* en la toma de decisiones tanto en servicios como en manufactura.

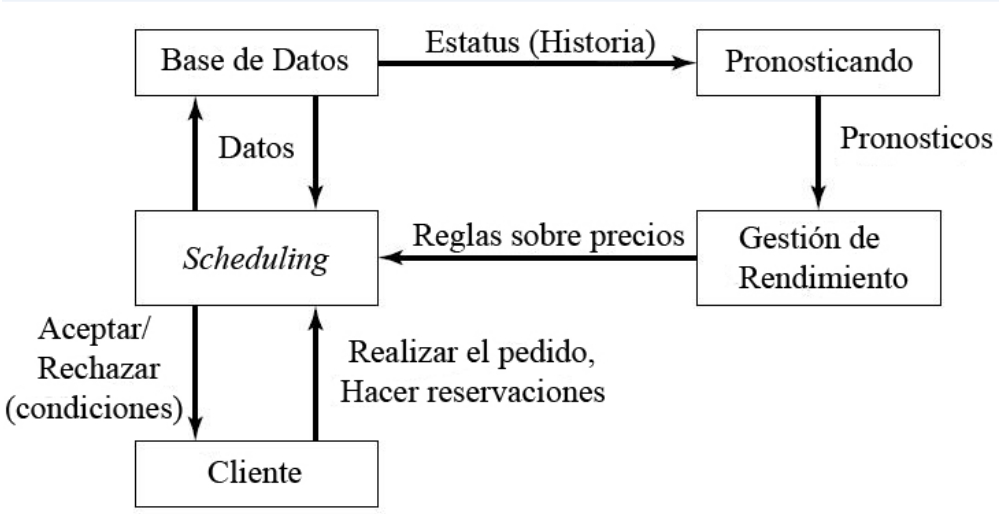


Figura 1.1 **Diagrama de flujo de un sistema de servicio** (basado en Pinedo, 2008)

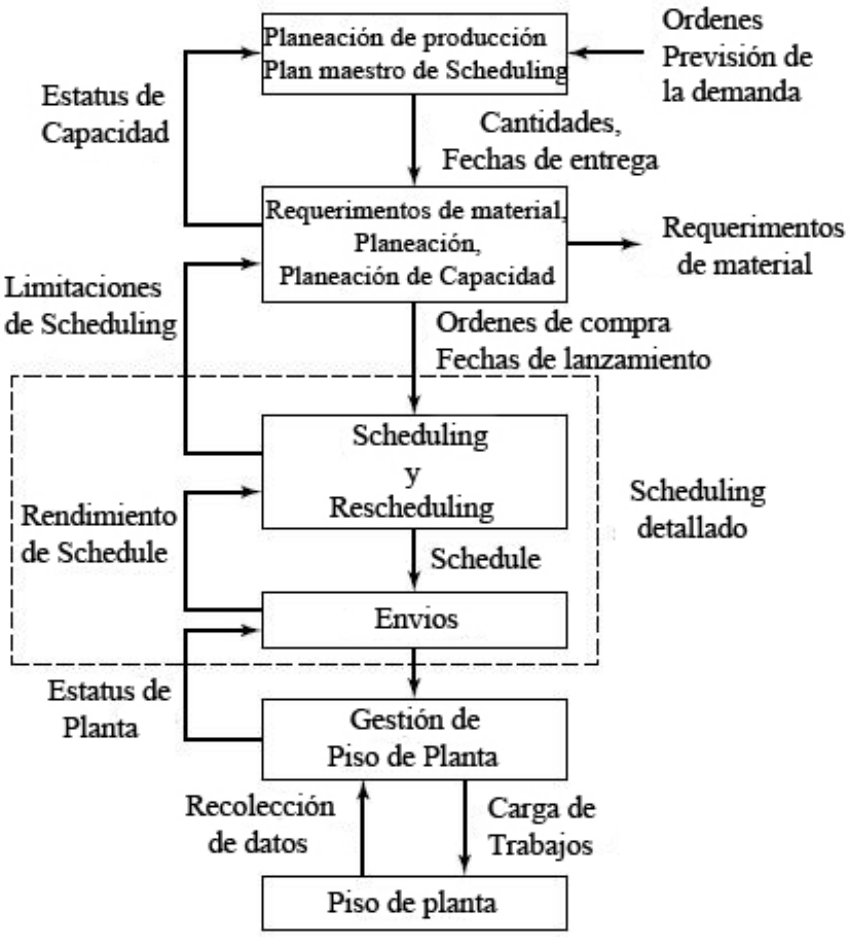


Figura 1.2 **Diagrama de flujo de un sistema de manufactura** (basado en Pinedo, 2008)

La programación de los vuelos, como se acaba de mencionar, se puede analizar desde esta perspectiva de *scheduling*. Los despegues y aterrizajes serán los trabajos, y las pistas serán los recursos. El análisis de la reprogramación de la secuencia se realizará en una sola pista.

Dado que los problemas de *scheduling* pertenecen a la clase de los NP-duros, por ser combinatorios, es recomendable el uso de un método heurístico para su resolución (Pinedo, 2008). En esta investigación se utilizará un método llamado *Búsqueda Tabú (Tabu Search)* considerado dentro de los algoritmos de mejora. Estos métodos, a pesar de no garantizar encontrar el valor óptimo, proporcionan una muy buena solución (véase apéndice A).

El objetivo de esta investigación será, por lo tanto, minimizar los costos y los retrasos en la secuencia de despegues y aterrizajes de vuelos comerciales en una pista en el Aeropuerto Internacional de la Ciudad de México (AICM) mediante un modelo matemático de Programación Entera Mixta. Dada la complejidad computacional del problema, se realizó la programación del método heurístico *Búsqueda Tabú* basado en el modelo matemático. Debido a que dicho método heurístico requiere una solución inicial factible o de arranque, fue necesario realizar una simulación del sistema actual. Dicha simulación se utilizó también para poder comparar el resultado obtenido mediante la metaheurística y así poder demostrar una disminución en costos y retrasos.

La estructura de esta investigación será la siguiente: en el primer segundo se presentará el estado del arte. En el tercer capítulo el marco teórico. En el cuarto capítulo se presenta el modelo matemático para la minimización de costos y retrasos para tomar como base en la programación del método heurístico. En el quinto capítulo se describe la simulación del sistema a tratar, la cual se utilizará como solución de arranque en el método heurístico. En el sexto capítulo, la explicación del método heurístico a utilizar, su solución y la comparación con los resultados obtenidos en la simulación. Finalmente se presentan las conclusiones de la investigación, referencias, apéndices y anexos.

Esta es sólo una propuesta que podría ayudar a la mejor planeación de la operación de cualquier aeropuerto, o en éste caso, del AICM.

1.1. Problemática de la situación actual

En las últimas décadas, el tráfico aéreo ha crecido de manera muy considerable y en los últimos años más. Del año 2002 al año 2007 hubo un incremento del 23% en el número de vuelos en todo el sistema aéreo (Soomer, 2007). Se han realizado mejoras en aumentar la capacidad del tráfico en ruta, sin embargo, como resultado se tiene que el poder llevar estos vuelos en ruta al aeropuerto, o hacer que los vuelos que están esperando en el aeropuerto puedan despegar/aterrizar, se ha vuelto un cuello de botella (Soomer, 2007). Se puede decir, que los aterrizajes y despegues están compitiendo y esperando su turno para utilizar la pista. Resulta mucho más económico, aunque parezca contradictorio, mantener un avión

en vuelo que tenerlo parado en el aeropuerto, ya que el uso de un espacio en un aeropuerto es muy caro.

Adicionalmente, hoy en día se requieren inversiones de capital enormes para poder operar una aerolínea. Los costos operacionales son muy altos, sobre todo, por los precios del combustible. Las medidas de seguridad han aumentado considerablemente desde los ataques en EEUU del 11 de septiembre del 2001 en todo el mundo, provocando más aumentos en costos. Adicionalmente, existe una gran competencia por bajar los costos de los boletos. Todo esto ha obligado a las aerolíneas a trabajar muy eficientemente.

Por otra parte, los retrasos de los vuelos provocan también muchos problemas en las conexiones que pueden llegar a perder los pasajeros de dicho vuelo. En estos casos, los pasajeros deben ser reasignados a otros vuelos e incluso, algunas veces, tienen que ser compensados por la aerolínea responsable. Además de ser complicado reasignar a todos los pasajeros que sufrieron retrasos en su vuelo, este percance implica pérdidas monetarias considerable, ya que debido a la falta de vuelos disponibles es necesario, en ocasiones, pagar hospedaje a los usuarios.

En la figura 1.3. se puede ver cómo se afecta la demanda de un vuelo si éste se llega a atrasar o adelantar. Además este retraso provoca que todos los demás vuelos que le preceden, realizados con el mismo avión, también sufran retrasos y, asociado a ellos, un cambio en la demanda de cada uno de éstos.

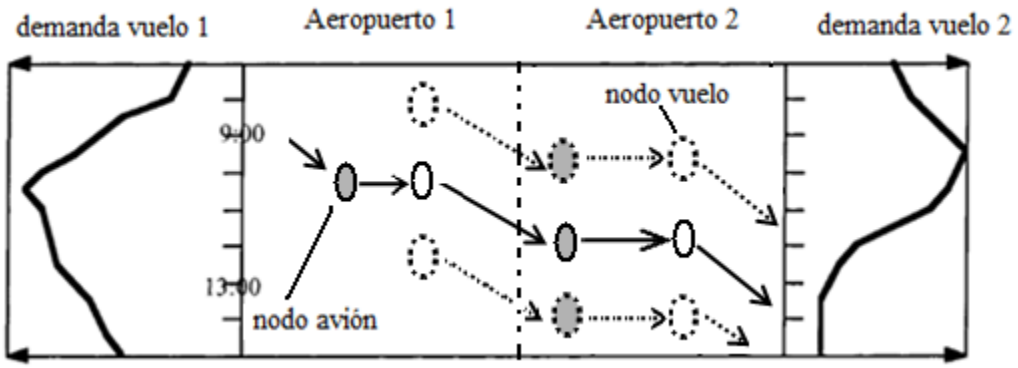


Figura 1.3 **Reasignación de vuelos** (Tomado de Cao, 2000)

Adicionalmente a todo lo anterior, las pistas de aterrizaje son un recurso limitado en los aeropuertos. La capacidad de las pistas está sujeta a muchos cambios como el clima y

cambios en las condiciones de visibilidad. Sin embargo, la demanda es predecible debido a que se basa en la programación de los vuelos. Durante las horas pico, la capacidad de una pista de aterrizaje/despegue se acerca a su límite e, incluso en algunos casos, se rebasa. Esto sucede generalmente en aeropuertos internacionales, como el Aeropuerto Internacional de la Ciudad de México AICM.

Actualmente, es difícil que exista un plan de aterrizajes/despegues entre 24 horas y 1 hora antes de ellos. En consecuencia, el flujo de los vuelos que entran al alcance del radar del aeropuerto no es muy ordenado. La secuencia de aterrizajes de los vuelos se asigna, por lo tanto, en forma similar a la secuencia de registro en el rango de alcance del radar. La torre de control tiene la habilidad de hacer pequeños cambios por razones de seguridad y eficiencia, pero estos cambios son mínimos. Una ventaja de la planeación de aterrizajes es la posibilidad de reasignar retrasos de los vuelos en el aire. En consecuencia, estos retrasos pueden ser absorbidos mientras están formados para aterrizar. Es dos veces más caro tener un avión parado en el aeropuerto que tenerlo en el aire. (Inniss and Ball 2004).

El control de tráfico aéreo es responsable de guiar los vuelos de manera segura, equitativa, y eficiente. Los vuelos que se acercan a los aeropuertos son guiados por los controladores de proximidad 30 minutos antes del aterrizaje (cuando entran al rango del radar). Desde ese momento, el controlador debe crear un flujo con las debidas separaciones entre aviones enfilados a la pista. Esta separación depende de las categorías de peso de los vuelos (ligero, mediano y pesado). Un avión muy pesado que vuela detrás de uno ligero necesita menos separación que uno ligero vuela detrás de uno pesado (Innis and Ball 2004).

La mayoría de los aeropuertos internacionales trabajan constantemente excediendo su capacidad, lo que provoca retrasos más frecuentes. Es importante saber que cada retraso en un aterrizaje o despegue, representa un costo para la aerolínea en cuestión.

Con base en los datos proporcionados por las aerolíneas en México, los ingresos, en promedio, de un vuelo comercial se aproxima a $25,000 USD, de los cuales un 40% es de carga ($15,000 USD) y un 60% es de pasajeros ($10,000 USD), aproximadamente. De esos ingresos cerca del 25% se paga en uso de aeropuerto ($6,250 USD).

A continuación se muestran algunas cifras de las operaciones del AICM para ejemplificar la importancia de este aeropuerto internacional, uno de los más importantes en América Latina.

En la tabla 1.4 podemos ver la cantidad de pasajeros, tanto de llegadas como de salidas, en cada una de las terminales, que hubo en el año 2009.

PASAJEROS POR TERMINAL

	Terminal 1					Terminal 2					Totales				
	Nacional		Internacional		Total	Nacional		Internacional		Total	Nacional		Internacional		
	llegadas	salidas	llegadas	salidas		llegadas	salidas	llegadas	salidas		llegadas	salidas	llegadas	salidas	Gran Total
Ene	470,174	448,284	238,623	243,289	1,400,370	274,847	268,745	131,960	137,014	812,566	745,021	717,029	370,583	380,303	2,212,936
Feb	440,454	434,058	197,389	189,473	1,261,374	263,048	264,278	102,811	104,334	734,471	703,502	698,336	300,200	293,807	1,995,845
Mar	477,579	468,931	218,738	232,779	1,398,027	287,156	285,124	123,567	132,265	828,112	764,735	754,055	342,305	365,044	2,226,139
Abr	433,064	443,619	206,411	230,978	1,314,072	255,859	256,186	123,628	142,185	777,858	688,923	699,805	330,039	373,163	2,091,930
May	295,489	279,938	116,176	121,025	812,628	225,121	219,821	80,740	79,506	605,188	520,610	499,759	196,916	200,531	1,417,816
Jun	364,193	367,887	185,412	180,642	1,098,134	292,026	292,788	114,499	110,043	809,356	656,219	660,675	299,911	290,685	1,907,490
Jul	417,186	420,288	245,590	259,873	1,342,937	336,845	345,066	147,602	158,024	987,537	754,031	765,354	393,192	417,897	2,330,474
Ago	393,448	381,572	241,215	232,928	1,249,163	327,270	317,692	152,587	148,295	945,844	720,718	699,264	393,802	381,223	2,195,007
Sep															
Oct															
Nov															
Dic															
Total	3,291,587	3,244,577	1,649,554	1,690,987	9,876,705	2,262,172	2,249,700	977,394	1,011,666	6,500,932	5,553,759	5,494,277	2,626,948	2,702,653	16,377,637

Figura 1.4 **Pasajeros** (Tomado de *AICM en cifras 2009*)

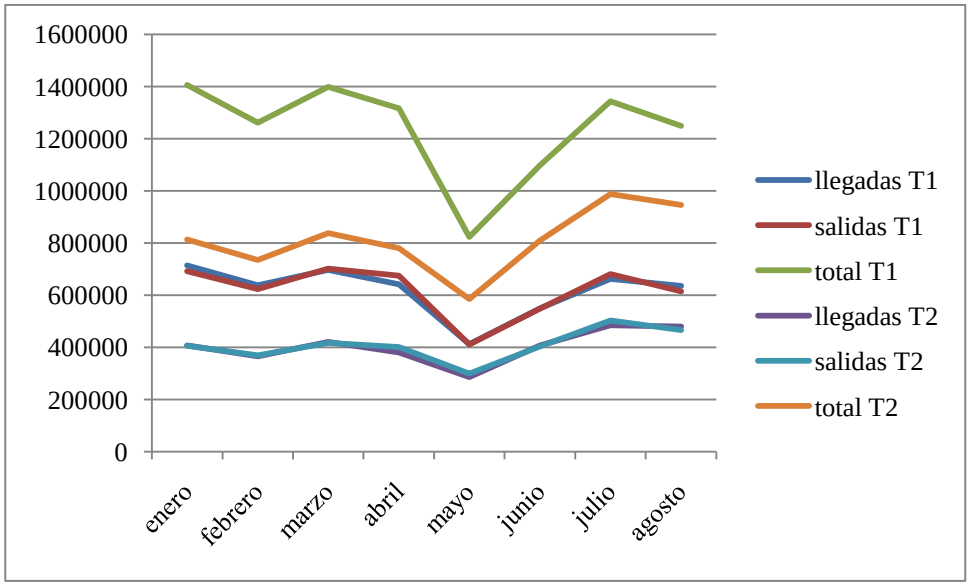


Figura 1.5 **Gráfica Pasajeros**

Como se puede ver en la figura 1.5 cada mes existen aproximadamente 1,000,000 de pasajeros por terminal (promediando el número de pasajeros en la terminal 1 y en la terminal 2). Eso representa una enorme cantidad de entradas monetarias y/o penalizaciones si los vuelos llegan a retrasarse.

En la figura 1.6 se muestran las operaciones promedio (despegues y aterrizajes) por hora en un día en el AICM.

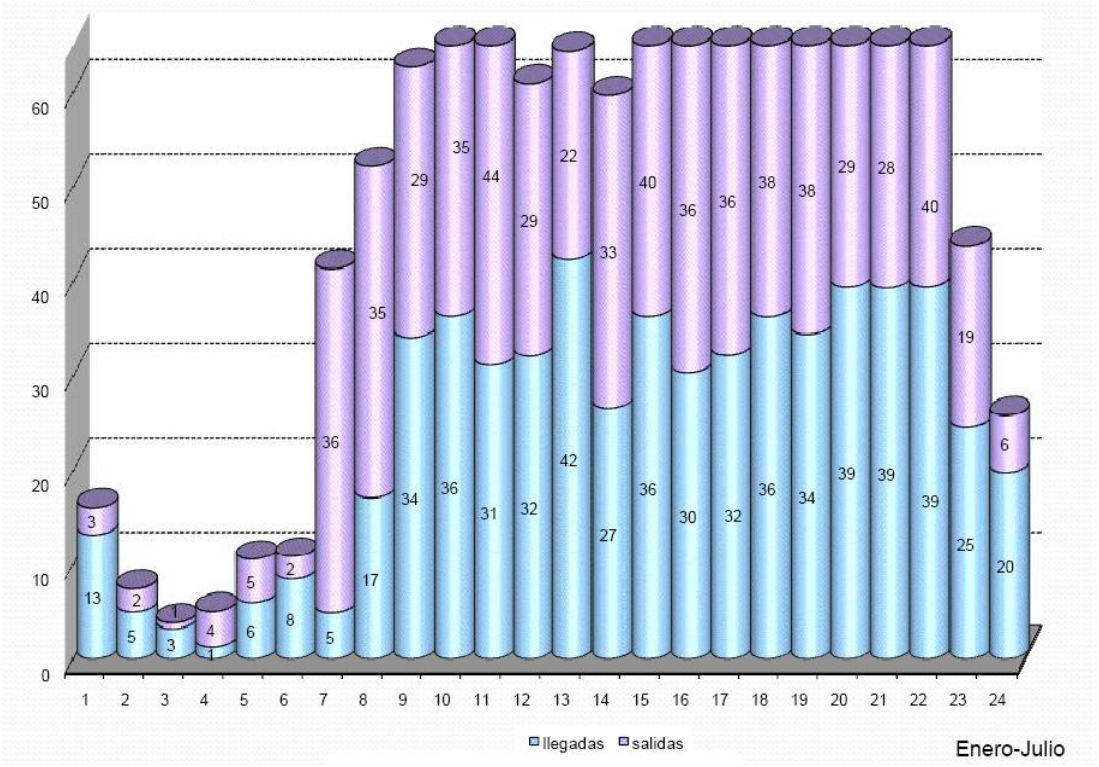


Figura 1.6 Operaciones promedio por terminal por hora (Tomado de AICM en cifras 2009)

Como se puede ver en la figura anterior, en el aeropuerto de la ciudad de México, las horas pico son desde las 9 hrs. hasta las 22 hrs., lo que es la mayor parte del día. Es por eso que es de suma importancia que los vuelos no se retrasen ya que ocasionarían grandes pérdidas económicas para las aerolíneas de cada uno de estos despegues o aterrizajes.

Basándonos en las cifras del AICM podemos ver que en proporción se tiene, aproximadamente, 630 vuelos diarios de los cuales un 50% son despegues y un 50% son aterrizajes, lo cual está muy balanceado facilitando así su análisis. Con base en estos datos se realizará la simulación del sistema actual. Se tendrán que generar datos aleatorios para las llegadas de los vuelos, cómo se verá en el capítulo 5. Posteriormente se programará un

método heurístico (capítulo 6) basado en un modelo matemático de programación exacta (capítulo 4) y en la teoría de *scheduling*. Por último se comparará la simulación con los resultados obtenidos del método heurístico para así poder demostrar la minimización de costos y retrasos en la programación de despegues y aterrizajes de vuelos en una pista en el Aeropuerto Internacional de la Ciudad de México (AICM).

A continuación se muestra el estado del arte para mostrar las investigaciones que se han estado haciendo en los últimos años acerca de este tema.

2. ESTADO DEL ARTE

El problema de programación de aterrizajes y despegues se ha resuelto de diferentes maneras en los últimos años. En muchos casos, la solución es cambiar la secuencia de los vuelos aunque no especifican la factibilidad de hacerlo en la realidad. Se tendría que hablar con especialistas en torre de control para saber que tan factible es hacer esos movimientos en el aire.

A continuación se muestran algunos ejemplos:

- Miltiadis A. Stamatopoulos, Konstantinos G. Zografos y Amedeo R. Odoni realizaron en el año 2002 el siguiente estudio (STAMATOPOULOS Miltiadis A., *et al*., 2002):

La planeación de la expansión en los aeropuertos debe ser una perspectiva a largo plazo (15 a 50 años). Stamatopoulos junto con otras personas presenta un grupo de modelos para apoyar a los operadores y administradores en la planeación estratégica para la expansión y optimización del campo de aviación (pistas, toma de taxis, etc.) y para mejorar los procesos operativos y la administración de la demanda. La planeación estratégica necesita la habilidad de examinar las implicaciones para el nivel de servicio en el aeropuerto en diversos escenarios, con hipótesis acerca de las condiciones futuras.

A pesar de existir modelos de simulación que ayudan a la asistencia detallada en el diseño, no suelen ayudar mucho en proporcionar una decisión estratégica en poco tiempo y esfuerzo. Es por eso que utilizan modelos estocásticos.

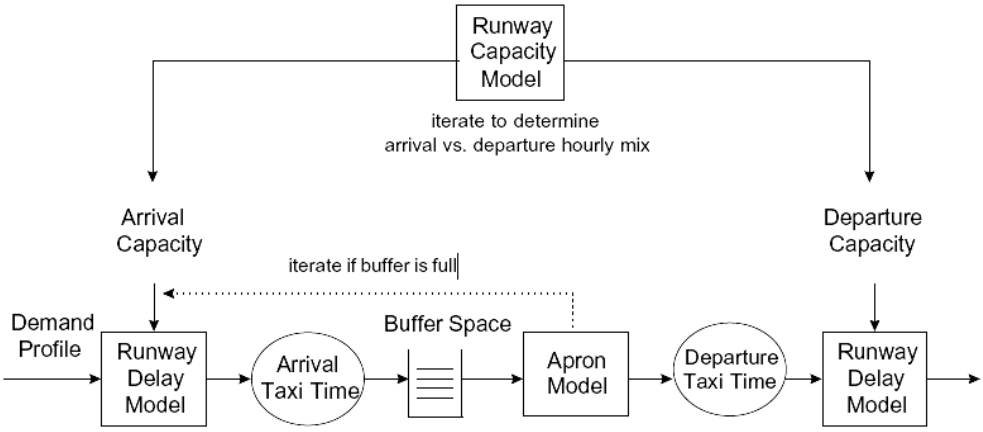


Figura 2.1 **Modelo de capacidad de las pistas** (Tomado de Miltiadis A 2002)

➢ Antonio Alonso, Laureano F. Escudero y M. Teresa Ortuño en el año 1998 (ALONSO Antonio, *et al*., 1998) presentan un modelo y un algoritmo robusto para el problema de la administración del flujo de tráfico aéreo con incertidumbre en llegadas, salidas y la capacidad aérea por condiciones climáticas. Con este propósito, utilizan el modelo determinístico 0-1 y con él crean dos modelos estocásticos binarios dependiendo del tipo de políticas que se utilicen. Se utiliza un escenario con multiobjetivos basado en un esquema de recursos. Presentan un acercamiento por medio de la solución relajada y los resultados obtenidos.

➢ Gregory C. Carr, Heinz Erzberger y Frank Neuman en el año 1998 (CARR Gregory C., *et al*. 1998) escriben que el objetivo de la secuencia y horario de las llegadas en el control del tráfico aéreo, es poder encontrar un equilibrio entre la demanda y la capacidad del aeropuerto minimizando al mismo tiempo los retrasos. Estos autores proponen un método de programar un banco de aviones de llegada con un orden de llegada preferido, en lugar de utilizar la secuencia FCFS *(first come first served)* basada en el tiempo estimado de llegadas en la pista. El método se prueba por medio de simulación mostrando resultados favorables.

➢ Hanif D. Sherali, Ebru K. Bish y Xiaomei Zhu escribieron en año 2005 (SHERALI Hanif D., *et al*., 2005):

El problema de asignación de vuelos (FAP) se enfrenta a la asignación de tipos de aviones, cada uno con una diferente capacidad, costos operacionales y potencial de ganancias para generar el horario y secuencia de vuelos. Resolver el FAP siempre ha sido de gran dificultad. Estos autores presentan un tutorial con ciertos modelos que dan un acercamiento a la resolución de dicho problema. Dicho tutorial contiene la integración del FAP con las decisiones de otras aerolíneas, horarios y asignación de personal; una propuesta de técnicas de solución tomando en cuenta los pronósticos de demanda; estudios dinámicos, etc.

- Xiao-Bing Hu y Wen-Hua Chen utilizan en el año 2005 (HU Xiao.Bing, *et al*., 2005) algoritmos genéticos (GA) para resolver el problema de secuenciación y horarios de llegadas (ASS) en un aeropuerto internacional. El algoritmo se utiliza para resolver el problema ASS dinámico. Por medio de la simulación se muestra que el algoritmo propuesto muestra resultados favorables en un ambiente dinámico.

- N. Bäuerle, O. Engelhardt-Funke y M. Kolonko describen en su artículo en el año 2006 (BÄUERLE N., *et al*., 2005) un modelo para el procedimiento de aterrizaje en un aeropuerto. Es necesario tener una cierta distancia, por seguridad, entre dos aviones que van a aterrizar. Esta varía dependiendo del tipo de aviones en cuestión. Se utiliza teoría de colas con una y dos pistas encontrando el tiempo promedio de espera para cada avión. (pasan de un *M/SM/1* a un *M/G/1*). Posteriormente, se utilizan heurísticas para las dos pistas (lanzamiento de moneda, *type splitting*, Round Robin y variantes de la regla ensamblar la menor carga).

- Paolo Dell'Olmo y Guglielmo Lulli describen en 2006 (DELL'OLMO Paolo, *et al*., 2001) una arquitectura jerarquizada en dos niveles para la administración de los problemas del tráfico aéreo, por medio de modelos matemáticos. El primer nivel representa la red rutas aéreas y los flujos en cada arco de la red. El segundo nivel, se refiere a la ruta de cada uno de los aviones. Este modelo permite asignar la ruta del tráfico aéreo para cada avión y optimizar la capacidad aérea. Se presentan tanto, técnicas de solución exactas para un MIP *(mixed integer problem)*, como heurísticas para la obtención de los resultados.

- Konstantin Artiouchine, Philippe Baptiste y Christoph Dürr estudian en el año 2007 (ARTIOUCHIN Konstantin, *et al*., 2007), el problema del horario en el flujo aéreo. Existen ventanas de tiempo en las cuales los aterrizajes son posibles para cada avión. El objetivo sería determinar los tiempos de aterrizaje considerando estas ventanas de tiempo, lo cual maximiza el mínimo tiempo requerido entre dos aterrizajes consecutivos. Se estudian varios casos que se pueden resolver en tiempo polinomial. También proporcionan una formulación MIP más completa cuando las ventanas de tiempo son iguales. Finalmente proporcionan una solución utilizando branch and cut con ventanas de tiempo aleatorias.

- Hamsa Balakrishnan y Bala Chandran explican en el año 2006 (BALAKRISHNAN Hansa, *et al*., 2006), el funcionamiento de los aeropuertos y sus aterrizajes con la política FCFS. Describen la problemática de la creación de la secuencia y los horarios de aterrizaje, ejemplificándolo con el aeropuerto de la ciudad de Denver.

- M. J. Soomer comienza un estudio acerca de la secuencia de los aterrizajes en el año 1978. Lo continúa en el año 2007 (SOOMER M.J. *et al*., 2007). Considera que la política utilizada actualmente en colas, FCFS, no toma en cuenta las preferencias de las aerolíneas y crea un modelo matemático MIP en el cual toma en cuenta las preferencias de las aerolíneas (sus funciones de costos). Considera ventanas de tiempo para cada aterrizaje e incluso la distancia que se debe tener entre aterrizajes consecutivos, dependiendo del tipo de aviones del que se trate. Su objetivo será minimizar la suma de los costos de las aerolíneas y con éstos, minimizar los retrasos creando una secuencia óptima para cada día en cuestión. Posteriormente utiliza este modelo y lo adapta a una heurística llamada *búsqueda local* para encontrar una mejora en la secuencia de aterrizajes.

A continuación se muestra una tabla para simplificar lo que se ha hecho en los últimos años, descrito anteriormente, con los autores, años y los diversos métodos utilizados por cada uno de ellos.

Tabla 2 1 Estado del arte

	Métodos exactos					Heurísticas				
	MIP	Estático	Dinámico	Determinístico	Estocástico	Algoritmos genéticos (dinámico)	Búsqueda local	"type splitting", Round Robin, etc.	Branch & cut (estocástico)	Simulación
Antonio Alonso, Laureano F. Escudero, M. Teresa Ortuño (1998)	X	X		X	X					
Gregory C. Carr, Heinz Erzberger, Frank Neuman (1998)										X
Hanif D. Sherali, Ebru K. Bish, Xiaomei Zhu (2005)	X		X		X					
Xiao-Bing Hu, Wen-Hua Chen (2005)						X				X
N. Bäuerle, O. Engelhardt-Funke, M. Kolonko (2006)								X		
Paolo Dell'Olmo, Guglielmo Lulli (2006)	X						X			
Konstantin Artiouchine, Philippe Baptiste, Christoph Dürr (2007)	X	X		X					X	
M. J. Soomer (1978 – 2007)	X	X		X			X			X
Hamsa Balakrishnan, Bala Chandran (2006)	Explican el funcionamiento de los aeropuertos, como sus aterrizajes con la política FCFS. Describen la problemática de la creación de la secuencia y los horarios de aterrizaje ejemplificándolo con el aeropuerto de la ciudad de Denver									

Como se pudo observar, existen muchas formas de abordar este problema por medio de heurísticas apoyadas de métodos exactos, en su generalidad.

Este trabajo se basa en el modelo propuesto por Soomer (SOOMER M.J. *et al*., 2007) debido a que considera más aspectos como: ventanas de tiempo, distancia entre aviones según su tamaño, funciones de costos asociadas a los retrasos, etc., en comparación a otros modelos existentes en la literatura. Cabe mencionar que el modelo de Soomer toma en cuenta únicamente la secuencia para aterrizajes, sin embargo en este trabajo se modificará para considerar tanto aterrizajes como despegues. El modelo se resolverá a través de la metaheurística llamada *Búsqueda Tabú*.

En el siguiente capítulo se explicará la teoría de *scheduling* y sus diferentes variantes para un mejor entendimiento del modelo , tanto matemático como heurístico.

3. MARCO TEÓRICO

En este capítulo se explicará brevemente la diferencia entre los modelos matemáticos que proporcionan una solución exacta y los métodos de aproximación, como lo son los métodos heurísticos. Posteriormente se explicará más a fondo la teoría de programación de actividades *scheduling* y sus diferentes variantes, ya que en esta teoría está basado el modelo del método heurístico explicado en el capítulo 6.

En la investigación de operaciones existen varios métodos para resolver problemas según su naturaleza. Existen los métodos exactos, basados en modelos matemáticos, y los métodos heurísticos.

Los modelos matemáticos consisten en una función objetivo o multiobjetivo que se quiere maximizar o minimizar (en los casos más complejos se busca un maximin, minimax, etc.) seguida de una serie de restricciones expresadas en funciones matemáticas, como por ejemplo un presupuesto. Estas restricciones son siempre igualdades *(=)* o desigualdades *(*$<$, $>$, $\leq$, $\geq$*)*. Estos métodos encuentran siempre la mejor solución, es decir el óptimo en el problema (ya sea el máximo o el mínimo).

Existen algunos casos donde la complejidad del problema es NP-duro o NP-completo. En el caso del problema anteriormente señalado que es la reorganización de una secuencia de aterrizajes y despegues, el problema se denomina *combinatorio*. Esto es porque para encontrar la mejor solución se tendrían que hacer todas las combinaciones posibles, por lo cual se tendrían que hacer *n!* intentos en la búsqueda de la mejor secuencia donde *n* es el tamaño de la entrada. Cuando uno se enfrenta a este tipo de problemas no es conveniente utilizar un método exacto ya que la computadora tardaría días e incluso años en encontrar la mejor solución, dependiendo del tamaño de *n*.

En estos casos se debe buscar otra alternativa para encontrar una solución al problema. Esta alternativa se llama la solución mediante métodos heurísticos. Estos métodos son basados en la experiencia, como su nombre lo dice. Tras el transcurso del tiempo, se han encontrado varias *metaheurísticas* diferentes que dan muy buenos resultados dependiendo del tipo de problema que se trate. Estos métodos, ya que no son exactos, no

prometen encontrar el óptimo, pero si una muy buena solución. (Ver apéndice A para una mejor explicación del tema de complejidad computacional).

El problema a tratar en este trabajo de tesis es del tipo NP-duro (PINEDO, 2008) y, por lo tanto, necesita el uso de una metaheurística para resolverse, esto será tratado en capítulos posteriores.

La programación de una secuencia de aterrizajes y despegues se asemeja mucho al tema de *scheduling* o programación de actividades que se presenta a continuación con todas sus variantes.

3.1. PROBLEMA DE PROGRAMACIÓN DE ACTIVIDADES (SCHEDULING)

La palabra programación de actividades o calendarización la usamos día con día. Es el plan que nos dice cuáles y cuándo hacer, ciertas actividades.

Muchas veces, esta programación de actividades tiene que ver con su secuencia. Primero se tiene que hacer la actividad *A* y una vez terminada ésta, se comienza la actividad *B*, y así sucesivamente.

Muchos eventos o sucesos son impredecibles, por lo tanto, existen cambios en un horario o en una programación de actividades. De todas formas, un horario sigue siendo útil para la programación de las tareas aún cuando éste tenga cierta incertidumbre inherente.

El calendario o la programación de actividades es un plan tangible, sin embargo, el proceso de creación de este plan es intangible. Este proceso tiene 2 etapas: secuenciación y programación de actividades (scheduling). En la primera etapa se planea la secuencia de las actividades o la decisión de cómo seleccionar la siguiente actividad. En la segunda etapa se planea el tiempo de inicio y el tiempo de término de la actividad. Junto con estos dos tiempos se requiere tener un tiempo o periodo de seguridad asociado, debido a la incertidumbre. Dadas las actividades y los recursos, junto con la incertidumbre, el problema de la programación de actividades determinará el tiempo a realizarse de cada una de ellas, tomando en cuenta las capacidades de nuestros recursos. Este problema está ligado con la toma de decisión de precedencia o jerarquía (para realizar la actividad *X* se necesita antes la actividad *Y*), que en muchos casos no es tan claro de establecer.

En el proceso de la programación de actividades se necesita saber el tipo y la cantidad de cada recurso para determinar el tiempo factible para realizar cada tarea o actividad. Estas tareas se deben describir en términos de los requerimientos de cada recurso: la duración, el tiempo más temprano en el que puede comenzar la tarea y el tiempo en el que debe ser terminada (o fecha de entrega). En general, el tiempo de duración de una actividad es incierto, sin embargo en muchos casos se omite al principio este hecho para plantear el problema.

En general se puede decir que un problema de programación de actividades (scheduling) está definido por la información acerca de los recursos y de las tareas o actividades. Resolver estos problemas puede ser altamente complejo, es por eso que son de gran ayuda las soluciones de problemas formales. Los modelos formales ayudan a entender el problema para encontrar, posteriormente, una buena solución. Un ejemplo de estos modelos son los *Diagramas de Gantt* como se muestra en la figura 3.1. Los diagramas básicos de Gantt suponen tiempos de procesos determinísticos.

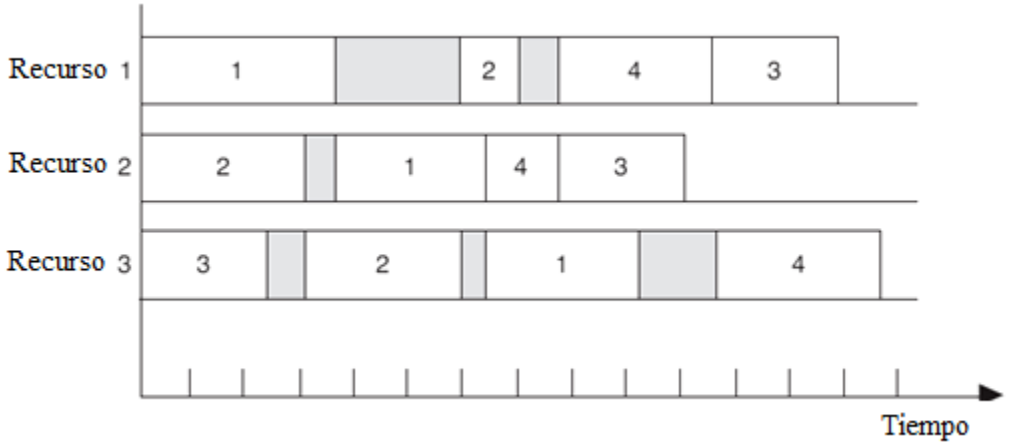


Figura 3.1 **Diagrama de Gantt** (Tomado de Baker, 2009)

Estos diagramas ayudan a visualizar la calendarización de las tareas junto con los recursos, es decir, qué tarea se realizará con qué recurso. Se pueden hacer intercambios entre las tareas para obtener información de un plan alternativo. De esta manera, los diagramas de Gantt sirven como ayuda para medidas de desempeño y comparación de diferentes planes de programación de tareas.

Dado que muchos estudios que se hicieron anteriormente en el campo de *scheduling* estaban enfocados a la manufactura, se utilizan términos relacionados a esta área para describir este tipo de problemas. Es por eso que se denotará a los recursos como *máquinas (machines)* y a las tareas o actividades como *trabajos (jobs)*. Muchas veces los trabajos consisten en varias tareas elementales llamadas *operaciones (operations).*

La teoría de scheduling se centra en modelos matemáticos. Se han desarrollado muchas técnicas que se han ido enriqueciendo con la continua interacción entre teoría y práctica. La perspectiva teórica es un acercamiento cuantitativo que intenta plasmar el problema mediante una estructura matemática. Comienza con la descripción de los recursos y las tareas mediante un objetivo de toma de decisiones dentro de una función explícita que es la función objetivo.

Idealmente, la función objetivo debe consistir en los costos de los que depende la decisión de la programación de tareas. Sin embargo, en la vida real es muy difícil medir estos costos; existen casos donde es difícil siquiera identificarlos. En general existen 3 tipos de metas en la toma de decisiones en scheduling: tiempo de procesamiento (*turnaround*), tiempo de terminación (*timeliness*) y rendimiento del procesamiento (*throughput*). El tiempo de procesamiento mide el tiempo necesario para completar la tarea. El tiempo de terminación, como su nombre lo indica, mide si se logró terminar una cierta tarea en un tiempo límite. El rendimiento de procesamiento mide el número de trabajos completados durante un cierto periodo fijo. Los primeros dos objetivos necesitan un análisis más a fondo debido a que se enfocan en cada una de las tareas y no al conjunto total de trabajos (como sucede con el rendimiento).

Los modelos se categorizan dependiendo de la configuración de los recursos y la naturaleza de las tareas o actividades. Se puede tener un modelo que conste de una o varias máquinas. Si consiste de una máquina, generalmente el trabajo consiste únicamente de una etapa; pero si son varias máquinas, el trabajo puede requerir de muchas etapas. En ambos casos las máquinas pueden existir como unidad o en paralelo.

En cuanto a la naturaleza de las tareas, si el conjunto de trabajos no cambia a través del tiempo, se trata de un sistema *estático.* Si, por el contrario, surgen nuevos trabajos con

el transcurso del tiempo, se trata de un sistema *dinámico*. Cuando las condiciones se saben con certeza, el modelo será *determinístico*; si existe incertidumbre acerca de las condiciones (con distribuciones de probabilidad explícitas), el modelo será *estocástico*.

Existen dos tipos de restricciones en cuanto a la factibilidad de los problemas de scheduling: los límites de capacidad de las máquinas y las restricciones tecnológicas en cuanto a cómo deben elaborarse los trabajos. Por lo tanto, un problema de scheduling tiene decisiones en cuanto a la localización y secuenciación. Si el modelo toma en cuenta estas dos decisiones en conjunción con el nivel de servicio, el problema se denomina *programación de actividades con seguridad (safe scheduling)*. En este caso se incluye la decisión de aceptar o rechazar el trabajo para mantener una cierta satisfacción del cliente.

Pueden existir problemas que sean únicamente de asignación o únicamente de secuenciación. En el primer caso, puede ser como el conjunto de problemas de mezcla de productos con recursos limitados, en el segundo caso, los problemas tienen que ser vistos y entendidos desde la teoría de programación de actividades.

La teoría de *scheduling* tiene una gran variedad de metodologías. Este campo de conocimiento se ha desarrollado mucho en la aplicación y evaluación de procesos combinatorios por medio de métodos exactos, técnicas de simulación y por medio de soluciones aproximadas mediante métodos heurísticos. La selección del método de solución apropiado depende de la naturaleza del modelo y de la función objetivo.

Los problemas de *scheduling* se encuentran dentro de la clase NP-duro (*NP-hard*) (Pinedo 2008), lo cual significa que es muy probable que no se pueda encontrar una solución óptima por medio de técnicas de optimización para instancias muy grandes de un problema. En ese caso es mejor utilizar métodos heurísticos que, a pesar de no garantizar la optimalidad, proporcionan una muy buena solución.

En la figura 3.2 se muestra la simplificación de los diferentes tipos de *scheduling* que existen para un mejor entendimiento. Estos diferentes tipos serán explicados con mayor profundidad a lo largo de éste capítulo.

En la siguiente figura se muestra de manera gráfica los diferentes tipos de programación de tareas que existen y como va a aumentando su complejidad.

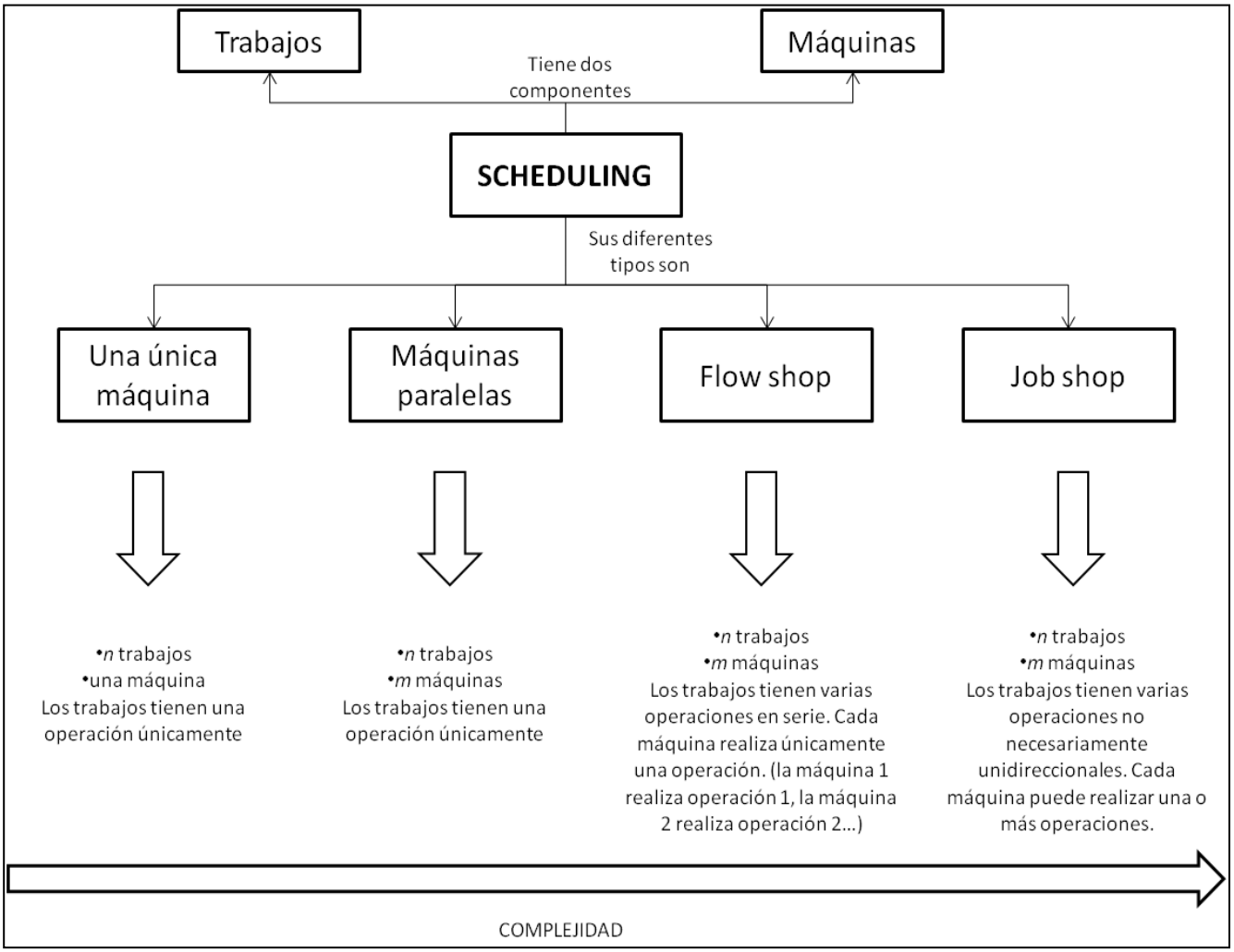


Figura 3.2 Scheduling

3.1.1. Secuenciación de una única máquina

La sola secuenciación es un problema particular de scheduling en donde se determina el orden de los trabajos. El problema de secuenciación más simple es aquel en donde existe un solo recurso (una única máquina) y donde todos los tiempos de procesamiento son

determinísticos. En muchos casos, estos problemas son una parte de un problema mucho más grande, por ejemplo, un proceso de operaciones múltiples en donde existe un cuello de botella en una etapa, el cual se tiene que analizar por separado.

Las limitaciones de los problemas con una única máquina se caracterizan por las siguientes condiciones:

1) Existen *n* trabajos de una operación disponibles simultáneamente (en *t=0* en el caso más simple).
2) Las máquinas pueden procesar únicamente un trabajo a la vez.
3) Los tiempos de disposición de los trabajos son independientes de la secuencia de los trabajos y están incluidos en los tiempos de procesamiento.
4) La información de los trabajos es determinística y conocida.
5) Las máquinas siempre están disponibles (no existen averías o descomposturas).
6) Las máquinas nunca están en ocio (*idle*) cuando hay trabajos esperando a ser procesados.
7) Una vez que comienza una operación, la máquina continúa sin interrupciones.

Bajo estas condiciones existe una correspondencia de uno-a-uno entre la secuencia de los *n* trabajos y la permutación de los índices *1, 2, … ,n*. Por lo tanto el número de soluciones para el problema de una única máquina es *n!*.

La información conocida o datos de entrada (*inputs*) en estos problemas son:

- Tiempo de procesamiento *(p_j) (processing time)*: tiempo requerido para el procesamiento del trabajo *j*.
- Fecha de comienzo *(r_j) (release date)*:el tiempo en el que el trabajo *j* está disponible para comenzar a procesarse.
- Fecha de entrega *(d_j) (due date)*: el tiempo en el que el trabajo *j* debe ser terminado.

La información obtenida de la programación de actividades *(outputs)* es:

- Tiempo de terminación *(C_j) (completion time)*: tiempo en el cual el procesamiento del trabajo *l* se termina.
- *Flowtime (F_j)*: el tiempo que el trabajo *j* pasa en el sistema

$$F_j = C_j - r_j \tag{1}$$

- Retraso *(L_j) (lateness)*: la cantidad de tiempo en la que el tiempo de terminación trabajo *j* excede o se anticipa a su fecha de entrega

$$L_j = C_j - d_j \tag{2}$$

Las dos últimas cantidades representan dos diferentes tipos de servicio. *Flowtime* mide la respuesta del sistema a demandas individuales por el servicio (también llamado *turnaround* anteriormente). El retraso mide la desviación (positiva o negativa) en relación a la fecha de entrega. Un retraso positivo es cuando el trabajo se entregó después de su fecha de entrega y uno negativo significa que el trabajo terminó antes de su fecha de entrega. Normalmente existen penalizaciones asociadas a un retraso positivo pero no existen beneficios asociados a un retraso negativo. Es por eso que se utiliza una cantidad que mida únicamente el retraso positivo:

- Tardanza *(T_j) (tardiness)*:

$$T_j = \max_{1 \le j \le n} \{0, L_j\} \tag{3}$$

La programación de actividades se evalúa generalmente mediante cantidades agregadas que incluyan información de todos los trabajos (medida de desempeño) por ejemplo:

- Flowtime total:

$$F = \sum_{j=1}^{n} F_j \tag{4}$$

- Tardanza total:

$$T = \sum_{j=1}^{n} T_j \tag{5}$$

- Máximo flowtime:

$$F_{max} = \max_{1 \le j \le n} \{F_j\} \tag{6}$$

- Máxima tardanza:

$$T_{max} = \max_{1 \le j \le n} \{T_j\} \tag{7}$$

- Número de trabajos con tardanza o el total de unidades de penalización:

$$U = \sum_{j=1}^{n} \delta(T_j) \tag{8}$$

donde:

$$\delta(x) = \begin{cases} 1 & x > 0 \\ 0 & e.o.c. \end{cases} \tag{9}$$

- Tiempo máximo de terminación:

$$C_{max} = \max_{1 \le j \le n} \{C_j\} \tag{10}$$

Bajo las suposiciones básicas $C_{max} = F_{max} = \sum p_j$ y esta cantidad es conocida como *makespan*. Cada una de las cantidades anteriormente descritas (F_{max}, T_{max}, U, F_j, T_j, etc.*)* están en función de su tiempo de terminación:

$$Z = f(C_1, C_2, \dots, C_n) \tag{11}$$

La medida de desempeño *Z* es regular si:

a) El objetivo del problema es minimizar *Z*
b) *Z* aumenta si al menos un C_k aumenta.

es decir:

$$Z' > Z \Rightarrow C'_j > C_j \qquad \text{para algún trabajo } j \tag{12}$$

En general, para los métodos de optimización, se supone una función de costos $g_j(t)$ no decreciente. Lo más común es minimizar el valor máximo $g_j(t)$ o minimizar la suma de los valores que tiene $g_j(t)$.

Sea *P* el tiempo total de procesamiento de los trabajos a programar. Es evidente que *P* es igual al tiempo de terminación del último trabajo. Es por eso que se debe encontrar el trabajo con el mínimo costo en el tiempo *P*, este trabajo será asignado en la última posición dentro de la secuencia como se muestra en la figura 3.3.

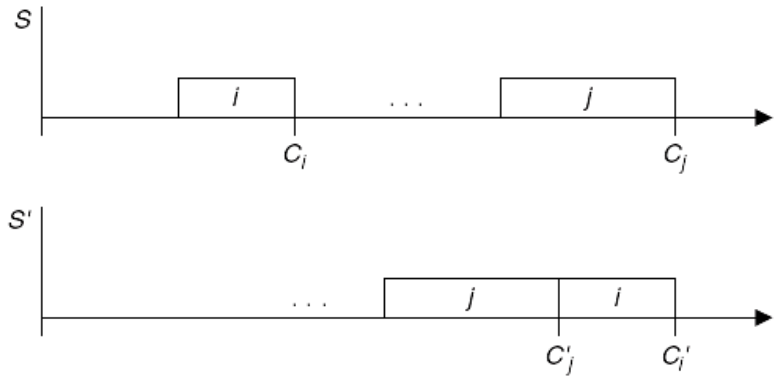


Figura 3.3 Insertar el trabajo *i* en la última posición (Tomado de Baker, 2009)

Quitando este trabajo, se repite lo mismo para los *(n - 1)* trabajos restantes hasta que todos los trabajos tengan asignada una posición en la secuencia:

$$\min_t \sum_{j=1}^{n} g_j(t) \tag{13}$$

3.1.1.1. Costos de tardanza o de terminar un trabajo antes *(earliness)*

El criterio de tardanza total es uno de los métodos más utilizados. Sin embargo, no toma en cuenta los trabajos que se terminan antes de tiempo y penaliza aquellos con tardanza. Esta perspectiva comenzó a cambiar con la llegada de *JIT (just in time)* que estipula que un producto terminado antes de tiempo *(E)* tiene que ser inventariado hasta su fecha de

entrega, mientras que el trabajo con tardanza *(T)* ocasiona problemas en las operaciones del consumidor. Por lo tanto:

$$E_j = max\{0, (d_j - C_j)\} = (d_j - C_j)^+ \tag{14}$$

$$T_j = max\{0, (C_j - d_j)\} = (C_j - d_j)^+ \tag{15}$$

Como se asume que la función de costos es lineal, se puede asociar una unidad de costo por terminar antes el trabajo $\alpha_j > 0$ y una unidad de costo de tardanza $\beta_j > 0$. La función objetivo para una programación de actividades *S* se denota por *f(S)* donde:

$$f(S) = \sum_{j=1}^{n} [\alpha_j (d_j - C_j)^+ + \beta_j (C_j - d_j)^+] \tag{16}$$

lo que es lo mismo que:

$$f(S) = \sum_{j=1}^{n} (\alpha_j E_j + \beta_j T_j) \tag{17}$$

Existen dos modelos principales para los problemas *E/T*: el primero tiene las mismas fechas de entrega para todos los trabajos y el segundo con distintas fechas de entrega.

3.1.1.2. Programación estocástica de actividades

Nos referimos a un problema de programación de actividades estocástico (*stochastic scheduling problem*) cuando los tiempos de procesamiento son aleatorios.

La secuencia de ordenar los trabajos por orden creciente de tiempos de procesamiento (*SPT, shortest processing time)* no está bien definido en este caso, dado que los tiempos de procesamiento no son conocidos de antemano. Sin embargo, es posible ordenar los trabajos en orden creciente de los tiempos *esperados* de procesamiento (*SEPT shortest expected processing time*). Similarmente, si se tiene un conjunto de trabajos con

pesos, la secuencia se crea de la misma manera: con los tiempos *esperados* (*SWEPT shortest weighted expected processing time*).

El análisis estocástico de estos problemas se ha enfocado en las mismas medidas de desempeño que en el caso determinístico (*F, T,* L_{max}*,* T_{max}*, U,* etc.), pero en el caso estocástico están denotadas por (*E(F),E(T),E(* L_{max}*),E(* T_{max}*), E(U),* etc.), Sin embargo, ninguna de estas medidas toma en cuenta un periodo o tiempo de seguridad (necesario debido a la variabilidad como en el inventario de seguridad). El periodo de seguridad se puede determinar mediante el nivel de servicio o mediante la minimización del costo total esperado.

El nivel de servicio del trabajo *j* se define como $P(C_j <= d_j)$, la probabilidad de que el trabajo *j* esté terminado para su fecha de entrega (también denotado por *SLj).* Sea b_j un objetivo del nivel de servicio. Por lo tanto:

$$SL_j = P\left(C_j \leq d_j\right) \geq b_j \tag{18}$$

Decimos que el trabajo *j* está *estocásticamente a tiempo* si se satisface su restricción de nivel de servicio. Una secuencia completa es *estocásticamente factible* si todos los trabajos están estocásticamente a tiempo.

3.1.2. Modelos de máquinas paralelas

Como se mencionó anteriormente, los problemas de programación de actividades necesitan decisiones de secuenciación y de asignación. Cuando existe un único recurso (una máquina) su asignación está completamente determinada por las decisiones de secuenciación. Para poder ver el "panorama completo" se necesita tener más de una máquina. Existen tres tipos básicos de modelos multi-máquina: sistemas paralelos, sistemas en serie (*flow shop*) y sistemas híbridos (*job shop*).

Los sistemas paralelos consisten en trabajos de una sola operación cuando se tienen varias máquinas. Se tienen *n* trabajos y *m* máquinas paralelas disponibles y se asume que cada trabajo puede ser procesado máximo por una máquina en un tiempo dado. En el

modelo básico de máquinas en paralelo, las máquinas son idénticas y los trabajos no tienen relación entre sí.

3.1.2.1. Minimización del tiempo de terminación de todos los trabajos *(Makespan)*

En el modelo de una única máquina el *makespan* es una constante. Pero en el problema estático de máquinas paralelas, cuando se permite la interrupción de los trabajos (*preemption*), el procesamiento de un trabajo puede ser interrumpido para ser terminado posteriormente en la misma máquina o incluso en una máquina diferente. La formulación para la minimización del *makespan* M^* es:

$$M^* = min\left[\sum_{j=1}^{n} {}^{p_j}/_{m},\ \max_{1\le j\le n}\{p_j\}\right] \tag{19}$$

En la figura 3.4 está descrito un ejemplo de 8 trabajos, a los cuales se les permite la interrupción. Se puede ver que los trabajos 5 y 7 fueron empezados en las máquinas 2 y 3, respectivamente, y fueron terminados posteriormente en las máquinas 1 y 2, respectivamente. Esto da como resultado un total de 12 unidades de tiempo para la terminación de todos los trabajos. Esta sería la solución para la minimización del *makespan* en este ejemplo en específico.

Job j	1	2	3	4	5	6	7	8
p_j	1	2	3	4	5	6	7	8

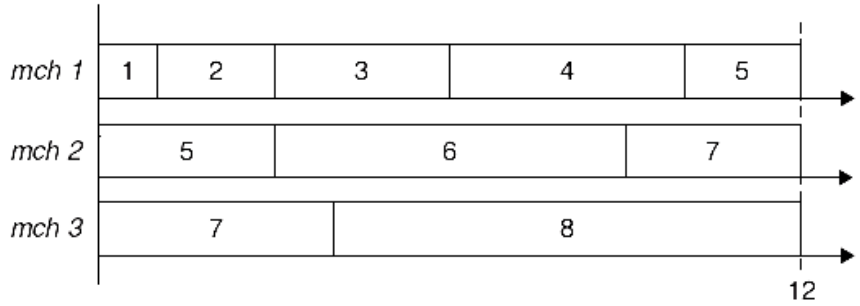


Figura 3.4 **Programación óptima para 8 trabajos** (Tomado de Baker, 2009)

Si se prohíbe la interrupción de los trabajos se requieren métodos como programación dinámica. Se recomienda que *m >= 3*. Sin embargo, se pueden obtener soluciones óptimas en el caso de instancias muy "pequeñas", es por eso que se recurre a los métodos heurísticos para la resolución de problemas de este tipo. Uno de estos procedimientos es la *lista a programar (list scheduling)* en donde se tiene una cola de trabajos esperando a ser atendidos y se toma el primer trabajo de esta cola, asignándolo a una máquina libre. Este procedimiento funciona para tiempos determinísticos de procesamiento, sin embargo, no garantiza una buena solución ya que existen *n!* diferentes secuencias posibles. En consecuencia, se puede utilizar *búsqueda de vecindad* para la secuenciación de la lista o alguna otra heurística (*Búsqueda Tabú, Recocido Simulado, algoritmos genéticos,* etc.). Estos métodos heurísticos se pueden usar también para el caso estocástico.

Para obtener resultados subóptimos se puede establecer una cota superior. Sea *M* el makespan obtenido en una heurística y M^* la solución óptima, entonces:

$$M = rM^* \qquad para \;\; r > 1 \tag{20}$$

obligando así a que el resultado sea no peor que *r-veces* el óptimo. De esta manera se puede establecer un nivel de acercamiento a la solución óptima.

En el caso de que los trabajos tengan relaciones de precedencia, se tiene un *árbol de conjuntos (assembly tree*), en donde ningún trabajo tiene más de un sucesor directo. En estos casos el último trabajo (aquel que no tiene sucesor) se llama *trabajo terminal (terminal job)* y $p_j = 1$ *para toda j* (todos los trabajos duran lo mismo).

En la figura 3.5 se muestra un ejemplo de un *árbol de conjuntos* en donde existen trabajos que tienen más de un predecesor (los trabajos 8, 7 y 1) pero ninguno tiene más de un sucesor. También se puede ver la programación de estos trabajos en 3 máquinas respetando esas reglas de precedencia. Sin embargo, en este ejemplo no se permite la interrupción de trabajos, como en ejemplos anteriores.

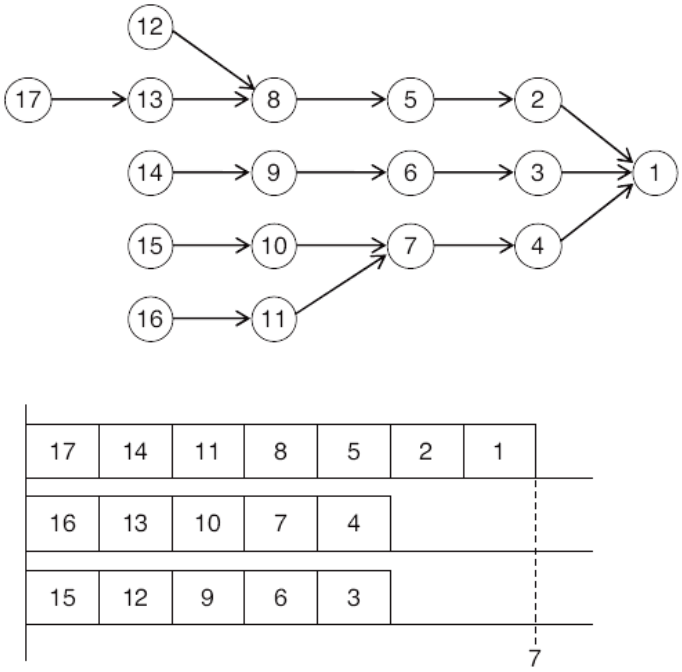


Figura 3.5 **Árbol de conjuntos** (Tomado de Baker, 2009)

En el caso de tener más de un sucesor, o varias secuencias, el árbol y su solución se pueden ver como se muestra en la figura 3.6 (en este caso, tampoco se permite la interrupción de trabajos).

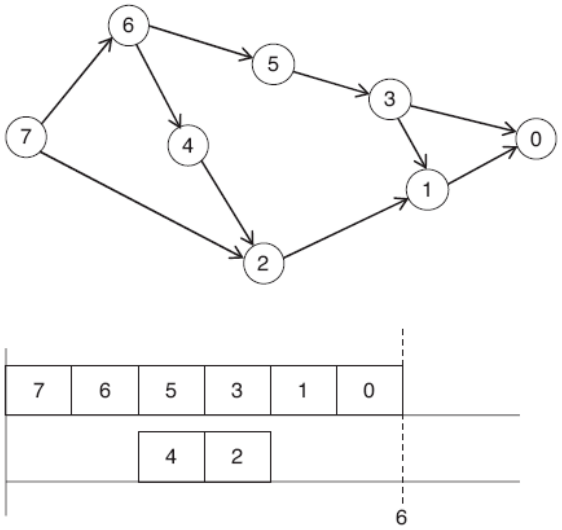


Figura 3.6 **Árbol con trabajos con más de un predecesor** (Tomado de Baker, 2009)

Si se tienen trabajos que pueden ser interrumpidos (*preemptable jobs*), se tiene una mejor flexibilidad al generar la secuencia. El makespan obtenido con trabajos con interrupciones, es al menos tan bueno como el obtenido con trabajos que no se les permite ser interrumpidos. En la figura 3.7 se muestra la red de precedencia de trabajos. En el lado izquierdo está el caso en el que no se permite la interrupción de éstos y a la derecha está el caso en el que sí se permite. Los trabajos 1, 2, 3 y 4 tienen una duración de una unidad de tiempo, mientras que los trabajos 5, 6 y 7 tienen una duración de 2 unidades de tiempo. Estos últimos trabajos son los que se van a poder interrumpir, para hacer más eficiente el makespan. En la figura 3.8 se muestra el resultado de la programación de los trabajos sin interrupción, dando como resultado un makespan de 7; mientras que en la figura 3.9 se muestra la programación de los trabajos con interrupción arrojando un makespan de 6. Por lo tanto se puede ver que se puede mejorar en una unidad de tiempo haciéndolo más eficiente en el segundo caso.

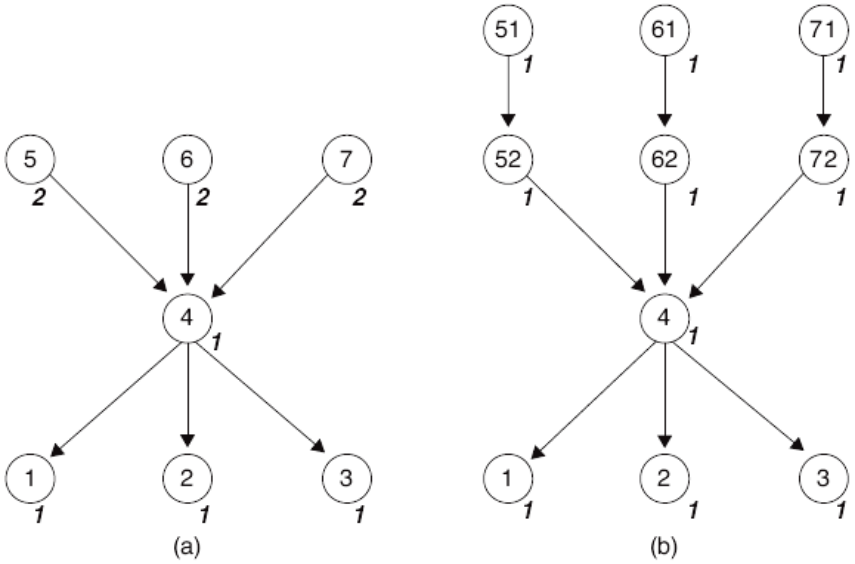


Figura 3.7 Ejemplo de 7 trabajos en (a) y en (b) su versión con interrupciones *(preemption)*

(Tomado de Baker, 2009)

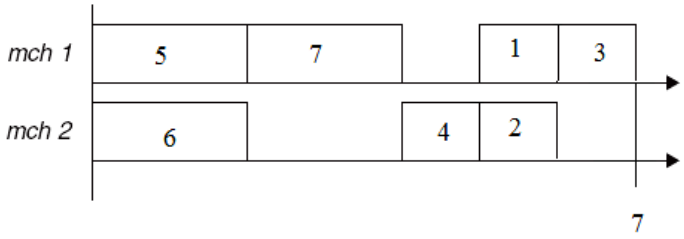


Figura 3.8 **programación de los 7 trabajos sin interrupción** (Basado en Baker, 2009)

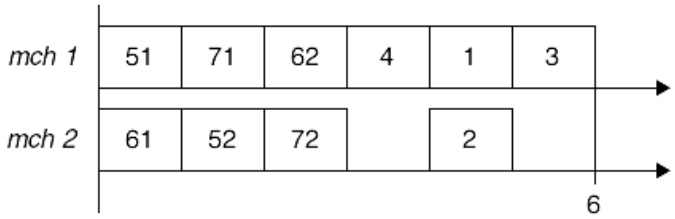


Figura 3.9 **programación de los 7 trabajos con interrupción** (Tomado de Baker, 2009)

3.1.2.2. Minimización del tiempo total que los trabajos pasan en el sistema *(total flowtime)*

La generalización del problema *F* en máquinas paralelas tiene la siguiente notación:

- $p_i(j)$ = tiempo de procesamiento del *j-ésimo* trabajo en la secuencia en la máquina *i*.
- $F_i(j)$ = flowtime del *j-ésimo* trabajo en la máquina *i*.
- n_i = número de trabajos a procesar en la máquina *i*.

La función objetivo es:

$$F = \sum_{i=1}^{m}\sum_{j=1}^{n_i} F_i(j) = \sum_{i=1}^{m}\sum_{j=1}^{n_i}(n_i - j + 1)p_i(j) \tag{20}$$

A diferencia del caso de una única máquina, el caso de máquinas paralelas tiene más flexibilidad al elegir los coeficientes, dado que n_i es arbitrario donde $n_1 + n_2 + \ldots + n_m = n$. Sin embargo, el producto escalar no puede minimizarse a menos que los n_i valores difieran a lo mucho en *1*.

$$|n_i - n_k| \leq 1 \qquad \forall\, par\ (i,k)$$

(21)

En particular, si *n* es un múltiplo de *m*, es recomendable asignar el mismo número de trabajos a cada máquina ($n_1 = n_2 = \ldots = n_m$). Una vez que se determinaron los valores n_i, se construye la secuencia asignando los *m-trabajos* más largos a las *m-máquinas* y así sucesivamente hasta que todos los trabajos sean asignados.

El problema de flowtime con peso (*Fw*) es NP-duro también (Pinedo 2008). Incluso para máquinas idénticas, la programación dinámica es de gran ayuda en estos casos pero se vuelve impráctica aún para problemas de tamaño "mediano". Estos problemas tienen dos propiedades.

1) Toda solución óptima debe tener un orden SWPT *(shortest weighted processing time)* en cada máquina.
2) Se puede calcular una cota inferior para el valor óptimo *Fw*.

Sea

B(1) = el valor máximo de *Fw* para un conjunto de trabajos si existiera sólo una máquina.

B(n) = el valor máximo de *Fw* para un conjunto de trabajos si existieran *n* máquinas.

Entonces la cota inferior para *m* máquinas ($1 \leq m \leq n$) es:

$$B(m) = \frac{1}{2m}\,[(m-1)B(n) + 2B(1)]$$

(22)

donde

$$B = max\{B(m), B(n)\}$$

(23)

3.1.2.3. Modelos estocásticos

Como se vio con anterioridad, el problema del *makespan* en máquinas paralelas con trabajos sin interrupciones, es generalmente muy difícil de resolver en el caso determinístico; lógicamente se espera que en el caso estocástico sea aún más difícil. Sin embargo, existe un caso en particular en el que la solución es muy accesible y es en el caso en el que los tiempos de procesamiento son independientes y se distribuyen exponencialmente. La resolución de estos problemas mediante heurísticas resulta ser una solución más robusta que el caso determinístico. Un ejemplo sería en donde existen *m* máquinas con trabajos sin interrupciones para el problema de makespan y además se tienen tiempos de procesamiento independientes y exponenciales. Este problema se resuelve mediante LEPT (*latest expected processing)*.

En la figura 3.10 se muestra una tabla donde, en lugar de tomar en cuenta los tiempos de procesamiento de los trabajos p_j, se tomarán sus valores esperados μ_j.

Job j	1	2	3	4
μ_j	1	2	3	4

Figura 3.10 Tiempos esperados de procesamiento (Tomado de Baker, 2009)

En la programación óptima para el caso de *2* máquinas, el tiempo entre las dos últimas terminaciones de trabajos tiene que ser menor que el tiempo de procesamiento del último trabajo.

3.1.3. Problema de máquinas en serie (*flow shop scheduling)*

Flow shop se refiere al arreglo de máquinas en serie. El trabajo está dividido en varias tareas llamadas *operaciones* y cada una de ellas se lleva a cabo en una máquina diferente. De este modo se puede decir que el trabajo es el conjunto de operaciones con una estructura de precedencia (un predecesor y un sucesor). En el modelo más simple de flow shop se tienen *m* máquinas y *m* operaciones (una operación por máquina). Las máquinas se enumeran *1, 2, ..., m* y las operaciones del trabajo *j* se enumeran *(1, j), (2, j), ..., (m, j)* como se muestra en la figura 3.11.

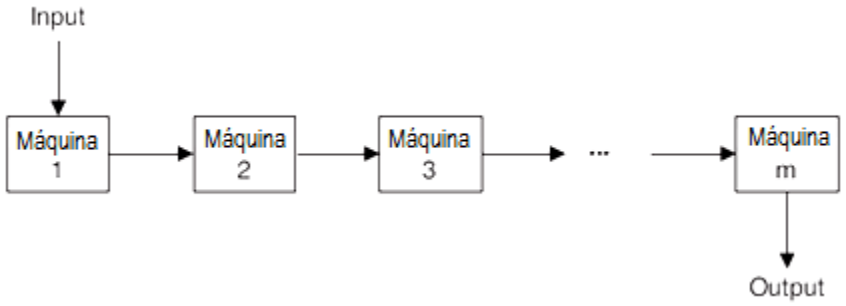


Figura 3.11 **Modelo de flow shop más simple** (Tomado de Baker, 2009)

En el caso general de flow shop los trabajos no necesariamente tienen *m* operaciones. Sin embargo, el flujo de los trabajos sigue siendo unidireccional y se puede representar como el caso más simple, en donde algunos tiempos de operación son cero.

Las condiciones de las máquinas en serie son similares a las condiciones del modelo de una única máquina:

1) Existe un conjunto de *n* trabajos multi-operación no relacionados listos para ser procesados (cada trabajo requiere *m* operaciones y cada operación necesita una máquina).
2) Los tiempos de preparación y cambio de operación (*setup*) están incluidos en los tiempos de procesamiento.
3) La descripción de cada trabajo se conoce de antemano.
4) Todas las máquinas están disponibles en un principio.
5) Una vez que comienza una operación, el proceso continúa sin interrupción.

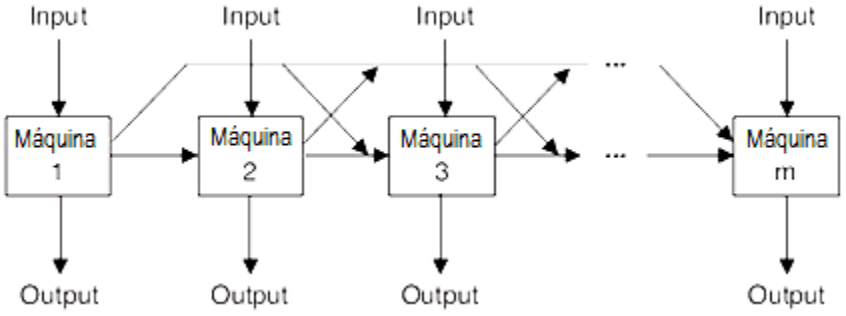


Figura 3.12 **Modelo general de flow shop** (Tomado de Baker, 2009)

En la figura 3.13 se muestran 2 trabajos con 4 operaciones cada uno y los tiempos de procesamiento de cada operación. Se dispone de 4 máquinas para procesar estos trabajos. Esta figura nos muestra 3 diferentes programaciones de estos trabajos, en donde se puede observar que la tercera resulta más eficiente que las dos anteriores.

Job j	1	2
p_{1j}	1	4
p_{2j}	4	1
p_{3j}	4	1
p_{4j}	1	4

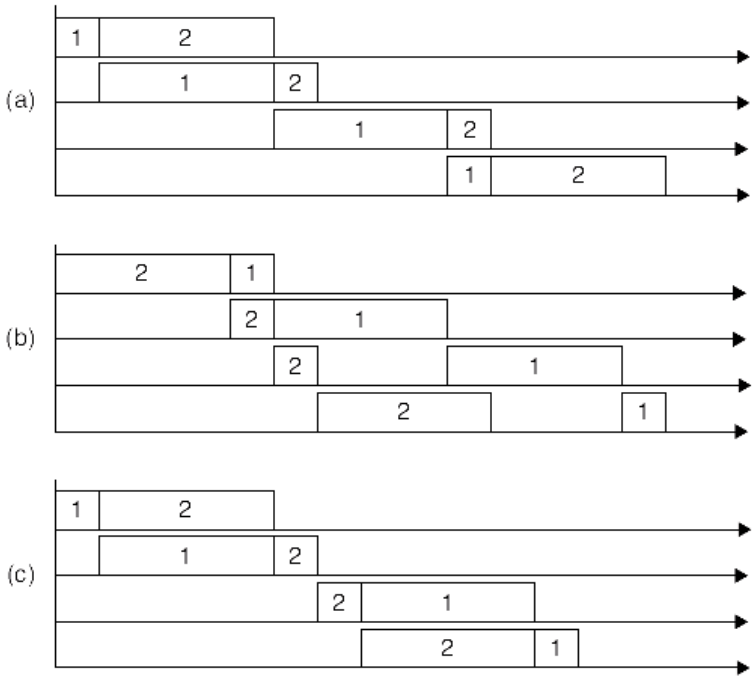


Figura 3.13 Programación de 2 trabajos con 4 operaciones (Tomado de Baker, 2009)

Una diferencia con el modelo de una única máquina es que puede ser ventajoso el uso del ocio en las máquinas. En el modelo de una única máquina existe una relación uno-a-uno entre la secuencia de trabajos y una permutación de los números *1, 2, …, n*. Para encontrar la secuencia óptima es necesario examinar las *n!* permutaciones o secuencias posibles. Similarmente, en el problema flow shop existen *n!* posibles secuencias para cada máquina y, por lo tanto, son un total de $(n!)^m$ diferentes programaciones de actividades.

El análisis del caso estocástico de flow shop está restringido a la minimización del makespan únicamente. Para la resolución de estos problemas se utilizan heurísticas (como búsqueda de vecindad).

Un trabajo en un flow shop puede ser un conjunto de sublotes, de tal manera que sus operaciones se puedan traslapar. El tamaño de lote ya está predeterminado por el consumidor o por la producción.

3.1.4. Problema de trabajos no unidireccionales *(job shop scheduling)*

El job shop se diferencia del flow shop en un aspecto: no es unidireccional. Existen *m* máquinas y *n* trabajos a programar. Cada trabajo consiste en varias operaciones con el mismo sistema lineal de precedencia como en el modelo flow shop. Pero a diferencia del flow shop, no existe una máquina inicial que realice únicamente la primera operación de un trabajo, como tampoco existe una máquina terminal que haga la última operación.

En flow shop, la máquina *k* realiza la *k-ésima* operación de cualquier trabajo y no hay necesidad de distinguir entre el número de operación y el número de máquina. Por el contrario, en el job shop es necesario especificar una operación con tres índices (*i, j, k) (*la operación *j* del trabajo *i* necesita la máquina *k)* como se muestra en la figura 3.14.

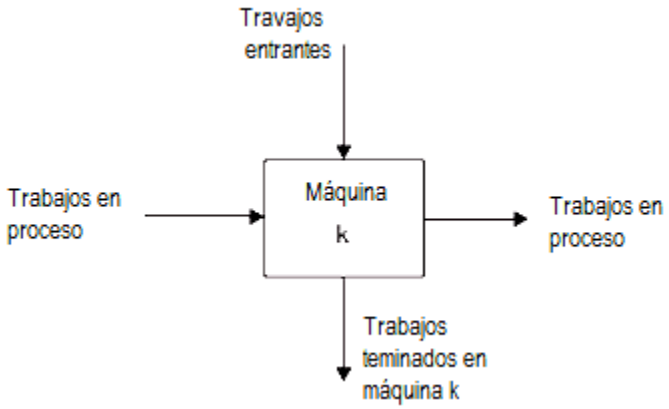


Figura 3.14 **Flujo de trabajo en un job shop** (Tomado de Baker, 2009)

Normalmente la representación gráfica de un job shop se hace mediante diagramas de Gantt.

Durante este capítulo se explicó la teoría de programación de actividades o *scheduling*. En el modelo matemático que se presenta en el siguiente capítulo, se utiliza esta teoría como también la nomenclatura explicada anteriormente.

4. MODELO MATEMÁTICO PARA MINIMIZACIÓN DE COSTOS Y RETRASOS EN VUELOS COMERCIALES

A continuación se presenta el modelo matemático propuesto por Soomer (SOOMER, 2007). Se tomará este modelo como base para la programación del modelo matemático para la minimización de costos y retrasos en una pista para resolver el objetivo de éste trabajo.

Posteriormente se reescribirá el modelo matemático describiendo los puntos que se tomarán del modelo de Soomer (2007) y las especificaciones extras que se tendrán a este problema en específico.

4.1. Modelo de programación entera mixta propuesto por Soomer

Se formulará el problema como un programa entero mixto (MIP), el cual se utiliza para determinar una secuencia de aterrizajes y los tiempos de aterrizaje para un determinado número de vuelos en una pista. Se asume que los vuelos ya están asignados a una pista. Si es que existiera más de una pista, se asume que éstas son independientes. Durante el período de planeación la pista considerada será analizada únicamente para llegadas.

Se asume para cada vuelo un intervalo de tiempo en el que se puede llevar a cabo el aterrizaje. Para vuelos en ruta, este intervalo se determinará por la distancia que falta, la velocidad máxima y mínima y la cantidad de combustible que se tenga. Para los vuelos que todavía no despegan, el intervalo estará determinado por el tiempo mínimo de vuelo y el máximo retraso posible de despegue. Para establecer una equidad, se usará el mismo retraso máximo en despegue para todos los vuelos.

4.1.1. Costos de las aerolíneas y equidad

El costo en el que incurre una aerolínea para los vuelos que llegan, depende del tiempo en el que aterrizan. Cada vuelo tiene diferentes características en cuanto a número de pasajeros provenientes de otras conexiones, el tiempo de salida del siguiente vuelo con la misma tripulación, etc. Como estos factores varían de vuelo a vuelo, cada aerolínea determinará su propia función de costos para cada vuelo. Cada una de estas funciones asocia el tiempo de aterrizaje del vuelo a un costo.

Se intentará permitir que las aerolíneas tengan mucha flexibilidad para determinar sus funciones de costos. Sin embargo, se buscará que estas funciones permitan determinar una programación de aterrizajes equitativa, es decir, que no exista un sesgo en la programación favoreciendo a alguna o algunas aerolíneas.

Primero se necesita modelar las funciones de costos como convexas y lineales por partes, y tener un costo mínimo de cero en el tiempo de aterrizaje deseado. Esto implica la no negatividad de los costos y que el incremento de costo esté asociado con la desviación de horarios en los aterrizajes. Las funciones convexas son en general realistas, dado que un retraso más grande representa un incremento en costos por unidad de tiempo.

Las aerolíneas tendrán, dentro de la modelación, la flexibilidad al determinar la localización de los puntos de quiebre y las pendientes de las líneas de la función de costos. Los puntos de quiebre deben representar los tiempos en los que los costos darán un salto o crecerán más rápidamente. Los puntos de quiebre serán, por lo tanto, un umbral o cota máxima del tiempo *t* en ese intervalo La función de costos se muestra en la figura 4.1.

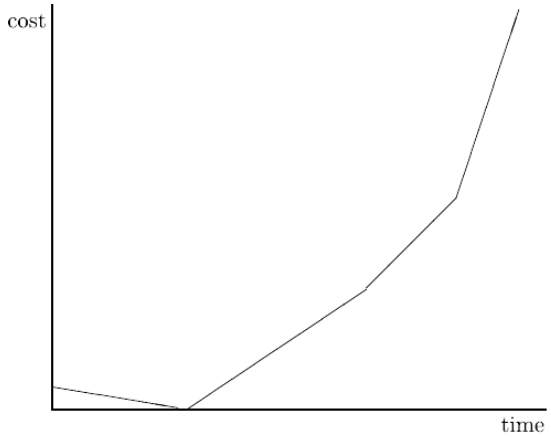


Figura 4.1 **Función de costos convexa lineal por partes** (Tomado de M. J. Soomer(2007))

Los costos pueden variar mucho entre las aerolíneas. Si se minimiza la suma de todas las funciones de costos, se favorecerá a las aerolíneas cuyos costos por retrasos sean mayores (las aerolíneas con mayor porcentaje de retrasos y por lo tanto de costos). Es por esto que se asignará un promedio del costo, por unidad de tiempo, igual para todos los vuelos. Las funciones de costo individuales se multiplicarán por el factor de escala (o ponderación de presencia en aterrizajes en la secuencia total) de cada aerolínea. Una

programación óptima no está definida como la programación con el mínimo costo total, sino como la programación con el mínimo costo total ponderado.

Sea la aerolínea *j* con N_j vuelos a aterrizar, los cuales se pueden representar con una función convexa lineal por partes $k_1(t), k_2(t), \ldots, k_{Nj}(t)$. Sean E_i y L_i el tiempo más temprano posible de aterrizaje y el tiempo más tardío posible, respectivamente, para el vuelo *i*. Por lo tanto $k_i(t)$ está definido en el intervalo $[E_i, L_i]$.

Para obtener equidad, las funciones de costo serán escaladas o ponderadas por:

$$f_i(t) = \alpha_j k_i(t) \; ; \quad i = 1, \ldots, N$$

Los factores de escala α_j se determinan de diferente manera por cada aerolínea. Esto asegura que la razón entre costos de sus vuelos se respete en las funciones de costos escaladas.

Definimos α_j a partir de:

$$\frac{1}{N_j} \sum_{i=1}^{N_j} \frac{\int_{E_i}^{L_i} \alpha_j k_i(t) dt}{(L_i - E_i)^p} = 1$$

Y despejando se obtiene:

$$\alpha_j = N_j \left(\sum_{i=1}^{N_j} \frac{\int_{E_i}^{L_j} k_i(t) dt}{(L_i - E_i)^p} \right)^{-1}$$

donde *p* es un parámetro para minimizar el efecto de diferencias en el tamaño de los intervalos de aterrizajes. Es preferible usar $p \geq 2$ (Soomer 2007).

Si se consideran dos aerolíneas con únicamente un vuelo y funciones de costos idénticas $k_1(t) = k_2(t) = t$, los posibles intervalos de aterrizajes para los vuelos *1* y *2* son $[0, T_1]$ y $[0, T_2]$ respectivamente con $T_2 > T_1$.

Por lo tanto:

$$\alpha_1 = 2T_1^{(p-2)}$$

$$\alpha_2 = 2T_2^{(p-2)}$$

y las funciones escaladas (ponderadas) son:

$$f_1 = 2T_1^{(p-2)}\, t$$

$$f_2 = 2T_2^{(p-2)}\, t$$

Si $p < 2$ entonces $f_1(t) \geq f_2(t)$ para $0 \leq t \leq T_1$. Esto no se considera justo debido a que el vuelo *2* tiene un intervalo más grande y en consecuencia más tiempos posibles de aterrizaje. Si ninguno de los dos vuelos puede aterrizar antes del tiempo T_1, el vuelo *2* puede aterrizar entre T_1 y T_2, mientras que el vuelo *1* no puede. Si se toma $p < 2$ resulta siempre más económico que el vuelo *1* aterrice antes que el vuelo *2* (aunque los dos pueden programarse antes de T_1), y los costos de la segunda aerolínea $(k_2(t_2^*))$ serán mayores que los costos de la primera aerolínea $(k_1(t_1^*))$. Por lo tanto, cuando $p < 2$ la primera aerolínea aterrizará siempre antes con un menor costo, mientras que la segunda aerolínea tiene mucha más flexibilidad.

Si $p = 2$ entonces $f_1(t) = f_2(t)$ para $0 \leq t \leq T_1$.y se puede considerar justo. Otra opción sería recompensar al vuelo *2* por tener mayor flexibilidad (con un intervalo más grande) escogiendo $p > 2$.

Sin embargo, el método deja aún espacio para que las aerolíneas incrementen las funciones de costos de algunos vuelos artificialmente para que estos vuelos tengan una mayor prioridad con respecto a otros. De hecho, las funciones de costos escaladas se pueden considerar como puntos de prioridad, los cuales se pueden dividir entre los vuelos de una aerolínea de acuerdo a sus preferencias. Esto concuerda con el objetivo de tener en consideración las preferencias de las aerolíneas tanto como sea posible.

En la mayoría de los aeropuertos, existen horas pico durante el día. Si las α_j*'s* se determinan por un día o más, una aerolínea que tenga programados vuelos en horas pico y en horas no pico, tiene una ventaja sobre aerolíneas con vuelos programados únicamente en horas pico. La primera aerolínea puede asignar grandes costos asociados a los retrasos de los vuelos durante horas pico comparado con sus vuelos en horas no pico. El promedio del costo por retraso ponderado por unidad de tiempo, será mucho mayor para la primera

aerolínea comparado con el costo de la segunda para los vuelos en horas pico. Por lo tanto, una minimización de los costos totales ponderados provocará menos retrasos para los vuelos de la primera aerolínea en horas pico. Los retrasos en las colas de una pista son mucho más frecuentes en horas pico, debido a que en estos períodos de tiempo, la demanda se acerca y, en ocasiones excede, la capacidad del aeropuerto. Fuera de las horas pico, existen menos retrasos, por lo que los vuelos de la primer aerolínea experimentan menos retrasos. Por esta razón es recomendable determinar distintos factores de ponderación para los diferentes tipos o clases de intervalos de tiempo (horas pico, horas no pico).

4.1.2. Formulación de programación entera mixta (MIP)

Sea

N: número de vuelos a aterrizar a programar

E_i: tiempo más temprano posible para aterrizar del vuelo *i* *para* $i = 1, \ldots, N$

L_i: tiempo más tardío posible para aterrizar del vuelo *i* *para* $i = 1, \ldots, N$

S_{ij}: tiempo requerido de separación cuando el vuelo *i* aterriza antes que el vuelo *j*

$i, j = 1, \ldots, N; \quad i \neq j$

Las variables de decisión:

t_i: tiempo de aterrizaje para el vuelo *i*. $i = 1, \ldots, N$

$$\delta_{ij} = \begin{cases} 1 & si\ el\ vuelo\ i\ aterriza\ antes\ que\ el\ vuelo\ j \\ 0 & c.o.c. \end{cases}$$

$i, j = 1, \ldots, N; \quad i \neq j$

Las siguientes son las restricciones básicas para asignar una programación de aterrizajes factible y segura:

$$E_i \leq t_i \leq L_i ; \qquad i = 1, \ldots, N; \tag{24}$$

Esta restricción define los posibles tiempos de aterrizaje.

Considerando pares de vuelo *(i, j)*:

$$\delta_{ij} + \delta_{ji} = 1 ; \qquad i,j, = 1, \ldots, N - 1; \qquad j > i \tag{25}$$

Este conjunto de restricciones declara que debe de ser cierto alguno de los siguientes enunciados:

i. El vuelo *i* tiene que aterrizar antes que el vuelo *j* ($\delta_{ij} = 1$) ó

ii. El vuelo *j* tiene que aterrizar antes que el vuelo i ($\delta_{ji} = 1$)

Utilizando la separación de tiempo y posibles intervalos de tiempo para aterrizajes entre dos vuelos, se definen tres conjuntos de pares de vuelos:

U: el conjunto de pares de vuelos *(i, j)* en los cuales existe incertidumbre acerca de si el vuelo *i* aterrizará antes que el vuelo *j*.

V: el conjunto de pares de vuelos *(i, j)* en los cuales existe certeza de que el vuelo *i* aterrizará antes que el vuelo *j*, pero para los cuales la separación no se satisface automáticamente.

W: el conjunto de pares de vuelos *(i, j)* en los cuales existe certeza de que el vuelo *i* aterrizará antes que el vuelo *j*, y la separación se satisface automáticamente.

Formalmente:

$$U = \{(i,j) \mid E_j < E_i < L_j \text{ ó } E_j < L_i < L_j \text{ ó } E_i < L_j < L_i \text{ ó}$$

$$E_i < L_j < L_i; \quad i, j = 1, \ldots, N; \quad i \neq j\}$$

$$V = \{(i,j) \mid L_i < E_j \text{ y } L_i + S_{ij} > E_j; \quad i, j = 1, \ldots, N; \quad i \neq j\}$$

$$W = \{(i,j) \mid L_i < E_j \text{ y } L_i + S_{ij} \leq E_j; \quad i, j = 1, \ldots, N; \quad i \neq j\}$$

Para los elementos del conjunto *V* y *W* se sabe el orden de los vuelos expresado por la siguiente restricción:

$$\delta_{ij} = 1 \qquad \forall\, (i,j) \in W \cup V \tag{26}$$

Sin embargo, para los vuelos del conjunto *V* se necesita asegurarse la debida separación:

$$t_j \geq t_i + S_{ij} \quad \forall (i,j) \in V \tag{27}$$

Finalmente se necesita también asegurar la separación si el orden de los vuelos es incierto (elementos de *U*):

$$t_j \geq t_i + S_{ij}\delta_{ij} - (L_i - E_j)\delta_{ji} \quad \forall (i,j) \in U \tag{28}$$

Existen dos opciones:

i. El vuelo *j* no puede aterrizar antes del mínimo tiempo de separación entre *i* y *j* después del tiempo de aterrizaje del vuelo *i*:

   $t_j \geq t_i + S_{ij}$

   en el caso que el vuelo *i* aterriza antes que el vuelo *j* ($\delta_{ij} = 1$), esta restricción se vuelve redundante dado que:

   $t_j \geq t_i - (L_i - E_j)$

   Siempre se satisface (porque $t_i \leq L_i$ y $t_j \geq E_j$) en el caso que:

ii. el vuelo *j* aterrice antes que el vuelo *i* ($\delta_{ji} = 1$).

Como se explicó anteriormente, cada aerolínea provee una función de costos convexa y lineal por partes para cada uno de sus vuelos, la cual estará ponderada para obtener equidad entre las aerolíneas.

La función escalada lineal por partes $f_i(t)$ para cada vuelo *i*, puede expresarse como un conjunto de funciones lineales conectadas con un número de intervalos:

$$f_i(t) = \begin{cases} A_{i0}t + B_{i0} & 0 \leq t \leq t_{i1} \\ A_{i1}t + B_{i1} & t_{i1} \leq t \leq t_{i2} \\ \vdots & \vdots \\ A_{ik_i}t + B_{ik_i} & t_{ik_i} \leq t \end{cases}$$

Donde:

K_i: número de puntos de quiebre de $f_i(t)$ $i = 1, \ldots, N.$

Ahora:

$$f_i(t) = \max_{k=0,\ldots,K_i} \{A_{ik}t + B_{ik}\}$$

Por la continuidad de $f_i(t)$ esto implica:

$$f_i(t_{i,k}) = A_{i,k-1}t_{i,k} + B_{i,k-1} = A_{i,k}t_{i,k} + B_{i,k} \qquad k = 1, \dots, K_i$$

y la convexidad de $f_i(t)$ implica:

$$A_{io} < A_{i1} < \cdots < A_{iK_i}$$

El objetivo del modelo MIP será minimizar la suma de estas funciones de costos. Sin embargo, estas funciones no son lineales en las variables de decisión actuales t_i, por lo tanto se necesita introducir nuevas variables de decisión c_i y así poder obtener un modelo lineal:

c_i: costo de aterrizaje del vuelo i; $\qquad i = 1, \dots, N$

c_i representa la función de costo $f_i(t_i)$. Para asegurar esto, se necesita introducir nuevas restricciones:

$$c_i \geq A_{i,k}t_i + B_{i,k}; \qquad i = 1, \dots, N; \quad k = 0, \dots, K_i \tag{29}$$

Esto asegura para el vuelo i que:

$$c_i \geq \max_{k=0,\dots,K_i} \{A_{ik}t_i + B_{ik}\} = f_i(t_i)$$

La función objetivo es:

$$z = \min \sum_{i=1}^{N} c_i \tag{30}$$

Las ecuaciones (29) y (30) aseguran que $c_i = f_i(t_i)$:

Supóngase que $t_1^*, t_2^*, \dots, t_n^*$; $\quad c_1^*, c_2^*, \dots, c_n^*$ es la solución óptima para el modelo MIP, donde $c_i^* > f_i(t_i^*)$ para algunos i. Sea $c'_i = f_i(t_i^*)$. Si se reemplaza c_i^* por c'_i la función objetivo decrece en $c_i^* - f_i(t_i^*)$ sin violar la restricción (6):

$c'_i = f_i(t_i^*) = \max_{k'=0,\dots,K_i}\{A_{ik'}t_i^* + B_{ik'} \geq A_{ik}t_i^* + B_{ik}\}$ para $k = 0, \dots, K_i$.

Por lo tanto $c_i^* > f_i(t_i^*)$ no puede ser óptimo (y $c'_i = f_i(t_i^*)$ sí lo es).

4.2. Modelo matemático de programación entera mixta (MIP) para la minimización de costos y retrasos en una pista.

A continuación se presentará la formulación del problema a tratar en este trabajo en específico. Se utilizará nomenclatura, tanto del modelo de Soomer (2007), como de la teoría de *scheduling* descrita anteriormente.

El problema consiste en determinar una secuencia de despegues y aterrizajes, tal que minimice los costos y retrasos en una pista a lo largo de un día.

No se consideró la distancia entre aviones como en el modelo anterior, dada la complejidad que este aspecto implicaba. Es por eso que se omiten las variables referentes a la distancia *(S_{ij}).*

Las ventanas de tiempo sí serán consideradas. Esto significa que todos los vuelos tendrán un límite inferior y un límite superior para poder aterrizar o despegar. Si la hora de despegue o aterrizaje se sale de este rango, dicho vuelo incurrirá en un costo asociado a dicho retraso, ya sea este positivo o negativo, es decir, si el vuelo aterrizó/despegó antes de tiempo o después de tiempo.

Por otro lado, se toma en cuenta tanto despegues como aterrizajes y no únicamente aterrizajes, como en el modelo anterior.

Las variables del problema son las siguientes:

C_i: tiempo (en minutos) de terminación del aterrizaje/despegue para el vuelo *i*.

$i = 1, \ldots, N$

$$\delta_{ij} = \begin{cases} 1 & si\ el\ vuelo\ i\ aterriza\ antes\ que\ el\ vuelo\ j \\ 0 & c.o.c. \end{cases}$$

$i, j = 1, \ldots, N; \quad i \neq j$

Los datos del problema son los siguientes:

N: número de vuelos a aterrizar o despegar que se programarán

E_i: tiempo más temprano posible para aterrizar del vuelo *i* *para* $i = 1, \ldots, N$

L_i*:* tiempo más tardío posible para aterrizar del vuelo *i* *para* $i = 1, \ldots, N$

p_i*:* tiempo de procesamiento, es decir, el tiempo requerido para el aterrizaje o despegue del vuelo *i* *para* $i = 1, \ldots, N$

R_i*:* Retraso del vuelo *i*.

S: el conjunto de vuelos con un retraso positivo. Cuando un vuelo aterriza/despega después de L_i.

T: el conjunto de vuelos con retraso negativo. Cuando un vuelo aterriza/despega antes de E_i.

V: el conjunto de vuelos sin retraso

U: el conjunto de pares de vuelos *(i, j)* en los cuales existe incertidumbre acerca de si el vuelo *i* aterrizará/despegará antes que el vuelo *j*.

W: el conjunto de pares de vuelos *(i, j)* en los cuales existe certeza de que el vuelo *i* aterrizará/despegará antes que el vuelo *j*.

$$U = \{(i,j) \mid E_j < E_i < L_j \text{ ó } E_j < L_i < L_j \text{ ó } E_i < L_j < L_i \text{ ó } E_i < L_j < L_i; \quad i,j = 1, \ldots, N; \quad i \neq j\}$$

$$W = \{(i,j) \mid L_i < E_j \text{ y } L_i \leq E_j; \quad i,j = 1, \ldots, N; \quad i \neq j\}$$

La función escalada lineal por partes $costo_i(R)$ para cada vuelo *i* por intervalos, es la siguiente:

$$costo_i(R_i) = \begin{cases} A_{i0}|R_i| + B_{i0} & 0 \leq |R_i| \leq R_{i1} \\ A_{i1}|R_i| + B_{i1} & R_{i1} \leq |R_i| \leq R_{i2} \\ \vdots & \vdots \\ A_{ik_i}|R_i| + B_{ik_i} & R_{ik_i} \leq |R_i| \end{cases}$$

K_i: número de puntos de quiebre de $costo_i(R)$ $\quad i = 1, \ldots, N.$

Por lo tanto, la formulación es la siguiente:

$$MIN \sum_{i=1}^{N} costo_i \tag{31}$$

s.a:

$$\delta_{ij} + \delta_{ji} = 1\ ; \qquad i, j = 1, \ldots, N \qquad i \neq j \tag{32}$$

$$\delta_{ij} = 1; \qquad \forall\ (i,j)\ \in W \tag{33}$$

$$C_i \geq (C_j + p_i)\delta_{ji}; \qquad i, j = 1, \ldots, N; \quad j \neq i; \quad \forall\ (i,j)\ \in U \tag{34}$$

$$C_N \leq 1440; \tag{35}$$

$$R_i = C_i - L_i; \qquad \forall i\ \in S \tag{36}$$

$$R_i = E_i - C_i; \qquad \forall i\ \in T \tag{37}$$

$$R_i = 0; \qquad \forall i\ \in V \tag{38}$$

$$costo_i \geq A_{i,k}|R_i| + B_{i,k}; \qquad i, j = 1, \ldots, N; \quad k = 0, \ldots, K_i \tag{39}$$

$$\delta_{ij}\ \in \{0,1\}; \tag{40}$$

$$C_i \geq 0; \tag{41}$$

A continuación se describirá cada una de las restricciones del problema:

- (31) La función objetivo será minimizar la suma de costos por los retrasos de cada uno de los vuelos. La función se conforma por varios intervalos lineales, los cuales dependen del tamaño (en valor absoluto) del retraso de cada vuelo.
- (32) Esta restricción obliga a que, si el vuelo *i* aterriza/despega antes que el vuelo *j*, entonces el vuelo *j* no puede aterrizar/despegar antes que el vuelo *i* (y viceversa).
- (33) Para los pares de vuelos, en los que se sabe con certeza que el vuelo *i* aterrizará/despegará antes que el vuelo *j*, se fija su valor.

- (34) Para los pares de vuelos, en los que no se sabe con certeza que el vuelo *i* aterrizará/despegará antes que el vuelo *j*, se tiene que asegurar que no exista más de un aterrizaje/despegue a la vez. Es por esto que el tiempo de terminación de un aterrizaje/despegue (*i*) tiene que ser mayor al tiempo de terminación del siguiente más el tiempo de procesamiento del vuelo *i*. Esto se cumple únicamente si el vuelo *j* aterriza/despega antes que el vuelo *i*. De lo contrario, esta restricción indica únicamente que el tiempo de terminación para cada vuelo *i* tiene que ser positivo (volviéndose redundante debido a la regla de la no negatividad).
- (35) El tiempo de terminación del aterrizaje/despegue del último vuelo debe ser menor a 1440 debido a que éste es el último minuto del día.
- (36) (37) (38) Estas restricciones calculan los retrasos de cada vuelo.
- (39) Esta restricción calcula el costo del retraso de cada vuelo *i*, evaluándolo en la función de costos.
- (41) Determina que la variable δ_{ij} sea binaria.
- (42) Determina que la variable C_i debe de ser positiva.

4.2.1. Complejidad computacional de vuelos en una pista

La complejidad computacional del problema de vuelos (aterrizajes y despegues) en una pista se reduce al problema de *scheduling* en una única máquina. Como ya se mencionó, éste problema es combinatorio y, por lo tanto, NP-duro.

Sea *n* el número de trabajos a realizar en una máquina. La secuencia de estos trabajos tendrá, por lo tanto, una complejidad de *n!* (por ser combinatorio).

La complejidad computacional de vuelos en una pista será por lo tanto:

Sea A el conjunto de los n vuelos a programar en una pista,
$\exists\, W \in A \; donde \quad W: k \; trabajos \; tal \; que \; r_i \geq$
$C_j (como \; se \; mencionó \; anteriormente)$

$con \quad k \in \mathbb{R} \; y \; k \leq n; \; i, j = 1, \dots, N; i \neq j$

Por lo, tanto el conjunto de vuelos que se podrá reprogramar en la secuencia, es aquel en el cual existe incertidumbre acerca de su despegue/aterrizaje con respecto a otros vuelos, es decir $A \backslash W$.

∴ la complejidad computacional de vuelos en una pista será: $(n-k)!$.

Este número sigue siendo demasiado grande para poder resolver el problema de manera exacta (ver apéndice A). Es por eso que se necesita programar un método heurístico para poder encontrar una solución a éste. En el capítulo 5 se encuentra la descripción de la heurística utilizada basada en el modelo matemático explicado anteriormente.

5. SIMULACIÓN DEL SISTEMA DE DESPEGUES Y ATERRIZAJES EN UNA PISTA A LO LARGO DE UN DÍA

Dado que no fue posible adquirir suficientes datos reales para que fuera una muestra representativa, se creará una simulación del sistema para poder compararla con el algoritmo presentado en el siguiente capítulo y así poder demostrar una mejora en los costos y retrasos de los aterrizajes y despegues.

Esta simulación del problema anteriormente descrito se utilizará para ser la solución de arranque o solución inicial factible en la programación del método heurístico *Búsqueda Tabú* explicado en el siguiente capítulo. Esta simulación está basada en los datos y restricciones propuestas en el capítulo anterior, donde se presentó el modelo matemático para la resolución de dicho problema.

Es importante mencionar las limitaciones de esta simulación. Durante este capítulo se utilizará únicamente una tasa de llegada de vuelos, tanto aterrizajes como despegues, a la cola para utilizar la pista. En la realidad, la tasa de llegadas en cualquier aeropuerto cambia a lo largo del día. No es la misma tasa de llegada en las primeras horas del día que a las horas pico o que en las últimas horas del día. Sin embargo, dado que no fue posible medir estos cambios en dichas tasas, se tomará el promedio entre todas utilizando una tasa fija durante todo el día.

Es importante señalar que no se tomarán en cuenta situaciones en el cambio de la programación de los vuelos, provocados por cambios climáticos o emergencias atípicas o poco probables.

5.1. Formulación del problema

Los datos necesarios que se obtendrán mediante la simulación son los siguientes:

- Tipo de vuelo, es decir, si el vuelo *i* representa un despegue o un aterrizaje
- Tiempo entre llegadas de cada uno de los vuelos
- Tiempo en que comienza cada vuelo su despegue o aterrizaje
- Tiempo en que termina cada vuelo su despegue o aterrizaje
- Retraso de cada vuelo

- Costo por el retraso de cada vuelo
- La secuencia resultante
- Resultado de función de costos totales en la secuencia

5.2. Conceptualización del sistema

Tanto para la simulación del sistema, como para la programación de la heurística, los datos y nomenclatura estarán basados en la teoría de *scheduling*.

Por lo tanto la información conocida o datos de entrada (*inputs*) en el problema son:

- Tiempo de procesamiento *(p_j) (processing time)*: tiempo requerido para el procesamiento del trabajo *j*, que en este caso será el tiempo del aterrizaje o despegue.

- Fecha de comienzo *(r_j) (release date)*: el tiempo en el que el trabajo *j* está disponible para comenzar a procesarse. En este caso r_j será el momento en que el avión llega a formarse a la cola para poder aterrizar (si es que ya existe una cola) o el momento en el que el avión ya está formado y listo para despegar.

- Fecha de entrega *(d_j) (due date)*: el tiempo en el que el trabajo *j* debe ser terminado. En este caso particular, d_j será el tiempo en el cual el vuelo debe aterrizar o despegar.

La información obtenida de la programación de actividades *(outputs)* es:

- Tiempo de terminación *(C_j) (completion time)*: tiempo en el cual el procesamiento del trabajo *j* se termina, es decir, el tiempo en el que cada vuelo termina su aterrizaje o despegue.

- Retraso *(R_j) (lateness)*: la cantidad de tiempo en la que el tiempo de terminación de trabajo *j* excede o se anticipa a su fecha de entrega, es decir, que tan retrasado o adelantado aterrizó o despegó el vuelo *j*.

$$R_j = C_j - d_j$$

5.3. Recolección de información y datos

A pesar de que los aviones que aterrizan y despegan no están formados físicamente en la misma cola, se puede decir que es una misma cola ya que utilizarán la misma pista y aunque no se vea la formación de la secuencia de los aviones, en realidad si están formados "virtualmente" con el sistema FIFO. Si un avión que tiene que despegar llegó a formarse antes que uno que va a aterrizar, el segundo tiene que esperar a que el primero complete su despegue.

Las llegadas de los aviones siguen una distribución Poisson con media un avión cada 2.28 minutos ($\lambda = (1/2.28)$ avión por minuto)

A pesar de que el tiempo de aterrizaje (o de procesamiento) es también estocástico, se tomará de manera determinístico debido a que no varía mucho (un aterrizaje cada dos minutos). Se tomará su valor esperado $E[p_j] = 2$. Lo mismo sucede con el tiempo de los despegues, se tomará su valor esperado $E[p_i] = 2.57$.

La fecha de entrega (d_j) *(due date)* está compuesta por un intervalo $[E_j, L_j]$ donde:

- E_j: tiempo más temprano posible para aterrizar o despegar del vuelo j; $j=1, \dots, N$
- L_j: tiempo más tardío posible para aterrizar o despegar del vuelo j; $j=1, \dots, N$

Los retrasos estarán asociados a una función de costos cóncava y lineal por partes, donde se tendrá un costo de cero en el intervalo $[E_j, L_j]$.

Es por eso que la función a minimizar está compuesta por la suma de la multiplicación de cada retraso por su costo asociado (función de costos):

$$MIN\ costo * |R| = \sum_{j=1}^{N} costo_j * |R_j|$$

Con los datos obtenidos del capítulo 1 acerca de los costos, la función de costos en USD es la siguiente:

$$f(x) = \begin{cases} 0 & E_j \leq |R_j| \leq L_j \\ \frac{200}{3}|R_j| & 0 < |R_j| \leq 60 \\ \frac{250}{3}|R_j| & 60 < |R_j| \leq 120 \\ \frac{300}{3}|R_j| & 120 < |R_j| \leq 180 \\ \vdots & \vdots \end{cases}$$

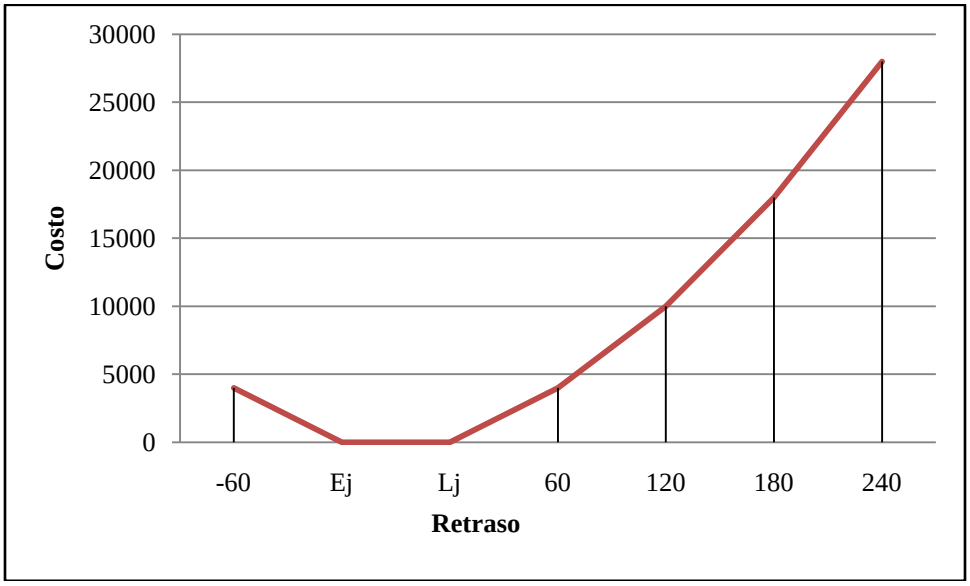


Figura 5.1. **Función de costos** (Basado en M. J. Soomer(2007))

La simulación tendrá una *cola* con llegada Poisson con media $\lambda = (1/2.28)$. El tipo de cola será FIFO *(first in first out)*, lo que significa que el primer avión que llegue a formarse a la cola será el primero que aterrice o despegue (*PEPS: primero en llegar primero en salir).*

Por lo anteriormente descrito, se puede ver que el análisis de este problema en particular está descrito por el problema de *scheduling* de una única máquina. El problema tiene dos tipos de trabajos: *m* aterrizajes y *n* despegues. Sin embargo, como se pudo ver en el capítulo 1 se puede considerar que $m \approx n$, es decir que existe aproximadamente la misma cantidad de aterrizajes y despegues en un día.

El análisis se hará únicamente para una pista ya que se tomará el supuesto de que las pistas son independientes y por lo tanto cada una de ellas tendrá su propia programación de secuencia.

La realización de la simulación será por lo tanto únicamente una cola con llegadas Poissson $\lambda = (1/2.28)$ de aviones a despegar o aterrizar y un solo servidor que será la pista con tiempo de procesamiento de $E[p_j] = 2$ para los aterrizajes y $E[p_i] = 2.57$ para los despegues.

Por lo tanto se tiene:

Tabla 5.1 tiempos de procesamiento de despegues y aterrizajes

Job: aterrizaje	1	2	…	M
p_j	2	2	…	2
Job: despegue	1	2	…	N
p_i	2.57	2.57	…	2.57

Donde $m \approx n$ y todos los tiempos de procesamiento de aterrizajes son iguales entre ellos así como los tiempos de cada uno de los despegues.

Durante esta simulación no se tendrán datos acerca de la cola (cola máxima, cola promedio) y del tiempo de espera en ella (tiempo de espera total, tiempo de espera promedio), ya que para este análisis en particular no son relevantes. Al contrario, si a un vuelo en particular se tiene que esperar más tiempo en la cola porque hay otros más que vienen muy retrasados será mejor, contrariamente a lo que comúnmente se busca en un análisis en la teoría de colas.

5.4. Generación de datos

La simulación se realizó mediante la programación en C++. A continuación se presenta la generación de los datos para dicha simulación. La programación completa se presenta en el anexo A.

Para simular la llegada de los eventos, se sabe que la probabilidad de encontrar x eventos en el tiempo t se distribuye Poisson cuya función de distribución es:

$$f(x) = \frac{e^{-\lambda}\lambda^x}{x!} \qquad donde\ x = 0, 1, 2, \ldots., n$$

El tiempo entre llegadas se distribuye Exponencial y su función de densidad es:

$$f(x) = \begin{cases} \lambda e^{-\lambda t} & x \geq 0 \\ 0 & c.o.c. \end{cases}$$

Y su función de distribución es:

$$F(x) = \begin{cases} 0 & x < 0 \\ 1 - e^{-\lambda t} & x \geq 0 \end{cases}$$

Para cuestiones de simulación, resulta mucho más fácil generar el tiempo entre eventos a encontrar el número de eventos en un tiempo determinado. Por lo tanto se utilizará la distribución Exponencial.

Aplicando el método de transformada inversa, los valores del tiempo entre eventos pueden simularse como:

$$t = \left(\frac{1}{\lambda}\right) \ln\left(\frac{1}{r}\right) \qquad donde\ r \sim U(0,1)$$

Para la generación del tipo de vuelo, es decir, si el vuelo es aterrizaje o despegue, se utilizó la probabilidad Binomial. Es decir:

$$tipo\ de\ vuelo \sim Bin(n,p) \qquad donde\ n = 630 \quad p = 0.5$$

La función de probabilidad Binomial es:

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x} \qquad donde\ x = 0, 1, 2, \dots, n$$

Ya que, como se mencionó anteriormente, existe aproximadamente el mismo número de aterrizajes y despegues, la probabilidad de aparición de cada uno de ellos es la misma (0.5).

Dentro del programa se tomó el éxito = 1 como aterrizajes y el fracaso = 0 como despegue para su diferenciación.

Para los retrasos se utilizó la distribución Triangular. Mediante la plática con expertos se determinaron los parámetros de dicha distribución:

- El tiempo que suelen adelantarse como máximo (en promedio) los vuelos al aterrizar o despegar es 60 minutos => a = -60.
- El tiempo que suelen atrasarse como máximo (en promedio) los vuelos al aterrizar o despegar es 240 minutos => b = 240.
- El retraso más común que experimentan los vuelos es de aproximadamente 0 minutos, es decir que la mayoría de los aviones no se retrasan (la moda) => c= 0.

Los datos atípicos no se tomaron en cuenta, como por ejemplo, el que un vuelo se retrase nueve horas. Sí puede darse ese caso pero es muy improbable. Es por eso que esos datos no son relevantes para este análisis.

La función de densidad Triangular es:

$$f(x) = \begin{cases} \dfrac{2(x-a)}{(b-a)(c-a)} & a \leq x \leq c \\ \dfrac{2(b-x)}{(b-a)(b-c)} & c \leq x \leq b \\ 0 & c.o.c. \end{cases}$$

Donde *a* y *b* son los extremos que toma la función y *c* es la moda.

Su gráfica es:

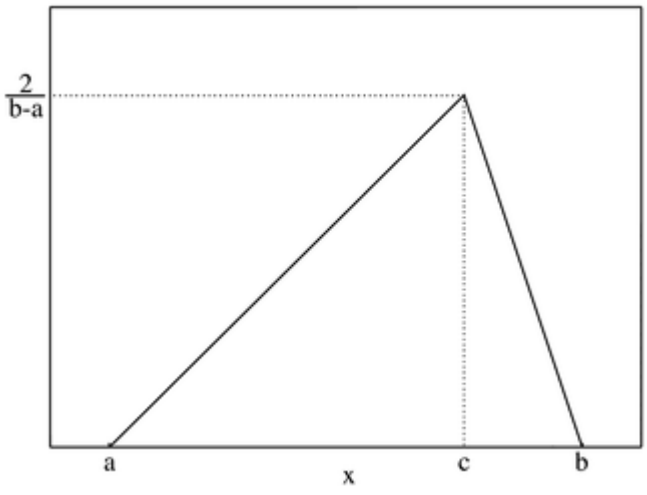


Figura 5.2. Función de densidad Triangular

Su función de distribución es:

$$f(x) = \begin{cases} 0 & x \le a \\ \dfrac{(x-a)^2}{(b-a)(c-a)} & a \le x \le c \\ 1 - \dfrac{(b-x)^2}{(b-a)(b-c)} & c \le x \le b \\ 1 & x \ge b \end{cases}$$

Y su gráfica es:

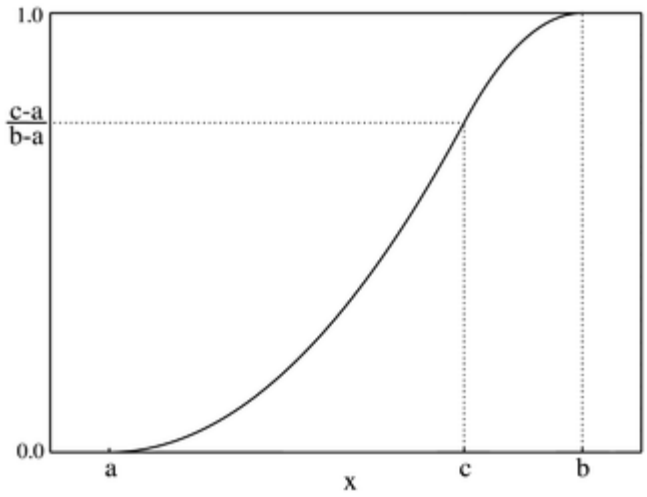


Figura 5.3. Función de distribución Triangular

Aplicando el método de transformada inversa se obtienen los siguientes valores:

$$f(r) = \begin{cases} a + \sqrt{(c-a)(b-a)r} & r \le (b-a)/(c-a) \\ c - \sqrt{(c-a)(c-b)(1-r)} & r \ge (b-a)/(c-a) \end{cases} \qquad donde\ r \sim U(0,1)$$

Dado que el modelo presentado sí representa la realidad, el paso siguiente fue programarlo en C++.

El código de la simulación se puede ver en el anexo A.

5.5. Número de corridas

Se puede obtener un tamaño de muestra o número de corridas por medio de la siguiente prueba de hipótesis:

Se tiene que probar $H_0 : \mu_i = \mu_j$, es decir, el promedio de retrasos para cada corrida sea aproximadamente igual

Contra la hipótesis alternativa de $H_A : \mu_i \neq \mu_j$ *para todo* $i \neq j$

Si $n_1 = n_2 = \ldots = n_k = n$

$$LSD = t_{\frac{\alpha}{2},N-k}\sqrt{2CM_E/n}$$

Por lo tanto:

$$n = \frac{2\left(t_{\frac{\alpha}{2},N-k}\right)^2 CM_E}{LSD^2}$$

(Véase apéndice B)

CM_E también se puede calcular como la varianza muestral:

$$CM_E = S^2$$

Y la fórmula de la varianza muestral es:

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Dando como resultado **3787.0503** y la desviación estándar es **61.539 minutos**.

Queremos in intervalo de confianza del 95%. Por lo tanto $\alpha=0.05$.

En cuanto a los grados de libertad, *k* es el número de tratamientos que se harán en el experimento. En este caso en particular no se hará más que un solo tratamiento, por lo tanto $k=1$. *N* se calcula multiplicando $n_0 * k$ donde n_0 es el número de réplicas por tratamiento, que en este caso son los 630 vuelos diarios. Por lo tanto $N=630 * 1 = 630$. Sin embargo las tablas para la distribución *t de Sudent* no tienen datos con una *n* tan grande. Es por eso que cuando se tiene una *n* tan grande se debe aproximar por medio de la distribución *Normal(0,1).*

$$\lim_{n\to\infty} t_n(x) = N_{(0,1)}(x)$$

El valor del estadístico que se obtiene en tablas es de 1.6 (ver anexo C).

La diferencia entre datos (retrasos) *LSD* es la que uno no quiere que se rebase de réplica a réplica. En este caso se sugirió una diferencia de 25 minutos, es decir, que si el promedio de retrasos en cada una de las réplicas no varía en $\mp$ 30 minutos, se tomarán como iguales estos como iguales. Se tomó este número 30, ya que el intervalo en el cual los vuelos no tendrán una penalización en costo relacionada a un retraso es de 30 minutos.

Por lo tanto, si sustituimos estos datos en la ecuación anterior obtenemos lo siguiente:

$$n = \frac{2 * 1.6^2 * 3586.1214}{30^2} = 20.40104$$

Este número se redondeó a 21 corridas.

5.6. Resultados de la simulación

A continuación se presentarán los resultados de la simulación (los resultados completos por vuelo se presentan en el anexo B):

Se obtuvo un total de 630 vuelos a lo largo de 1440 minutos o 24 horas. 317 de ellos fueron aterrizajes y 313 fueron despegues.

El retraso máximo fue de **223.242935 minutos**, es decir, 3.7207 horas. El retraso mínimo fue de **0 minutos**. La suma de retrasos fue de **43549.146 minutos** o 725.8191 horas y la suma de los costos por los retrasos en un día fue de **$3,965,367.42 USD**. El promedio de retrasos fue de **69.125628 minutos** o 1.152 horas y el promedio del costo por retrasos fue de **$6,294.234 USD**.

De los 630 vuelos, 177 aterrizaron o despegaron sin retraso, lo cual significa que se necesita reasignar 453 vuelos.

El modelo programado representa muy bien el sistema real, por lo tanto es un modelo válido.

No se hizo un análisis del tamaño de cola ni del tiempo de espera en ella, ya que en esta investigación en particular, no son relevantes esos datos.

6. PROGRAMACIÓN DEL MÉTODO HEURÍSTICO *BÚSQUEDA TABÚ* (BT)

Dada la complejidad computacional del problema a tratar, que es un problema combinatorio y por lo tanto NP-duro, como ya se mencionó (ver apéndice A), lo más conveniente es recurrir al uso de algún método heurístico.

En este capítulo se explicará el método heurístico utilizado, sus características, pasos del algoritmo, un ejemplo para un mejor entendimiento del problema en cuestión y los resultados obtenidos. La programación de este método está basado en el modelo matemático descrito en el capítulo 4 y en la teoría y nomenclatura de la programación de actividades descrita en el capítulo 3.

Para este tipo de problemas de programación de actividades o *scheduling* se recomienda el uso de dos metaheurísticas: *Búsqueda Tabú* y *Recocido Simulado*.

Las metaheurísticas Búsqueda Tabú (BT), como recocido simulado (RS), están basadas en la búsqueda en vecindades evitando óptimos locales.

En este trabajo se utilizará el método de Búsqueda Tabú. Éste intenta modelar el proceso de la memoria humana. La memoria registra las soluciones vistas, usando estructuras de datos simples, guardándolas en una lista tabú o prohibida durante cierto número de iteraciones. Esto permite diversificar la búsqueda de soluciones.

6.1. Búsqueda Tabú (BT)

BT opera bajo el supuesto de que es posible construir una vecindad para identificar soluciones adyacentes. Frecuentemente se usan permutaciones para definir vecindades en cierto tipo de problemas. Los valores de los movimientos, o cambios en el valor de la función objetivo, generalmente son la base para evaluar la calidad de un movimiento.

La clasificación de los movimientos tabú depende de la historia de la búsqueda; particularmente la frecuencia con que ciertos movimientos o componentes de la solución, llamados atributos, han participado en la generación de las soluciones pasadas. En este caso, un atributo de una permutación será la identidad de los vuelos que cambiaron de posición.

Las restricciones tabú no son inviolables. Estas condiciones se pueden retirar, si con el cambio se obtiene una mejor solución que la mejor encontrada hasta el momento. El tamaño de la lista tabú puede variar dependiendo del problema que se trate en específico.

Búsqueda tabú se basa en tres puntos principales:

1) El uso de estructuras de memoria basados en atributos diseñados para permitir criterios de evaluación e información de búsqueda histórica.
2) Un mecanismo asociado de control mediante el empleo de estructuras de memoria basado en el inter-juego entre las condiciones que restringen y liberan el proceso de búsqueda.
3) La incorporación de funciones de memoria de diferentes tipos, para implantar estrategias que refuercen la combinación de movimientos y las características de solución que históricamente se han encontrado buenas. Las estrategias de diversificación manejan la búsqueda dentro de nuevas regiones.

6.2. Descripción de los componentes de la Búsqueda Tabú

La función de memoria se utilizó de manera estática. El tamaño *t* de la lista tabú se fue variando desde $t = 3$ hasta $t = \sqrt{n}$ que en este caso $n = 630$, es decir el número de vuelos en un día. Por lo tanto $\sqrt{n} \approx 25$, encontrando mejores resultados para $t = 5$.

Las vecindades, en este caso en específico, eran aquellas secuencias resultantes de reasignar un vuelo a una nueva posición. Se busca el mejor tiempo para su cambio en la *i-ésima* posición, esto es, buscar el tiempo posible más cercano al intervalo $[E_i, L_i]$ para aterrizaje/despegue de ese vuelo. La vecindad resulta al colocar este vuelo en la *i-1-ésima* posición y recorrer todos los demás vuelos, colocar el vuelo en la *i-ésima* posición y recorrer todos los demás vuelos y colocar el vuelo en la *i+1-ésima* posición y recorrer todos los demás vuelos.

Primero se hizo la reasignación para aquél vuelo con un mayor retraso en cada iteración. Dicho vuelo se cambia de posición en todas aquellas posibles dentro de la secuencia de todos los vuelos, es decir, se realizan permutaciones de éste vuelo con todos

los demás siempre y cuando sea posible, debido a factibilidad, y se permita dicha permutación por la lista tabú.

Se intentó diversificar más la búsqueda variando o intercalando una iteración para el vuelo con mayor retraso y la siguiente para el vuelo con mayor retraso menor a *30 minutos*. Sin embargo se observó que arrojó mejores resultados centrarse únicamente en los vuelos con mayores retrasos.

La regla de detención se fue variando desde *3* hasta *25* iteraciones consecutivas sin mejora de la función objetivo. A partir de *5* iteraciones consecutivas hasta *25* ya arrojaba el mismo resultado, por lo que se decidió dejar la regla de detención en *5* iteraciones consecutivas sin mejora de la función objetivo o sí en esa iteración, todos los movimientos ya son tabú.

6.3. Pasos en el algoritmo de Búsqueda Tabú

A continuación se presenta el algoritmo por pasos de manera simple y resumida que se utilizó:

1. Se comienza con una solución de prueba inicial factible. Esta solución de arranque será la obtenida en la simulación (capítulo anterior). Ésta estará formada por el tipo de cola FIFO, como se mencionó anteriormente.

2. Se utiliza el procedimiento de búsqueda local. Si es posible, aquél vuelo con un mayor retraso será reasignado a una nueva posición (en secuencia y en tiempo). Si no, se busca el vuelo con el mayor retraso que sí pueda ser cambiado de posición. Los vuelos que tienen un retraso asociado pero aterrizaron justo al tiempo de su llegada, ya no podrán ser reasignados para mejorar el retraso ya que no existe el tiempo para reubicarlos, es decir que $L_j - p_j \geq r_j$.

3. Si el cambio de posición se encuentra en la lista tabú, sólo se aceptará si mejora la solución actual. Si no mejora la solución actual, se escoge el vuelo con el siguiente mayor retraso (y que sea factible mover de posición). La reasignación del vuelo será tomando a éste y reacomodarlo lo más cercano posible a su intervalo *[E_i, L_i]* recorriendo todos los demás vuelos.

4. Se coloca el cambio del vuelo, la posición y los vecinos (en la secuencia) en la lista tabú. La lista tabú tendrá un tamaño de 5. Cuando llegue el sexto cambio, se borrará el cambio más antiguo guardando el nuevo (estructura de cola).

5. Regla de detención: detener el proceso después de 5 iteraciones consecutivas sin mejora del valor de la función objetivo (o si todos los movimientos permitidos en esa iteración son tabú).

6.4. Ejemplo con 10 vuelos

A continuación se presenta un pequeño ejemplo con 10 vuelos, considerando únicamente aterrizajes, para un mejor entendimiento del problema y de cómo se hacen los cambios o permutaciones de vuelos:

Tabla 6.1 Ejemplo: información de vuelos

Vuelo *j*	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
r_j	1	3	8	9	10	15	19	24	25	28
d_j	[6, 14]	[4, 12]	[2, 10]	[7, 15]	[3, 11]	[10, 8]	[13, 21]	[14, 22]	[16, 24]	[20, 28]

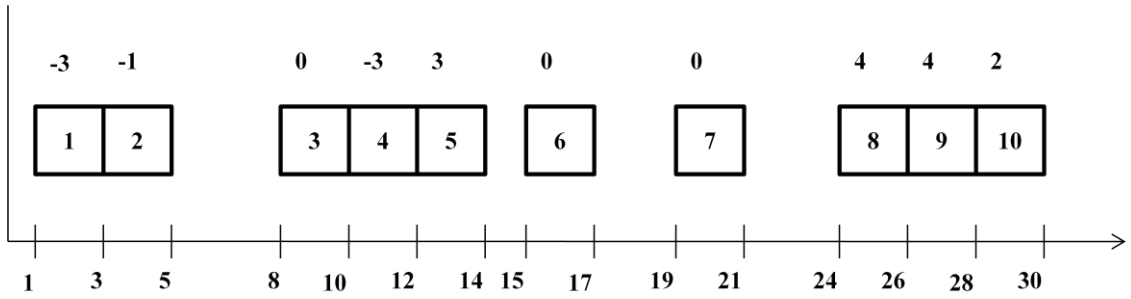
La función de costos estará determinada por los siguientes intervalos, ya que se necesita únicamente una función de costos convexa y linear por partes:

$0 < |R_j| < 2 \quad costo = |R_j| * 1$

$2 <= |R_j| < 4 \quad costo = |R_j| * 2$

$4 <= |R_j| < 6 \quad costo = |R_j| * 3 \ldots$

El resultado que se obtendría mediante el sistema FIFO se muestra en la siguiente figura en un diagrama de Gantt que representa la secuencia de aterrizajes dentro de la pista. El número que se encuentra arriba de cada caja (vuelo) es el retraso de cada uno de ellos. Este retraso puede ser positivo o negativo dependiendo si se adelantó a su aterrizaje o si se atrasó.



Secuencia: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10$

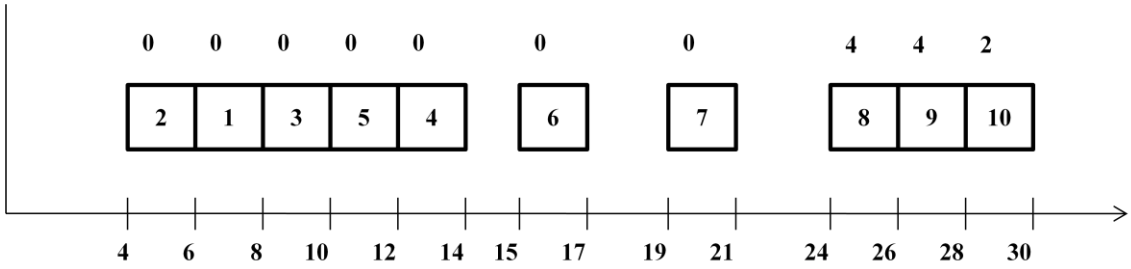
Figura 6.1 Diagrama de Gantt con el sistema FIFO

El resultado en secuencia sería la secuencia ordenada de cómo fueron llegando los vuelos. La suma de los retrasos, en valor absoluto, y los costos totales son:

$$\sum_{j=1}^{N} |R_j| = 20$$

$$\sum_{j=1}^{N} costo * |R_j| = 47$$

Haciendo cambios en la secuencia según las reglas presentadas anteriormente se puede obtener lo siguiente:



Secuencia: $2 \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10$

Figura 6.2 Diagrama de Gantt con Búsqueda Tabú

Obteniendo los siguientes resultados:

$$\sum_{j=1}^{N} |R_j| = 10$$

$$\sum_{j=1}^{N} costo * |R_j| = 28$$

Como se puede observar, se mejoró considerablemente tanto los retrasos como los costos asociados a ellos.

Los últimos *3* vuelos ya no se pueden reacomodar para una mejora ya que aterrizaron en el momento de su llegada, es decir, ya venían retrasados. En esos casos no se puede mejorar los retrasos de esos vuelos.

En el sistema completo, con 630 vuelos, es muy improbable tener ventanas de tiempo vacías, ya que por lo general los aeropuertos internacionales trabajan generalmente agotando toda su capacidad. Por lo tanto, es mucho más probable tener vuelo tras vuelo sin ventanas de tiempo vacías.

Otra diferencia consiste en que en el ejemplo no existió nunca una cola de aviones esperando a aterrizar debido a el tamaño tan pequeño de *n = 10*. Sin embargo, en el sistema completo con *630* vuelos, sí existirá una cola, por lo que se podrá cambiar la secuencia en ella.

6.5. Resultados obtenidos mediante Búsqueda Tabú

La programación del método heurístico se hizo en C++. En el anexo D se puede encontrar el código completo. Dentro de esta programación se utilizó el sistema completo, es decir, los 630 vuelos (aterrizajes y despegues).

A continuación se presentan los resultados de la programación del método heurístico *Búsqueda Tabú*:

El retraso máximo fue de **176.5738 minutos**, es decir, 2.943 horas. El retraso mínimo siguió siendo de **0 minutos**. La suma de retrasos fue de **40502.4969 minutos** o 675.0416 horas y la suma de los costos por los retrasos en un día fue de **$3,611,543.62 USD**. El promedio de retrasos fue de **64.28 minutos** o 1.07 horas y el promedio del costo por retrasos fue de **$5,732.6089USD**.

La desviación estándar de retrasos fue de **55.1553 minutos** de costos fue de **$5,395.927 USD**.

De los 630 vuelos, 178 aterrizaron o despegaron sin retraso.

Los resultados completos de la Búsqueda Tabú se pueden ver en el anexo E.

6.6. Validación del modelo

Para la validación del modelo se busca que la diferencia de medias sea positiva, es decir, que el promedio de costos del sistema simulado sea mayor al promedio de los costos obtenidos mediante la mejora propuesta por medio de la *Búsqueda Tabú.*

La prueba de hipótesis para la diferencia de medias es la siguiente:

Sean:

$X_1, X_2, \ldots, X_n$ $m.a. \sim N(\mu_1, \sigma_1^2)$

$Y_1, Y_2, \ldots, Y_m$ $m.a. \sim N(\mu_2, \sigma_2^2)$

Muestras independientes. Se supone igualdad de varianzas pero se desconoce las medias.

$H_0: \mu_1 - \mu_2 = D_0$ vs

a) $H_1: \mu_1 - \mu_2 \geq D_0 \Rightarrow RR = \{T > t_{(\alpha, n+m-2)}\}$

b) $H_1: \mu_1 - \mu_2 < D_0 \Rightarrow RR = \{T < -t_{(\alpha, n+m-2)}\}$

c) $H_1: \mu_1 - \mu_2 \neq D_0 \Rightarrow RR = \{|T| > t_{(\alpha/2, n+m-2)}\}$

donde $Z = \frac{(\bar{X} - \bar{Y}) - D_0}{S_p \sqrt{\frac{1}{n} + \frac{1}{m}}} \sim t_{(n+m-2)}$ y $S_p{}^2 = \frac{(n-1)S_x{}^2 (m-1)S_y{}^2}{n+m-2}$

En este caso, nuestra primera muestra es la obtenida por la simulación, la segunda es la obtenida mediante Búsqueda Tabú. $D_0 > 0$, esto es necesario para comprobar que existe una mejora en los retrasos y por lo tanto en los costos. También se puede ver que $m=n=630$ debido a que se tiene el mismo número de vuelos en cada una de las muestras.

Debido a que el número *(n+m-2)* es muy grande y no se encuentra tablas, se debe recurrir al Teorema Central del Límite (TCL) el cual dice:

Sea X_1, X_2, X_3...una sucesión de v.a.i.i.d. con media $\mu \in \mathbb{R}$ y varianza $\sigma^2 > 0$, entonces:

$$P\left[\frac{S_n - n\mu}{\sqrt{n\sigma^2}} \leq x\right] \xrightarrow{n \to \infty} \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-y^2}{2}} dy$$

donde $S_n^2 = \sum \frac{(X_i - \bar{X}_i)^2}{n-1}$ es la varianza muestral y $\frac{S_n - n\mu}{\sqrt{n\sigma^2}} = \frac{S_n - n\mu/n}{\sqrt{n\sigma^2}/n} = \frac{(\bar{X} - \mu)\sqrt{n}}{\sigma}$

por lo tanto el TCL se escribe:

$$P\left[\frac{(\bar{X} - \mu)\sqrt{n}}{\sigma} \leq x\right] \xrightarrow{n \to \infty} \Phi(x)$$

Si n es "grande"

$$P\left[\frac{(\bar{X} - \mu)\sqrt{n}}{\sigma} \leq x\right] \cong \Phi(x)$$

Por lo tanto la prueba de hipótesis queda como sigue:

H_0: $\mu_1 - \mu_2 = D_0$ *vs*

a) H_1: $\mu_1 - \mu_2 \geq D_0 \Rightarrow RR = \{Z \geq Z_\alpha\}$

b) H_1: $\mu_1 - \mu_2 < D_0 \Rightarrow RR = \{Z < -Z_\alpha\}$

c) H_1: $\mu_1 - \mu_2 \neq D_0 \Rightarrow RR = \left\{|Z| > Z_{\alpha/2}\right\}$

donde $Z = \frac{(\bar{X} - \bar{Y})}{\sqrt{\frac{S_x^2}{n} + \frac{S_y^2}{m}}}$ y $D_0 \in \mathbb{R}^+$

Nuestra $Z = 1.4909$. Si se quiere una prueba de hipótesis con el 95% de confianza, entonces se tiene que $\alpha = 0.05$. En tablas se puede buscar el valor de $Z_\alpha = 0.5199$ y por lo tanto se cumple que $Z \geq Z_\alpha$. Si quisiéramos una confianza del 99%, entonces se tiene que $\alpha = 0.01$. En tablas, el valor de $Z_\alpha = 0.5040$ y, por lo tanto, se sigue cumpliendo que $Z \geq Z_\alpha$.

En este caso en particular, necesitamos que la diferencia de medias exista y sea mayor a cero. Entonces $D_0 = 0$. Con la prueba anterior se muestra que se cumple una mejora en los retrasos utilizando el método heurístico de Búsqueda Tabú.

6.7. Comparación de los resultados de la simulación y Búsqueda Tabú

Anteriormente se validó el modelo mediante pruebas de hipótesis. Sin embargo se puede demostrar una mejora simplemente mediante la comparación de resultados obtenidos en la simulación y en el método heurístico BT.

En la siguiente tabla se muestra la comparación entre estos dos modelos para así poder mostrar la mejora realizada.

Tabla 6.2. Comparación

	Simulación	**Búsqueda Tabú**	**Mejora**
Costos totales (USD)	3965367.42	3611543.62	353823.8
Costo promedio (USD)	6294.234	5732.60893	561.62507
Varianza de costo (USD^2)	40529415.1	29116028.6	11413386.5
Desviación estándar del costo (USD)	6366.27168	5395.92704	970.34464
Retrasos totales (min)	43549.146	40502.4969	3046.6491
Retraso promedio (min)	69.125628	64.2896777	4.83595
Varianza del retraso (min^2)	3586.1214	3042.1138	544.0076
Desviación estándar del retraso (min)	59.884233	55.1553606	4.7289

CONCLUSIONES

Este tipo de problemas se pueden modelar de diferentes maneras, como se ´explicó en el estado del arte o antecedentes. Muchos autores utilizan la programación matemática para poder modelar estos problemas y buscar una solución del problema de manera exacta. Esta programación puede ser de manera estática o dinámica y estocástica o determinística. El caso más complejo y más realista sería un modelo dinámico y estocástico. Sin embargo, este tipo de modelaciones pueden ser demasiado complejas para poder realizarse.

Como se mencionó anteriormente, resolver este tipo de problemas de manera exacta es muy poco probable dada su naturaleza y la complejidad computacional inherente que tiene. Dado que son problemas combinatorios, se recomienda utilizar métodos de aproximación como los son los métodos heurísticos. Si bien, éste tipo de métodos no aseguran proporcionar la mejor solución, pueden arrojar una solución muy cercana al óptimo. Sobre todo las metaheurísticas que ya han sido probadas y se pueden aplicar a muchos problemas. En este caso, existen dos metaheurísticas probadas, que son las que mejor funcionan para resolver problemas de *scheduling*: *Búsqueda Tabú* y *Recocido Simulado*.

En este trabajo de investigación, primero se explicó la problemática. Posteriormente se describió el sistema y se investigó las diferentes maneras en las que se ha resuelto en los últimos años (estado del arte). Se explicó toda la teoría de *scheduling*, ya que en ella se basa el sistema de esta investigación. La teoría de *scheduling* es muy extensa, como se pudo ver en capítulos anteriores. Esta investigación se centró en la formulación del problema mediante la analogía de la programación de actividades de una única máquina ya que se trata solamente de despegues y aterrizajes en una pista. Se realizó un modelo matemático para la resolución del problema .Dada la complejidad computacional de dicho problema, fue necesario utilizar un método de aproximación, el cual fue una metaheurística llamado *Búsqueda Tabú.* Este tipo de métodos heurísticos, requieren una solución inicial factible para comenzar. Debido a que no se contaba con información suficiente, se realizó una simulación del sistema para utilizarla como solución inicial de arranque para la *Búsqueda Tabú* y para poder demostrar posteriormente una mejora en los costos y retrasos

en la secuencia de despegues y aterrizajes en una pista en el Aeropuerto Internacional de la Ciudad de México.

Mediante la metodología utilizada en este trabajo de investigación, anteriormente descrita, se pudo obtener una reducción tanto en costos como en retrasos los cuales se presentan a continuación. Esta reducción se presentan tanto en cantidad como en porcentaje diario y anual.

La reducción de minutos en cuanto a retrasos fue de **3046.646 minutos**, lo que es lo mismo que **50.7774 horas**. Esto equivale a un **7%** en reducción de retrasos diarios. Si se toma esta cifra de manera anual, se obtiene una reducción de **18533.7632 horas** anuales.

El retraso máximo se redujo de **223.243 minutos** a **176.5738 minutos**, es decir que se logró reducir el retraso máximo **46.6692 minutos.**

El valor esperado o promedio de costos por vuelo relacionado a retrasos se disminuyó en **$561.624 USD**. Si se toma el tipo de cambio del 7/01/2011 (12.2256) se tiene una reducción **$6,866.1904 PM**.

En cuanto al costo, la reducción fue de $**353,823.918 USD** diarios. Esto representa una reducción del **9%** diario. Anualmente existiría un ahorro total de **$129,145,730.07 USD**. Tomando el mismo tipo de cambio se tiene una reducción anual de **$1’578,884,038 PM**.

Para futuras investigaciones acerca de este tema, se puede intentar usar la clasificación de tamaños de aviones (grande, mediano y chico) para añadir las separaciones necesarias entre éstos en la secuencia. Esta distancia, dividida por la velocidad aproximada que llevan los aviones, da como resultado el tiempo que existe entre cada par de aviones. Este tiempo se puede sumar en cada vuelo a su tiempo de procesamiento y hacer el mismo análisis.

Se puede buscar resolver problemas muy similares aumentando su complejidad (varias pistas, ruteo, etc.).

También se puede probar resolver el mismo problema mediante un método heurístico diferente como *Recocido Simulado*, ya que es la otra metaheurística recomendada para problemas de *scheduling*.

A continuación se presenta un cuadro haciendo una comparación para cada uno de los tipos de *scheduling* que existen con el problema de minimización de costos y retrasos de despegues y aterrizajes para futuras investigaciones. Como ya se mencionó anteriormente, durante esta investigación se analizó únicamente el primer caso, es decir, una única máquina.

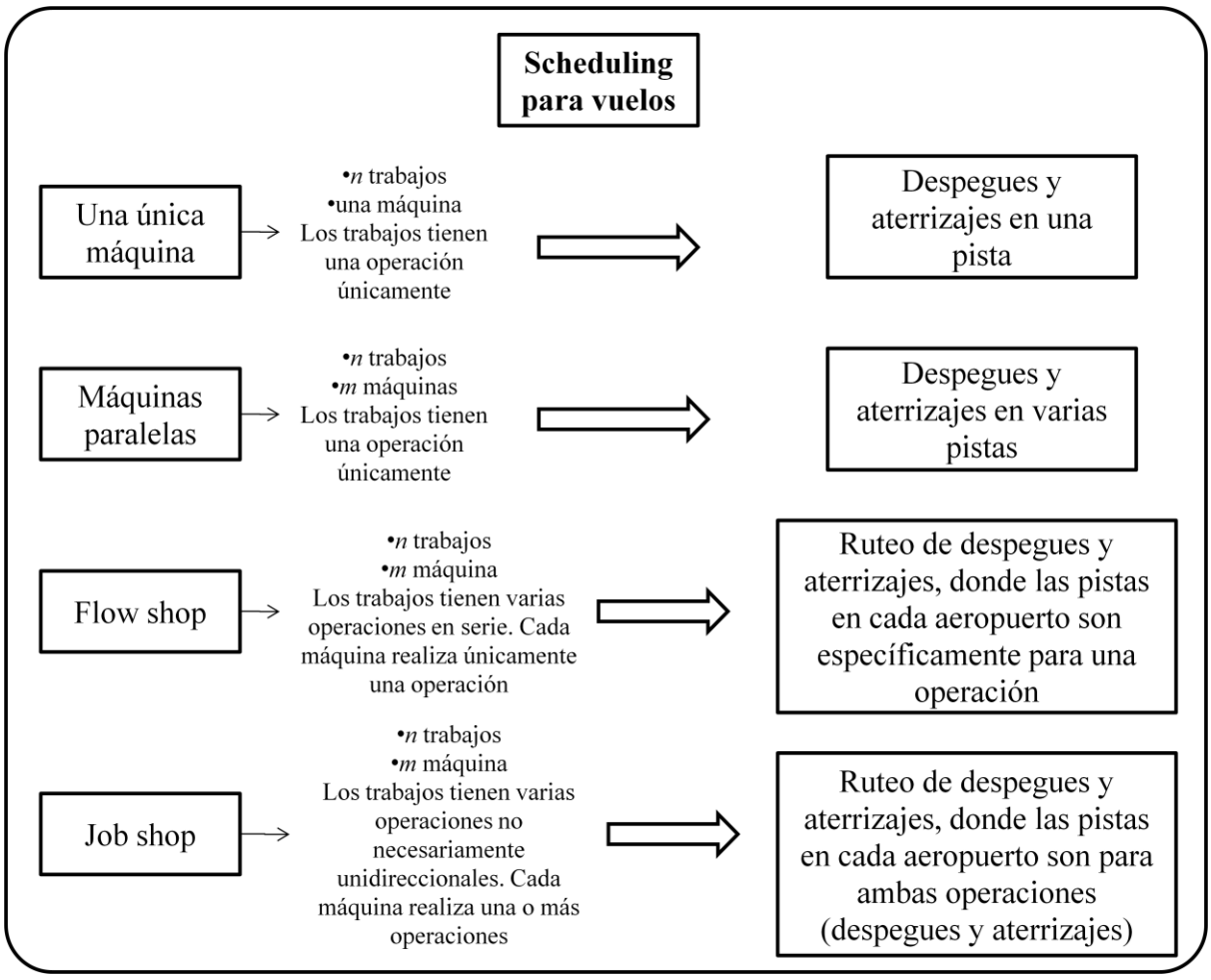


Figura 7.1. Scheduling en vuelos

Cada una de estas cuatro variantes también tiene la posibilidad de ser determinística o estocástica aumentando así su complejidad.

Es importante señalar también la dificultad que representa la modificación en la secuencia de despegues y aterrizajes. Se tendría que hacer un análisis de factibilidad y riesgos de cambiar la secuencia de los aviones que se encuentran en cola formados para utilizar la pista. Sabemos que es posible dado que se hace en casos de emergencia, pero se tendría que investigar el hacer estas operaciones continuamente sin provocar accidentes aéreos. Los autores presentados en el estado del arte consideran también permutaciones en dicha secuencia e incluso varias mejoras presentadas por ellos, han sido probadas en algunos aeropuertos, sin embargo, no especifican de qué manera es posible hacer estos cambios en la cola de aviones.

BIBLIOGRAFÍA

- ALONSO Antonio, *et al*., 1998, *A stochastic 0-1 program based approach for the air traffic flow management problem*, European Journal of Operational Research.
- ARTIOUCHIN Konstantin, *et al*., 2007, *Runway sequencing with holding patterns*, European Journal of Operational Research.
- BAKER R. Kenneth, Dan Trietsch, 2009, *Principles of sequencing and scheduling*, John Wiley & Sons Inc., New Yersey E.U.A.
- BALAKRISHNAN Hansa, *et al*., 2006, *Scheduling Aircraft Landings under Constrained Position Shifting*, AIAA Guidance, Navigation, and Control Conference and Exhibit, Keystone, Colorado.
- BARD Jonathan F., *et al*., 2007, *Reallocating arrival slots during a ground delay program,* Transportation research part B, Science Direct.
- BÄUERLE N., *et al*., 2005, *On the waiting time of arriving aircrafts and the capacity of airports with one or two runways*, European Journal of Operational Research.
- BLY Elizabeth, 2005, *2nd USA/EUROPE AIR TRAFFIC MANAGEMENT R&D SEMINAR Orlando*, Massachusetts Institute of Technology.
- CARR Gregory C., *et al*. 1998, *Airline arrival prioritization in sequencing and scheduling,* 2nd USA/EUROPE AIR TRAFFIC MANAGEMENT R&D SEMINAR Orlando.
- CAO Jung-Ming, Kanafani Adib, 2000, *The value of runway time slots for airlines*, European Journal of Operational Research
- DE MATOS Paula Leal, *et al*., 1999, *The application of operational research to European air traffic flow management - understanding the context*, European Journal of Operational Research.
- DELL'OLMO Paolo, *et al*., 2001, *A new hierarchical architecture for Air Traffic Management: Optimization of airway capacity in a Free Flight scenario*, European Journal of Operational Research.

- DORNDORF Ulrich, *et al*., 2005, *Flight gate scheduling: State-of-the-art and recent developments*, OMEGA the international journal of management science.
- GUTIÉRREZ Pulido Humberto, 2008, Román de la Vara Salazar, *Análisis y diseño de experimentos*, McGraw Hill Interamericana, México.
- HU Xiao.Bing, *et al*., 2005, *Genetic algorithm based on receding horizon control for arrival sequencing and scheduling*, Engineering Applications of Artificial Intelligence.
- INNIS T., M. O. Ball, (2004), *Estimating one-parameter airport arrival capacity distributions for air traffic flow management*, Air Traffic Control Quarterly.
- LIU Pei-chen Barry, *et al*. 2008, *Scenario-based air traffic flow management: From theory to practice*, Transportation research part B, Science Direct.
- PINEDO Michael L., 2008, *Scheduling theory, algorithms and systems*, Springer, tercera edición, New York E.U.A.
- SHERALI Hanif D., *et al*., 2005, *Airline fleet assignment concepts, models, and algorithms*, European Journal of Operational Research.
- SOOMER M.J. *et al*., 2007, *Scheduling aircraft landings using airlines' preferences,* European Journal of operational research.
- STAMATOPOULOS Miltiadis A., *et al*., 2002, *A decision support system for airport trategic planning*
- *AICM en cifras 2009*, http://www.aicm.com.mx/acercadelaicm/archivos/files/Estadisticas/EstaDiciembre2009.pdf, [Acceso: marzo 2010]

APÉNDICE A

COMPLEJIDAD COMPUTACIONAL

La complejidad computacional estudia tanto la eficiencia de los algoritmos como la dificultad inherente de problemas de importancia práctica y teórica.

El esfuerzo computacional requerido para resolver los problemas varía mucho de uno a otro cuando se quiere resolverlos de manera precisa. Como se dijo anteriormente, el problema de *scheduling* es un problema *NP-duro*. El término técnico para un problema *duro* es *NP-completo*, lo cual significa que no se puede definir un algoritmo eficiente para la solución exacta de ese problema en particular. Debido a esto, para este tipo de problemas NP-completos se deben definir soluciones aproximadas.

Al hablar de eficiencia de los algoritmos, los rápidos avances en la teoría de las computadoras hacen medidas físicas (tiempo de corrida, requerimientos de memoria) irrelevantes. Una medida más estandarizada es el número de operaciones elementales de cómputo que toma el resolver un problema en el peor caso. El desempeño promedio no es seguro. El número de operaciones elementales depende, obviamente, del tamaño de los datos de entrada del problema. Clasificar 1 millón de datos es completamente diferente a clasificar tan sólo 10. Por lo tanto, se expresa el número de operaciones elementales como una función dependiente del "tamaño" del trabajo requerido para resolver el problema en el peor de los casos.

Si un algoritmo resuelve un problema de tamaño n en un máximo de $8n^3 + 3n^2 + 15$ operaciones, significa que su complejidad esta acotada por n^3 operaciones. Esto es porque el término de n^3 es el mayor grado y éste determina la tasa de crecimiento y porque no existe diferencia entre $8n^3$ y n^3. Por lo tanto, la complejidad de este algoritmo está descrita por la función $g(n)= n^3$. Formalmente se dice que este algoritmo es *de orden* $O(n^3)$ o *toma un tiempo de* $O(n^3)$.

Para n suficientemente grande:

$$\log n < n < n^2 < n^3 < 2^n$$

Lo que es lo mismo que:

$$O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n).$$

Decir que un algoritmo corre en un tiempo *O(f(n))*, para alguna función de crecimiento *f(n)*, significa que para *n* suficientemente grande, el tiempo requerido para ejecutar el algoritmo no excede a *cf(n)* (donde *c* es una constante no especificada pero que no es significativa para valores grandes de *n*). Por lo que es importante saber si *f(n)* es polinomial.

T(n) (tiempo de corrida) es *O(f(n))* si existen dos constantes positivas *c* y n_0 tales que *T(n)* <= *cf(n)* para todo *n* > n_0. Cuando *T(n)=O(f(n))*, se dice que *T(n)* no crece tan rápido como *f(n)*, por lo tanto *f(n)* es una cota superior de *T(n)*.

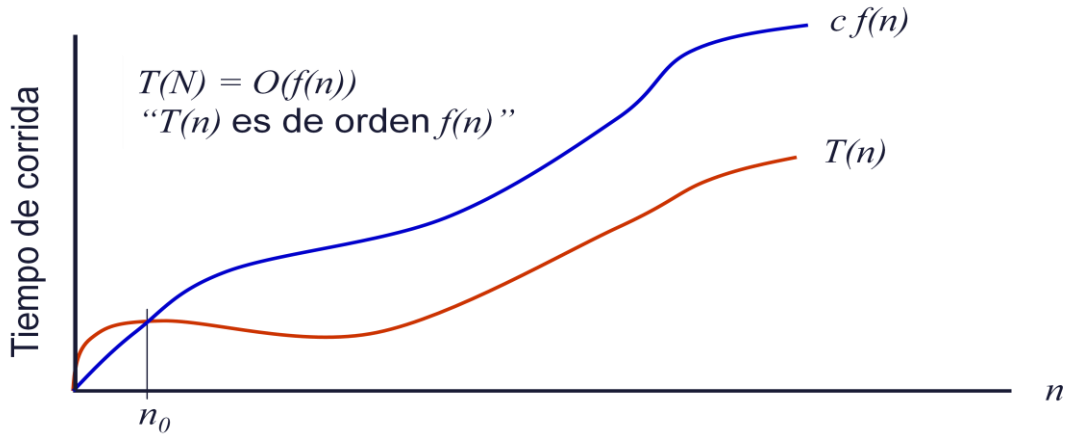


Figura I **Función de crecimiento (**Tomado de Elizondo, 2009)

Si $T_1(n) = O(f(n))$ y $T_2(n) = O(g(n))$ entonces:

$T_1(n) + T_2(n) = max\{O(f(n)), O(g(n))\}$.

También:

$T_1(n) \times T_2(n) = O(f(n) \times g(n))$.

En la siguiente tabla se muestra el tiempo aproximado que se tardaría un algoritmo con *n=100*.

Tabla I **Tiempos de resolución para *n=100* (**Tomado de Elizondo 2009)

Complejidad	**Tiempo**
O(log n)	10^{-7} segundos
O(n)	10^{-6} segundos
O(n log n)	10^{-5} segundos
$O(n^2)$	10^{-4} segundos
$O(n^3)$	1 segundo
$O(n^6)$	3 minutos
$O(2^n)$	10^{14} **años**
O(n!)	10^{142} **años**

Algoritmos de tiempo polinomial y exponencial

Aquellos algoritmos que tienen una complejidad de tiempo polinomial, es decir, que toman un tiempo *O(g(n))* donde *g(n)* es un polinomio o una función acotada por un polinomio, se puede decir que son prácticos.

Los algoritmos que tienen un orden *O(log n), O(n), O(n log n),* $O(n^2)$, $O(n^3)$, etc., son llamados algoritmos de tiempo polinomial. Los algoritmos que no se pueden acotar por funciones polinomiales se denominan algoritmos de tiempo exponencial. Estos incluyen órdenes de crecimiento explosivo, dentro de los cuales también se encuentran factores no exponenciales como *n!*.

En la siguiente tabla se muestran las mejoras dependiendo del tamaño de la instancia del problema:

Tabla II **Mejoras computacionales** (Tomado de Elizondo 2009)

Función de complejidad de tiempo	Tamaño de la mayor instancia del problema solucionable en 1 hora		
	Con una computadora actual	Con una computadora 100 veces más rápida	Con una computadora 1000 veces más rápida
n	$N1$	$100N1$	$1000N1$
n^2	$N2$	$10N2$	$31.6N2$
n^3	$N3$	$4.64N3$	$10N3$
2^n	$N4$	$N4+6.64$	$N4+9.97$
3^n	$N5$	$N5+4.19$	$N5+6.29$

Por ejemplo, en el caso de $O(n^2)$:

Una computadora actual corre para an^2 segundos, 1hr. = $aN_2{}^2$, una computadora 100 veces más rápida corre para $an^2/100$, 1hr. = $aN_X{}^2/100$, por lo tanto $100N_2{}^2 = N_X{}^2$, $N_X = 10N_2$.

Y en el caso de $O(2^n)$:

Una computadora actual corre para $a2^n$ segundos, 1hr.= $a2^{N_4}$, una computadora 100 veces más rápida corre para $a2^n/100$ segundos, 1hr. = $a2^{N_X}/100$, por lo tanto $100(2^{N_4}) = 2^{N_X}$, $N_X = N_4 + (\log 100/\log 2) = N_4 + 6.64$.

Como puede verse, la mejora en tecnología es multiplicativa en algoritmos de tiempo polinomial y únicamente aditiva en algoritmos de tiempo exponencial. La situación es mucho peor que la mostrada en la tabla si las complejidades involucran factoriales (como sería el caso de *scheduling* que es un problema combinatorio).

Un segundo argumento es que, en algoritmos de tiempo polinomial, una vez que son descubiertos, pasan por una serie de mejoras, las cuales son reducciones en las constantes en sus funciones de complejidad y en sus propios grados. Este es un resultado empírico y, en parte, explica por qué complejidades como $O(n^{80})$ u $O(n^{1000})$ no aparecen en la práctica. Los algoritmos de tiempo polinomial típicamente tienen un grado de entre 2 y 3, y no incluyen coeficientes muy grandes. La brecha real para la solución de un problema, está en la definición del primer algoritmo de tiempo polinomial. Para problemas combinatorios, el "salto" a la clase polinomial requiere comúnmente un análisis profundo de la naturaleza del problema (si es que este salto puede realizarse).

La conclusión general es que un problema puede considerarse "eficientemente resuelto" cuando se ha encontrado un algoritmo de tiempo polinomial para su solución.

Algunos ejemplos de problemas de optimización que son considerados polinomiales son:

- Programación Lineal

- Algunos problemas de Ruta más Corta
- Algunos problemas de Ruta más Larga (por ejemplo los cálculos de PERT/CPM simple)
- El problema de Árbol de Mínima Expansión
- Algunos problemas de flujo de redes
- Problemas de Pareamiento (*matching*)
- Problemas de Transporte, Asignación y Trasbordo
- Varios casos especiales de problemas duros.

Esta lista, a pesar de todos los problemas que incluye, es muy corta comparada con la lista de problemas para los cuales no se conocen aún algoritmos de tiempo polinomial.

Problemas de decisión y el problema de factibilidad

Un marco contextual apropiado para el análisis de problemas es el de los problemas de optimización, esto es, minimizar o maximizar una función objetivo sujeta a una o más restricciones (como se menciono anteriormente).

Una clase especial de problemas de optimización requiere únicamente de una respuesta "sí" o "no". Estos son llamados problemas de decisión. La teoría de Complejidad Computacional implica estricta atención en estos problemas por su uniformidad. Los resultados pueden ser fácilmente generalizados ya que:

1. Para cada problema de optimización existe una versión de decisión,
2. cualquier resultado de complejidad para la versión de decisión se mantiene también para el problema original, esto es, el problema original no es "mucho más duro" que la versión de decisión.

Se sabe que, dado un problema de optimización, se define un problema de decisión asociado a él, lo cual es una pregunta que puede ser contestada sí o no. Por otro lado, varios problemas computacionales bien conocidos son problemas de decisión. Además, se ha puntualizado que un problema de optimización original no es más difícil que el problema de decisión; cualquier resultado probado sobre la complejidad del problema de decisión será aplicable también al problema de optimización.

Llamemos a $\pi(*)$ la versión de decisión del problema "*". La definición del problema de decisión a partir del problema de optimización, permite estudiar ambos tipos de problemas de manera uniforme.

Los problemas de factibilidad son de central importancia en este contexto. Si para algún problema de decisión existe un conjunto de valores "sí" y "no" que, asignados a las variables componen un resultado "sí", entonces se tiene un problema factible (problema de factibilidad)

Algoritmos no-determinísticos

Un algoritmo no-determinístico es una herramienta teórica que no existe en la realidad. Es como un algoritmo ordinario, con la excepción de que tenemos permitido usa la siguiente instrucción imposible:

goto **both** label 1, label 2

Esto divide el cómputo de dos procesos paralelos en un solo paso en el mismo procesador (no en procesamiento en paralelo).

Para tener una idea del poder de cómputo no-determinístico, consideramos la conocida versión del problema de Programación entera 0-1 que es un problema "duro" (no se conoce un algoritmo determinístico para él):

Dada una matriz **A** de *m x n* y un *m* vector entero **b**, ¿existe un *n* vector **x**, con elementos 0-1, tal que **Ax=b**?

Esta poderosa instrucción permite resolver el problema en tiempo polinomial, precisamente por medio de la exploración de todas las posibles combinaciones de valores de los elementos del vector **x**;

```
begin
        for j=1,…,n
                goto both A,B
                A: x_j = 0
                goto again
```

B: $x_j = 1$

: again

next j

if $x=(x_1,\ldots,x_n)$ satisfies Ax=b then output "yes" else output "no"

end

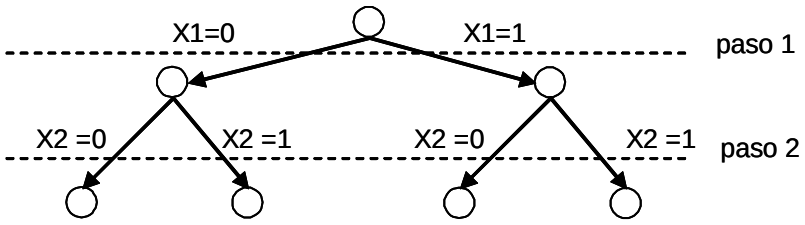


Figura II Algoritmo no-determinístico

De hecho, si se tuviera computadoras capaces de ejecutar esta instrucción extra, no sería necesaria la teoría de Complejidad Computacional.

Clases P y NP

Existen dos clases importantes de problemas de decisión: las clases P y NP.

La clase P consiste en todos aquellos problemas de decisión para los cuales existe un algoritmo polinomial.

Para la clase NP simplemente se requiere que alguna respuesta "sí" sea "fácilmente" verificable. Tanto la codificación de la respuesta como el tiempo que toma verificar su validez, deben ser "cortos", esto es, tiempo polinomial o polinomialmente acotado. Formalmente decimos que cualquier instancia "sí" del problema tiene la propiedad de "pertenencia".

NP se establece para "Polinomial No-Determinístico", debido a una alternativa (y equivalente) definición basada en la noción de algoritmos no-determinísticos. Una observación de la definición anterior es que, con la finalidad de mostrar que un problema pertenece a la clase NP, no es necesario mostrar de qué manera esta respuesta, fácilmente verificable, puede ser producida. En otras palabras es suficiente mostrar que tal estado de pertenencia existe.

Para clarificar lo anterior, consideremos el Problema del Agente Viajero (TSP), que dice: "dado un número n de ciudades, $n \geq 3$ y entero, una matriz nxn de distancias enteras no negativa $C = [c_{ij}]$, y un entero no negativo L : "¿existe un tour cerrado que pasa exactamente una vez por cada ciudad, con una longitud total $\leq L$?".

Si imaginamos que tenemos una respuesta "sí", ésta es un orden de ciudades, digamos $c_1, c_2, c_3, ..., c_n$, tal que la suma de las ligas $(c_1, c_2), (c_2, c_3), ..., (c_n, c_1)$ es menor o igual a L. La forma en que se obtuvo esta solución es irrelevante. Es claro que podemos verificar fácilmente la distancia del tour, así que sabemos que $\pi(\text{TSP}) \in$ NP. Por otro lado, no conocemos si el problema está también en la clase P, debido a que ningún algoritmo polinomial ha sido aún encontrado para él.

Por lo tanto, la clase NP consiste de todos los problemas "razonables" de importancia teórica y/o práctica. Para problemas que no están en la clase NP, la propia verificación de que una solución es válida puede ser extremadamente difícil.

Evidentemente, los problemas "fáciles" de la clase P son también "razonables", esto es, ellos también pertenecen a NP. Esto es P es un subconjunto de NP (P $\subseteq$ NP).

La cuestión importante es ahora si P es un subconjunto propio de NP, esto es, P $\subset$ NP. Esta es una conjetura muy importante en Complejidad Computacional, en otras palabras dice: "no todos los problemas pueden resolverse eficientemente". Aunque no existe la prueba formal aún, existe una fuerte evidencia para la validez de esta conjetura.

Reducciones y transformaciones polinomiales.

Las herramientas básicas para relacionar las complejidades de varios problemas son las reducciones y transformaciones.

Decimos que un problema A se reduce en tiempo polinomial a otro problema B, si y sólo si:

1. hay un algoritmo para A que usa una subrutina para B,
2. cada llamado a la subrutina para B cuenta como un único paso, y
3. el algoritmo para A corre en tiempo polinomial.

Lo anterior implica que la subrutina para B puede ser llamada a lo más un número polinomialmente acotado de veces.

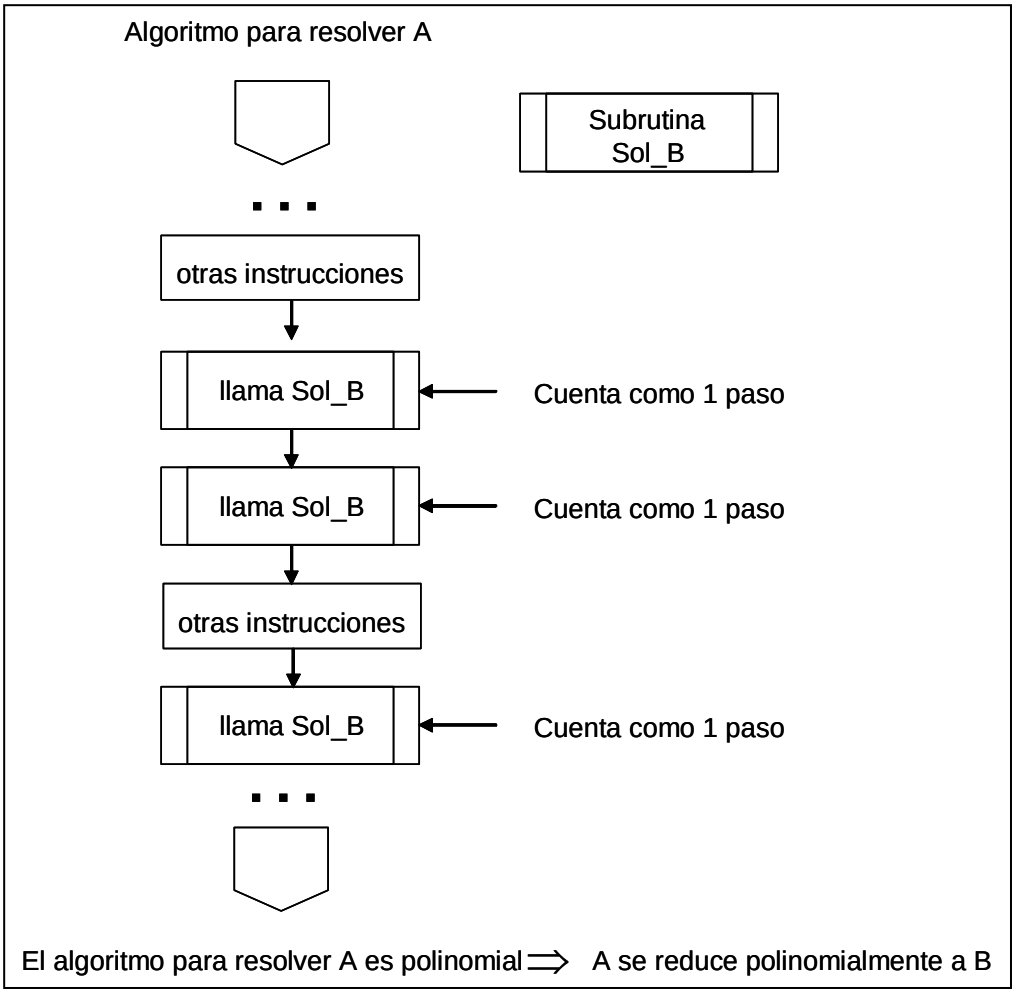


Figura III **Algoritmo A con subrutina B** (Tomado de Elizondo 2009)

La implicación práctica viene de la siguiente proposición:

Si A reduce polinomialmente a B y existe un algoritmo de tiempo polinomial para B, entonces hay un algoritmo de tiempo polinomial para A.

La prueba está basada en el hecho de que la composición de dos polinomiales es polinomial.

Los tres casos básicos son los siguientes:

- (A reduce a B) y (B es "fácil") $\Rightarrow$ A es fácil

- (A reduce a B) y (A es "duro") $\Rightarrow$ B es duro (debido a que si B fuera fácil A sería duro y eso es una contradicción)
- (A reduce a B) y (B es "duro") $\Rightarrow$ (no hay conclusión para A, un caso muy común)

Ahora, supongamos que representamos la "dureza" de un problema con una variable binaria, esto es, *d=1* si el problema es "duro" y *d=0* si el problema es "fácil".

Entonces, "A reduce polinomialmente a B" puede expresarse con $d_B \geq d_A$. Desde luego:

- $(d_B \geq d_A)$ *y* $(d_B = 0) \Rightarrow d_A = 0$
- $(d_B \geq d_A)$ *y* $(d_A = 1) \Rightarrow d_B = 1$
- $(d_B \geq d_A)$ *y* $(d_B = 1) \Rightarrow d_A$ puede tomar cualquier valor

De aquí, podemos justificar el decir que "B es al menos tan duro como A":

Si A reduce polinomialmente a B, entonces es posible implicar que A puede verse como un caso especial de B y en consecuencia, "B es al menos tan duro como A".

Un caso especial de reducciones polinomiales es la siguiente: el algoritmo para A, primero construye las entradas para la subrutina hipotética B, entonces llama a ésta última exactamente una vez, y finalmente regresa la respuesta al algoritmo para A.

Las reducciones de este tipo son llamadas transformaciones polinomiales (decimos que un problema A se transforma en tiempo polinomial al problema B, y se usa la notación $A \propto B$). Son especialmente útiles debido a que la relación de transformabilidad polinomial es transitiva y ésto es fácilmente probado usando de nuevo el hecho de que la composición de dos polinomiales es polinomial.

Problemas NP-completos

Si $A \propto B$ para todo $A \in NP$, entonces algún problema perteneciente a la clase NP puede ser visto como un caso especial de B y llamarlo NP-duro. Dicho de otra forma, otro C es un problema NP-duro, si pertenece a NP y puede ser visto como un caso especial de B.

Decimos que un problema es NP-completo si:

i. pertenece a la clase NP y

ii. todos los demás problemas en NP se transforman polinomialmente a él.

Un problema NP-completo tiene la siguiente propiedad importante:

Si existe un algoritmo eficiente (esto es, polinomial) para algún problema NP-completo, entonces existe una algoritmo eficiente para cada problema en NP.

Lo anterior debido a que, si decimos que P^* es este problema:

Todos los problemas en NP se transforman polinomialmente a P^* y dado que la transformación polinomial es un caso especial de reducibilidad polinomial, se aplica el caso1:

(Todos los problemas en NP se reducen a P^*) y (P^* es "fácil") $\Rightarrow$ todos los problemas en NP son fáciles.

Por esta razón, los problemas NP-completos son el principal objetivo en la búsqueda de algoritmos eficientes.

Además, dado que $A \propto B$, $B \in NP$ y A es NP-completo, entonces B es también NP-completo.

El siguiente es un mapa del ámbito NP (de manera simplificada):

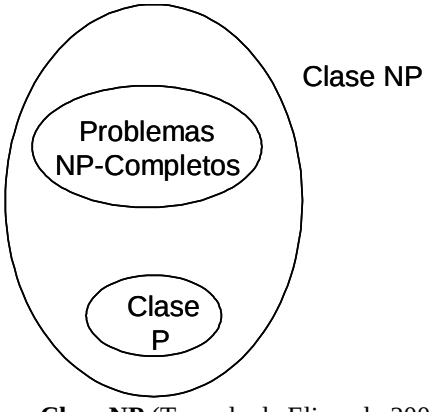


Figura IV **Clase NP** (Tomado de Elizondo 2009)

APÉNDICE B

DISEÑO DE EXPERIMENTOS (DDE)

DCA es el diseño más simple que se utiliza para comparar dos o más tratamientos. Se hacen *N* pruebas al azar de manera que los posibles efectos ambientales y temporales se vayan repartiendo equitativamente entre los tratamientos.

Supongamos que se tienen *k* tratamientos independientes y con medias desconocidas $\mu_1, \mu_2, \ldots, \mu_k$, así como varianzas que se suponen iguales $\sigma_1 = \sigma_2 = \ldots = \sigma_k$. Es recomendable usar el mismo número de repeticiones en cada tratamiento (diseño balanceado).

Cada observación Y_{ij} (observación *j* con tratamiento *i*) será descrita por:

$Y_{ij} = \mu + \tau_i + \varepsilon_i$ Modelo de efectos fijos

donde:

μ = media global

τ_i = efecto del tratamiento *i* sobre la variable de respuesta ($\tau_i = \mu_i - \mu$)

ε_i = error atribuible a la medición Y_{ij}

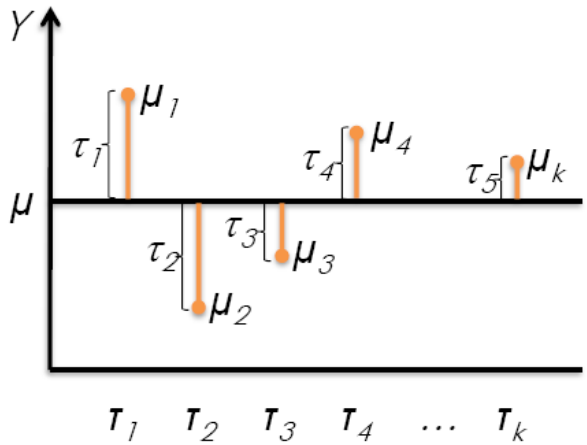


Figura I Efecto del tratamiento τ_i(Tomado de Gutiérrez 2008)

ANOVA

El análisis de varianza (ANOVA) es la técnica central en el análisis de datos experimentales. Separa la variación total en las partes con las que contribuye cada fuente de variación en el experimento.

Figura II **Variación total en cada parte de un DCA** (Tomado de Gutiérrez 2008)

Notación:

$Y_{i\bullet}$ = suma de las observaciones del tratamiento *i*

$\bar{Y}_{i\bullet}$ = media de las observaciones del *i-ésimo* tratamiento

$Y_{\bullet\bullet}$ = suma total de las $N = n_1 + n_2 + \ldots + n_k$ mediciones

$\bar{Y}_{\bullet\bullet}$ = media global de todas las observaciones

El objetivo del análisis de varianza en el DCA es probar la hipótesis de igualdad de los tratamientos con respecto a la media de la correspondiente variable de respuesta.

$$H_0 : \mu_1 = \mu_2 = \ldots = \mu_k = \mu$$

$$H_A : \mu_i \neq \mu_j \quad \textit{para algún } i \neq j$$

Lo cual es lo mismo que:

$$H_0 : \sigma_1 = \sigma_2 = \ldots = \sigma_k = 0$$

$$H_A : \tau_i \neq 0 \quad \textit{para algún } i$$

Mediante la técnica ANOVA se tiene que descomponer la variabilidad total de los datos en sus dos componentes (variabilidad del tratamiento y error aleatorio). Una medida de variabilidad total es la *suma total de cuadrados.*

$$SC_T = \sum_{i=1}^{k}\sum_{j=1}^{n_i}(Y_{ij} - Y_{..})^2$$

$$SC_T = \sum_{i=1}^{k} n_i(Y_{i.} - Y_{..})^2 + \sum_{i=1}^{k}\sum_{j=1}^{n_i}(Y_{ij} - Y_{i.})^2$$

$$SC_T = SC_{TRAT} + SC_E$$

Cuyos grados de libertado son;

$$N - 1 = (k - 1) + (N - k)$$

Las sumas de los cuadrados divididos entre sus respectivos grados de libertad se llaman *cuadrados medios.*

$$CM_{TRAT} = \frac{SC_{TRAT}}{(k-1)}$$

$$CM_E = \frac{SC_E}{(N-k)}$$

Si la hipótesis nula es verdadera, ambos cuadrados medios estiman la varianza σ^2, por lo tanto SC_E/σ^2 y SC_{TRAT}/σ^2 son v.a.i.i.d (variables aleatorias independientes e idénticamente distribuidas) χ^2. Entonces, bajo el supuesto de que la hipótesis H_0 es verdadera, el estadístico sigue una distribución Fisher:

$$F_0 = \frac{CM_{TRAT}}{CM_E}$$

Para un nivel de significancia α, se rechaza H_0 si $F_0 > F_{\alpha, k-1, N-k}$ o si *p-value* $< \alpha$ (*donde p-value* $= P(F > F_0)$).

En esta simulación, como se mencionó con anterioridad, la variable de respuesta va a ser el retraso promedio por día de los aterrizajes y despegues. Para tener un sistema estable, todas estas medias deben ser aproximadamente iguales, por lo que se requiere saber el tamaño de de la muestra o de réplicas durante la simulación.

Se puede obtener un tamaño de muestra por medio del intervalo de confianza:

Se tiene que probar $H_0 : \mu_i = \mu_j$

$H_A : \mu_i \neq \mu_j$ *para todo* $i \neq j$

Para k tratamientos se tiene un total de $k(k-1)/2$ pares de medias.

H_0 se rechaza si:

$$\left|Y_{i.} + Y_{j.}\right| > t_{\frac{\alpha}{2},N-k}\sqrt{CM_E\left(\frac{1}{n_i}\right) + \left(\frac{1}{n_j}\right)} = LSD$$

donde *LSD* es la *diferencia mínima significativa (least significant difference)*

Si $n_1 = n_2 = \ldots = n_k = n$

$$LSD = t_{\frac{\alpha}{2},N-k}\sqrt{2CM_E/n}$$

Por lo tanto:

$$n = \frac{2(t_{\frac{\alpha}{2},N-k})^2 CM_E}{LSD^2}$$

ANEXO A

Programación de la simulación en C++

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>

#define MAX 630

float rj[MAX], retraso[MAX], costo[MAX], Ej[MAX], Lj[MAX], cj[MAX];
int secuencia[MAX], tipo[MAX];

int leer_archivo(){

   FILE *archivo;
         FILE *temp;
         int num;
         int iF,iC;
         char c=0;
         int comas=0, renglones=0;

         archivo = fopen("leer.csv","r+");
         temp = fopen("temp.txt","w");
         if(archivo == NULL){
                  printf("Error, no se puede abrir el archivo\n");
         }
         else{

                  while(c!='\n'){
      c = fgetc(archivo);
                           if(c==',' || c=='\n'){
                                    comas++;
                                    fprintf(temp," ");
                           }else
                                    fprintf(temp,"%c",c);
   }

                  while(c!=EOF){
                           c = fgetc(archivo);

                           if(c==',' || c=='\n'){
            if(c=='\n')
                  renglones++;
                                    fprintf(temp," ");
      }
                           else
                                    fprintf(temp,"%c",c);
                  }
                  rewind(archivo);
                  fclose(archivo);
                  fclose(temp);

                  printf("\ncolumnas: %d, renglones: %d\n",comas,renglones+1);
   temp = fopen("temp.txt","r");
```

```
                for(iF=0;iF<3;iF++){
                        for(iC=0;iC<630;iC++){
            if(iF==0){
                                        fscanf(temp,"%d",&secuencia[iC]);
                    }
            if(iF==1){
                                        fscanf(temp,"%d",&tipo[iC]);
                    }
            if(iF==2){
                                        fscanf(temp,"%f",&rj[iC]);
                    }
            }
                      }
                      fclose(temp);
          }

          return 0;
}

void calcular_cj(){
     int i;
     i=0;

     if(tipo[i]==1)
          cj[i]=2;
     else
          cj[i]=2.57;

     i++;

     for(i=1; i<MAX; i++){
          if(cj[i-1]>=rj[i]){
               if(tipo[i]==1)
                    cj[i]=cj[i-1]+2;
               else
                    cj[i]=cj[i-1]+2.57;
                        }
          else{
               if(tipo[i]==1)
                    cj[i]=rj[i]+2;
               else
                    cj[i]=rj[i]+2.57;
                        }
                    }
}

void calcular_retraso(){
           srand(time(NULL));
           int i, a, b, c;
           i=0;
           a=-60;
           c=240;
           float r, r2, x;
           b=0;

           while(i < MAX){
```

```
            r2=1+ (float) (rand()%10000);
            r = r2/10000;

            if(r<= ((b-a)/(c-a))){
                x= a + sqrt((c-a)*(b-a)*r);
                if(rj[i]==0){
                    retraso[i]=0;
                    i++;
                    }
                else{
                    if((x <= rj[i]) && (x >= 0)){
                        retraso[i]=x;
                        i++;
                            }
                     }
                }
            else{
                x= c - sqrt((c-a)*(c-b)*(1-r));
                if(rj[i]==0){
                    retraso[i]=0;
                    i++;
                    }
                else{
                    if((x <= rj[i]) && (x >= 0)){
                        retraso[i]=x;
                        i++;
                                }
                        }
                }
            }
}

void calcular_rango(){
    int i;
    i=0;

    for(i=0; i<MAX; i++){
        if(fabs(retraso[i]) > 0){
            Lj[i] = cj[i] - fabs(retraso[i]);
            Ej[i] = Lj[i] - 20;
                    }
        if(fabs(retraso[i])<0){
            Ej[i] = fabs(retraso[i]) + cj[i];
            Lj[i] = Ej[i] + 20;
                    }
        if(fabs(retraso[i]) == 0){
            Lj[i] = cj[i] + 17;
            Ej[i] = Lj[i] - 30;
                  }
            }
}

void calcular_costo(){
    int i;
    i=0;
```

```
    for(i=0; i<MAX; i++){
        if(cj[i] <= Lj[i] && cj[i] >= Ej[i])
                costo[i]=0;
        if(fabs(retraso[i]) <= 60 && fabs(retraso[i])> 0)
                costo[i]=(200/3)*fabs(retraso[i]);
        if(fabs(retraso[i]) <= 120 && fabs(retraso[i])> 60)
                costo[i]=(250/3)*fabs(retraso[i]);
        if(fabs(retraso[i]) <= 180 && fabs(retraso[i])> 120)
                costo[i]=(300/3)*fabs(retraso[i]);
        if(fabs(retraso[i]) <= 240 && fabs(retraso[i])> 180)
                costo[i]=(350/3)*fabs(retraso[i]);
           }
}


main(){
   FILE *archivo2;
           int i, j;
           char c=0;
           int renglones=0;
           int comas=0;
           i=0;
           j=0;

      leer_archivo();
      calcular_cj();
      calcular_retraso();
      calcular_rango();
      calcular_costo();

           archivo2 = fopen("leer.csv","r+");
           if(archivo2 == NULL){
                   printf("Error, no se puede abrir el archivo\n");
           }
           else{
               while(c!='\n'){
         c = fgetc(archivo2);
                                   if(c==',' || c=='\n'){
                                           comas++;
                           }
                   }

        while(c!=EOF){
          c = fgetc(archivo2);
                                   if(c==',' || c=='\n'){
                            printf("todo bien");
                 if(c=='\n')
                         renglones++;
                                           printf("%d", renglones);
                    }
                        for(j=0; j<5; j++){
              i=0;
              for(i=0; i<MAX; i++){
                  if(j==0){
                     fprintf(archivo2, "%f", cj[i]);
```

```
                    fprintf(archivo2, ",");
                    }
                if(j==1){
                    fprintf(archivo2, "%f", Ej[i]);
                    fprintf(archivo2, ",");
                     }
                if(j==2){
                    fprintf(archivo2, "%f", Lj[i]);
                    fprintf(archivo2, ",");
                     }
                if(j==3){
                    fprintf(archivo2, "%f", retraso[i]);
                    fprintf(archivo2, ",");
                     }
                if(j==4){
                    fprintf(archivo2, "%f", costo[i]);
                    fprintf(archivo2, ",");
                     }
                          }
                fprintf(archivo2, "\n");
            }
            }
   fclose(archivo2);
   }

     fflush(stdin);
     getchar();
     return 0;

}
```

ANEXO B

Resultados de la simulación

Vuelo (j)	tiempo entre eventos	aterrizajes/ despegues	llegadas a la cola (rj)	Cj	dj		Rj	costo retraso
					Ej	Lj		
1	0	1	0	2	-11	19	0	0
2	1.728096741	1	1.728096741	4	-9	21	0	0
3	0.62373869	0	2.351835431	6.57	-6.43	23.57	0	0
4	1.274724226	0	3.626559657	9.14	-3.860001	26.139999	0	0
5	1.260696948	0	4.887256605	11.71	-1.290001	28.709999	0	0
6	2.534193306	0	7.421449912	14.28	1.279999	31.279999	0	0
7	1.460180099	1	8.881630011	16.279999	3.279999	33.279999	0	0
8	1.526267338	1	10.40789735	18.279999	5.279999	35.279999	0	0
9	0.044396533	1	10.45229388	20.279999	7.279999	37.279999	0	0
10	2.957598163	1	13.40989205	22.279999	9.279999	39.279999	0	0
11	1.311920154	0	14.7218122	24.849998	11.849998	41.849998	0	0
12	2.719120382	1	17.44093258	26.849998	13.849998	43.849998	0	0
13	2.284147279	1	19.72507986	28.849998	15.849998	45.849998	0	0
14	1.889701218	0	21.61478108	31.419998	18.419998	48.419998	0	0
15	0.901940603	1	22.51672168	33.419998	20.419998	50.419998	0	0
16	0.166441053	0	22.68316273	35.989998	22.989998	52.989998	0	0
17	4.213838355	1	26.89700109	37.989998	24.989998	54.989998	0	0
18	2.936475339	1	29.83347643	39.989998	26.989998	56.989998	0	0
19	0.536742252	0	30.37021868	42.559998	29.559998	59.559998	0	0
20	0.919235552	1	31.28945423	44.559998	31.559998	61.559998	0	0
21	7.410150962	0	38.69960519	47.129997	34.129997	64.129997	0	0
22	2.493957437	1	41.19356263	49.129997	36.129997	66.129997	0	0
23	2.122690822	1	43.31625345	51.129997	38.129997	68.129997	0	0
24	0.039421225	1	43.35567468	53.129997	40.129997	70.129997	0	0
25	1.234111482	1	44.58978616	55.129997	42.129997	72.129997	0	0
26	2.989294301	0	47.57908046	57.699997	0.265541	20.265541	37.434456	2470.674072
27	8.857198223	0	56.43627868	60.269997	47.269997	77.269997	0	0
28	3.590244823	1	60.0265235	62.269997	-12.059326	7.940674	54.329323	3585.735352
29	0.520045362	0	60.54656887	64.839996	-13.48777	6.51223	58.327766	3849.632568
30	0.129040193	1	60.67560906	66.839996	53.839996	83.839996	0	0
31	0.111986933	1	60.78759599	68.839996	-9.765404	10.234596	58.6054	3867.956299
32	0.869022689	0	61.65661868	71.409996	8.517075	28.517075	42.892921	2830.932861
33	0.953918815	1	62.6105375	73.409996	60.409996	90.409996	0	0
34	3.04880303	0	65.65934053	75.979996	2.17345	22.17345	53.806545	3551.231934
35	2.47431787	1	68.1336584	77.979996	11.418358	31.418358	46.561638	3073.068115
36	1.69990338	0	69.83356178	80.549995	67.549995	97.549995	0	0
37	2.093721359	0	71.92728314	83.119995	18.539547	38.539547	44.580448	2942.30957
38	0.985874263	1	72.9131574	85.119995	72.119995	102.119995	0	0
39	1.225245383	1	74.13840278	87.119995	19.326134	39.326134	47.793861	3154.394775
40	3.369708402	1	77.50811118	89.119995	76.119995	106.119995	0	0
41	0.510411704	1	78.01852289	91.119995	78.119995	108.119995	0	0
42	0.284587492	0	78.30311038	93.689995	80.689995	110.689995	0	0
43	0.555465967	1	78.85857635	95.689995	38.202213	58.202213	37.487782	2474.193604
44	1.513928175	1	80.37250452	97.689995	32.241787	52.241787	45.448208	2999.581787
45	3.658036341	0	84.03054086	100.26	30.980732	50.980732	49.279263	3252.431396
46	4.831835836	0	88.8623767	102.82999	41.39283	61.39283	41.437164	2734.852783
47	2.34670809	1	91.20908479	104.82999	-3.808189	16.191811	88.638184	7356.969238
48	0.742899464	1	91.95198425	106.82999	36.072693	56.072693	50.757301	3349.981934
49	0.519168436	1	92.47115269	108.82999	3.9506	23.9506	84.879395	7044.989746
50	1.085239448	1	93.55639214	110.82999	97.829994	127.829994	0	0
51	2.275030049	1	95.83142218	112.82999	53.215622	73.215622	39.614368	2614.54834
52	2.75886107	1	98.59028325	114.82999	101.829987	131.829987	0	0
53	0.154101828	1	98.74438508	116.82999	57.986649	77.986649	38.843342	2563.660645
54	2.104639998	0	100.8490251	119.39999	47.3255	67.3255	52.07449	3436.91626
55	0.022807554	1	100.8718326	121.39999	7.015923	27.015923	94.384071	7833.87793
56	0.445665919	0	101.3174986	123.96999	110.970001	140.970001	0	0
57	0.159565034	0	101.4770636	126.53999	22.908791	42.908791	83.631203	6941.389648
58	3.026922114	1	104.5039857	128.53999	60.108261	80.108261	48.431736	3196.494629
59	3.107713676	0	107.6116994	131.11	32.12925	52.12925	78.980751	6555.402344

60	5.537808226	1	113.1495076	133.11	120.110001	150.110001	0	0
61	3.763619669	0	116.9131273	135.68001	122.680008	152.680008	0	0
62	4.424936494	1	121.3380638	137.68001	85.924095	105.924095	31.755915	2095.890381
63	6.649564304	1	127.9876281	139.68001	39.23941	59.23941	80.440598	6676.569824
64	0.088949481	1	128.0765775	141.68001	74.50325	94.50325	47.176762	3113.66626
65	1.348436061	0	129.4250136	144.25002	63.869606	83.869606	60.380409	5011.57373
66	1.243588779	0	130.6686024	146.82002	133.820023	163.820023	0	0
67	0.161595882	1	130.8301983	148.82002	94.034058	114.034058	34.785965	2295.873779
68	1.420603936	1	132.2508022	150.82002	69.957954	89.957954	60.862068	5051.551758
69	1.496345725	1	133.7471479	152.82002	81.891441	101.891441	50.928581	3361.286377
70	0.216842534	1	133.9639905	154.82002	103.323257	123.323257	31.496763	2078.786377
71	0.742606573	1	134.706597	156.82002	55.065407	75.065407	81.754616	6785.633301
72	2.586281969	1	137.292879	158.82002	1.733566	21.733566	137.086456	13708.64551
73	1.030087621	1	138.3229666	160.82002	147.820023	177.820023	0	0
74	4.279023925	1	142.6019906	162.82002	75.156906	95.156906	67.663116	5616.038574
75	0.609789951	0	143.2117805	165.39003	51.917847	71.917847	93.472183	7758.191406
76	3.076723601	0	146.2885041	167.96004	94.211487	114.211487	53.748554	3547.404541
77	0.15928427	0	146.4477884	170.53005	112.66861	132.66861	37.861435	2498.854736
78	2.419512507	0	148.8673009	173.10005	57.224998	77.224998	95.875053	7957.629395
79	0.959164712	0	149.8264656	175.67006	162.670059	192.670059	0	0
80	2.790572027	1	152.6170376	177.67006	96.264572	116.264572	61.405483	5096.655273
81	1.167945948	0	153.7849836	180.24007	167.240067	197.240067	0	0
82	3.677162571	0	157.4621461	182.81007	169.810074	199.810074	0	0
83	4.655908165	0	162.1180543	185.38008	60.613213	80.613213	104.766869	8695.650391
84	3.205226288	0	165.3232806	187.95009	69.14782	89.14782	98.802269	8200.587891
85	8.259982549	0	173.5832631	190.5201	81.28614	101.28614	89.233955	7406.418457
86	6.015988984	1	179.5992521	192.5201	39.884888	59.884888	132.635208	13263.52051
87	0.861462683	0	180.4607148	195.0901	142.259598	162.259598	32.830505	2166.813477
88	3.637944032	0	184.0986588	197.66011	44.149551	64.149551	133.510559	13351.05566
89	1.244742748	0	185.3434016	200.23012	187.230118	217.230118	0	0
90	0.118575362	1	185.461977	202.23012	189.230118	219.230118	0	0
91	0.154225424	0	185.6162024	204.80013	28.499588	48.499588	156.300537	15630.05371
92	2.966530212	0	188.5827326	207.37013	149.04512	169.04512	38.325012	2529.450684
93	1.888832861	0	190.4715655	209.94014	196.94014	226.94014	0	0
94	1.280822357	1	191.7523878	211.94014	149.794632	169.794632	42.145508	2781.603516
95	0.012014522	0	191.7644023	214.51015	48.194733	68.194733	146.315414	14631.54102
96	2.755790735	0	194.5201931	217.08015	87.736961	107.736961	109.343193	9075.485352
97	0.971322408	1	195.4915155	219.08015	206.080154	236.080154	0	0
98	0.418073852	0	195.9095893	221.65016	208.650162	238.650162	0	0
99	4.905134923	0	200.8147242	224.22017	105.188255	125.188255	99.031914	8219.648438
100	3.192952737	1	204.007677	226.22017	132.778259	152.778259	73.44191	6095.678711
101	1.177148083	0	205.1848251	228.79018	215.790176	245.790176	0	0
102	3.284119419	0	208.4689445	231.36018	218.360184	248.360184	0	0
103	7.265943694	1	215.7348882	233.36018	84.22226	104.22226	129.137924	12913.79199
104	2.458839052	1	218.1937272	235.36018	63.745697	83.745697	151.614487	15161.44824
105	0.601041177	0	218.7947684	237.93019	177.415527	197.415527	40.51466	2673.967529
106	1.466000338	1	220.2607687	239.93019	150.397781	170.397781	69.53241	5771.189941
107	0.902341346	1	221.1631101	241.93019	228.930176	258.930176	0	0
108	0.312162707	1	221.4752728	243.93019	46.168442	66.168442	177.761749	17776.17578
109	0.926770111	0	222.4020429	246.5002	128.714111	148.714111	97.786079	8116.244629
110	1.121669997	1	223.5237129	248.5002	235.500183	265.500183	0	0
111	2.000246564	0	225.5239595	251.07021	238.07019	268.07019	0	0
112	0.875422811	0	226.3993823	253.64021	150.698151	170.698151	82.942055	6884.19043
113	1.268262597	1	227.6676449	255.64021	133.012695	153.012695	102.627518	8518.083984
114	1.040579073	1	228.708224	257.6402	244.640198	274.640198	0	0
115	1.68579856	1	230.3940225	259.6402	246.640198	276.640198	0	0
116	5.74822036	0	236.1422429	262.21021	174.191589	194.191589	68.018608	5645.544434
117	2.91972888	1	239.0619718	264.21021	151.985718	171.985718	92.224487	7654.632324
118	2.581214216	1	241.643186	266.21021	209.308212	229.308212	36.901993	2435.531494
119	4.394664461	1	246.0378504	268.21021	255.210205	285.210205	0	0
120	4.585709484	1	250.6235599	270.21021	120.61673	140.61673	129.593475	12959.34766
121	2.501384303	0	253.1249442	272.78021	45.643555	65.643555	207.136658	24027.85156
122	2.83419743	1	255.9591417	274.78021	203.508591	223.508591	51.271622	3383.927002
123	1.545231103	1	257.5043728	276.78021	224.348999	244.348999	32.431217	2140.460449
124	1.876773863	1	259.3811466	278.78021	192.945648	212.945648	65.834564	5464.269043
125	1.57369925	0	260.9548459	281.35022	268.35022	298.35022	0	0
126	0.410101348	0	261.3649472	283.92023	132.153305	152.153305	131.766922	13176.69238

127	0.966520005	1	262.3314672	285.92023	193.233124	213.233124	72.687111	6033.030273
128	0.572866984	1	262.9043342	287.92023	210.836594	230.836594	57.08363	3767.519531
129	3.164494743	0	266.0688289	290.49023	218.204926	238.204926	52.285316	3450.830811
130	2.283713557	0	268.3525425	293.06024	226.275146	246.275146	46.785095	3087.816406
131	5.281312751	0	273.6338553	295.63025	282.630249	312.630249	0	0
132	3.272290461	0	276.9061457	298.20026	157.899872	177.899872	120.300377	12030.03809
133	3.075533311	1	279.981679	300.20026	202.400146	222.400146	77.800117	6457.409668
134	0.224178758	1	280.2058578	302.20026	121.799255	141.799255	160.401001	16040.09961
135	4.522610062	1	284.7284678	304.20026	230.548309	250.548309	53.651943	3541.02832
136	1.861462179	1	286.58993	306.20026	177.516663	197.516663	108.683594	9020.738281
137	5.083229331	1	291.6731594	308.20026	83.696716	103.696716	204.50354	23722.41016
138	1.352111336	0	293.0252707	310.77026	297.770264	327.770264	0	0
139	8.446651604	0	301.4719223	313.34027	188.865921	208.865921	104.47435	8671.371094
140	1.413417854	1	302.8853401	315.34027	302.340271	332.340271	0	0
141	1.909870249	0	304.7952104	317.91028	175.332596	195.332596	122.577682	12257.76856
142	1.030676771	0	305.8258872	320.48029	140.214844	160.214844	160.265442	16026.54395
143	0.975358748	0	306.8012459	323.05029	249.050293	269.050293	54.000004	3564.000244
144	1.812740817	0	308.6139867	325.6203	249.420197	269.420197	56.200108	3709.207031
145	2.399166546	0	311.0131533	328.19031	146.558685	166.558685	161.631622	16163.16211
146	0.605612575	0	311.6187659	330.76032	250.139252	270.139252	60.621071	5031.548828
147	5.761726436	0	317.3804923	333.33032	320.330322	350.330322	0	0
148	7.980072891	0	325.3605652	335.90033	322.90033	352.90033	0	0
149	0.757853389	0	326.1184186	338.47034	269.7565	289.7565	48.713829	3215.112793
150	1.950123905	1	328.0685425	340.47034	165.323166	185.323166	155.147171	15514.7168
151	3.499281813	0	331.5678243	343.04034	330.040344	360.040344	0	0
152	2.956576187	1	334.5244005	345.04034	332.040344	362.040344	0	0
153	0.358381166	0	334.8827816	347.61035	138.733734	158.733734	188.876617	21909.6875
154	0.151444743	1	335.0342264	349.61035	131.610367	151.610367	197.999985	22967.99805
155	1.879336283	1	336.9135627	351.61035	184.173843	204.173843	147.436508	14743.65039
156	1.315075119	0	338.2286378	354.18036	263.461182	283.461182	70.71917	5869.690918
157	2.070173853	1	340.2988116	356.18036	343.180359	373.180359	0	0
158	5.126661089	1	345.4254727	358.18036	298.332306	318.332306	39.848057	2629.97168
159	0.360286244	0	345.785759	360.75037	307.989349	327.989349	32.761005	2162.226318
160	0.169045183	0	345.9548042	363.32037	205.108383	225.108383	138.21199	13821.19922
161	4.265396931	1	350.2202011	365.32037	352.320374	382.320374	0	0
162	3.343942089	0	353.5641432	367.89038	354.890381	384.890381	0	0
163	0.758587474	1	354.3227306	369.89038	242.271637	262.271637	107.618736	8932.355469
164	2.723934074	0	357.0466647	372.46039	359.460388	389.460388	0	0
165	2.473893295	0	359.520558	375.0304	235.509445	255.509445	119.52095	9920.239258
166	1.216388323	0	360.7369463	377.6004	364.600403	394.600403	0	0
167	4.911591722	1	365.6485381	379.6004	329.188568	349.188568	30.411833	2007.180908
168	0.0361585	1	365.6846966	381.6004	368.600403	398.600403	0	0
169	1.848631763	1	367.5333283	383.6004	370.600403	400.600403	0	0
170	2.118703365	0	369.6520317	386.17041	373.17041	403.17041	0	0
171	0.823683256	0	370.4757149	388.74042	321.619659	341.619659	47.120762	3109.970215
172	1.182541632	0	371.6582566	391.31043	287.656189	307.656189	83.654228	6943.300781
173	1.95845689	1	373.6167135	393.31043	294.887665	314.887665	78.422768	6509.089844
174	2.577025573	0	376.193739	395.88043	213.278091	233.278091	162.602341	16260.23438
175	0.802156801	1	376.9958958	397.88043	384.880432	414.880432	0	0
176	3.155308685	1	380.1512045	399.88043	253.469177	273.469177	126.41127	12641.12695
177	2.818009564	0	382.9692141	402.45044	347.225433	367.225433	35.225002	2324.850098
178	3.342107156	1	386.3113212	404.45044	232.550385	252.550385	151.900055	15190.00586
179	1.460408284	1	387.7717295	406.45044	393.450439	423.450439	0	0
180	3.203734779	0	390.9754643	409.02045	396.020447	426.020447	0	0
181	1.112438999	0	392.0879033	411.59045	356.909729	376.909729	34.680737	2288.928711
182	6.808543262	1	398.8964466	413.59045	299.946198	319.946198	93.644264	7772.474121
183	7.484468225	0	406.3809148	416.16046	296.334625	316.334625	99.825829	8285.543945
184	7.95284983	1	414.3337646	418.16046	239.104889	259.104889	159.055573	15905.55762
185	0.943887326	1	415.277652	420.16046	243.298889	263.298889	156.861557	15686.15527
186	0.228344305	1	415.5059963	422.16046	286.694794	306.694794	115.465675	9583.651367
187	1.720973829	0	417.2269701	424.73047	231.812515	251.812515	172.917953	17291.79492
188	0.05859443	1	417.2855645	426.73047	346.069244	366.069244	60.661221	5034.881348
189	2.28841681	0	419.5739813	429.30048	318.482239	338.482239	90.818237	7537.913574
190	0.894906024	0	420.4688874	431.87048	361.11319	381.11319	50.757301	3349.981934
191	1.874220743	0	422.3431081	434.44049	339.936096	359.936096	74.504379	6183.863281
192	9.484588207	1	431.8276963	436.44049	320.839935	340.839935	95.600548	7934.845703
193	1.32616177	1	433.1538581	438.44049	276.297546	296.297546	142.14296	14214.2959

194	1.13167263	1	434.2855307	440.44049	366.517883	386.517883	53.9226	3558.891602
195	0.376295344	1	434.661826	442.44049	429.440491	459.440491	0	0
196	3.121406759	0	437.7832328	445.0105	355.287903	375.287903	69.72258	5786.974121
197	1.93928474	0	439.7225175	447.58051	434.580505	464.580505	0	0
198	3.468462041	1	443.1909796	449.58051	436.580505	466.580505	0	0
199	0.170448738	0	443.3614283	452.15051	381.107635	401.107635	51.042866	3368.829102
200	0.177002845	0	443.5384312	454.72052	346.10614	366.10614	88.614395	7354.994629
201	0.822480335	0	444.3609115	457.29053	349.789703	369.789703	87.500816	7262.567871
202	1.668602968	0	446.0295145	459.86054	446.860535	476.860535	0	0
203	0.890910342	0	446.9204248	462.43054	265.35907	285.35907	177.071472	17707.14648
204	1.22492783	1	448.1453526	464.43054	311.021362	331.021362	133.40918	13340.91797
205	0.377017086	0	448.5223697	467.00055	281.070801	301.070801	165.929764	16592.97656
206	3.534039091	1	452.0564088	469.00055	251.000549	271.000549	197.999985	22967.99805
207	1.353113596	1	453.4095224	471.00055	352.656464	372.656464	98.344078	8162.558594
208	1.757603692	1	455.1671261	473.00055	460.000549	490.000549	0	0
209	2.36889952	0	457.5360256	475.57056	413.897552	433.897552	41.672997	2750.417725
210	2.035659507	1	459.5716851	477.57056	362.889526	382.889526	94.681046	7858.526855
211	0.820788259	0	460.3924734	480.14056	467.140564	497.140564	0	0
212	3.758177793	0	464.1506512	482.71057	291.749695	311.749695	170.960876	17096.08789
213	2.266377303	1	466.4170285	484.71057	325.109772	345.109772	139.6008	13960.08008
214	2.001860329	1	468.4188888	486.71057	353.422852	373.422852	113.287727	9402.881836
215	0.618854666	1	469.0377435	488.71057	413.604065	433.604065	55.106518	3637.030273
216	3.894626486	0	472.93237	491.28058	421.963562	441.963562	49.317017	3254.923096
217	0.279456698	0	473.2118267	493.85059	418.412781	438.412781	55.437813	3658.895752
218	2.306599153	0	475.5184258	496.42059	327.888611	347.888611	148.531982	14853.19824
219	1.197750663	0	476.7161765	498.9906	415.25177	435.25177	63.738827	5290.322754
220	5.966897498	0	482.683074	501.56061	488.560608	518.560608	0	0
221	1.435329627	0	484.1184036	504.13062	491.130615	521.130615	0	0
222	1.211537512	0	485.3299411	506.70062	493.700623	523.700623	0	0
223	1.206732969	0	486.5366741	509.27063	433.579071	453.579071	55.691559	3675.642822
224	0.631215279	0	487.1678894	511.84064	444.364929	464.364929	47.475712	3133.396973
225	0.838461786	1	488.0063512	513.84064	462.843994	482.843994	30.996651	2045.778931
226	0.251066879	0	488.257418	516.41065	410.389862	430.389862	86.020782	7139.725098
227	1.526715864	1	489.7841339	518.41065	437.669128	457.669128	60.74152	5041.546387
228	1.316644142	0	491.100778	520.98065	507.980652	537.980652	0	0
229	1.069439719	1	492.1702178	522.98065	438.771423	458.771423	64.209213	5329.364746
230	2.116709261	1	494.286927	524.98065	416.627625	436.627625	88.353035	7333.301758
231	2.179954531	1	496.4668816	526.98065	414.414703	434.414703	92.565941	7682.973145
232	3.275550169	0	499.7424317	529.55066	403.634216	423.634216	105.916443	8791.064453
233	7.82511471	1	507.5675464	531.55066	330.826294	350.826294	180.72438	20964.02734
234	0.944789171	1	508.5123356	533.55066	425.268829	445.268829	88.281837	7327.392578
235	2.092764805	0	510.6051004	536.12067	402.320496	422.320496	113.800163	9445.413086
236	0.642522073	0	511.2476225	538.69067	435.17453	455.17453	83.516136	6931.839355
237	1.404299773	1	512.6519223	540.69067	441.575714	461.575714	79.114952	6566.541016
238	0.68496031	1	513.3368826	542.69067	422.839172	442.839172	99.851501	8287.674805
239	0.314023037	1	513.6509056	544.69067	301.447754	321.447754	223.242935	25896.17969
240	1.724165652	0	515.3750713	547.26068	376.2146	396.2146	151.046082	15104.6084
241	0.793812611	0	516.1688839	549.83069	536.830688	566.830688	0	0
242	0.022387445	1	516.1912713	551.83069	538.830688	568.830688	0	0
243	0.307130897	0	516.4984022	554.4007	473.458221	493.458221	60.942471	5058.225098
244	0.506972447	1	517.0053747	556.4007	401.186188	421.186188	135.214508	13521.45117
245	1.825273778	1	518.8306484	558.4007	481.572754	501.572754	56.827946	3750.644531
246	0.983509194	0	519.8141576	560.9707	492.708008	512.708008	48.26268	3185.336914
247	1.154723756	0	520.9688814	563.54071	437.167053	457.167053	106.37365	8829.012695
248	0.449192354	1	521.4180737	565.54071	436.472748	456.472748	109.067955	9052.640625
249	3.282138167	1	524.7002119	567.54071	490.830658	510.830658	56.710064	3742.864258
250	5.117530514	1	529.8177424	569.54071	556.54071	586.54071	0	0
251	0.016386647	1	529.8341291	571.54071	510.067261	530.067261	41.473427	2737.246094
252	0.636745176	0	530.4708742	574.11072	398.495544	418.495544	155.615158	15561.51563
253	2.213223672	0	532.6840979	576.68073	401.533569	421.533569	155.147171	15514.7168
254	0.159679147	0	532.8437771	579.25073	480.046265	500.046265	79.204483	6573.972168
255	0.543907116	1	533.3876842	581.25073	510.05542	530.05542	51.195335	3378.89209
256	2.993497594	0	536.3811818	583.82074	503.380188	523.380188	60.440544	5016.564941
257	3.218206719	0	539.5993885	586.39075	488.501831	508.501831	77.888924	6464.780762
258	4.048435482	0	543.647824	588.96075	533.366211	553.366211	35.594521	2349.238281
259	0.430006889	0	544.0778309	591.53076	578.530762	608.530762	0	0
260	1.215422014	1	545.2932529	593.53076	538.657043	558.657043	34.873699	2301.664063

261	0.675674547	0	545.9689274	596.10077	534.118958	554.118958	41.981815	2770.799805
262	0.653515441	0	546.6224429	598.67078	585.670776	615.670776	0	0
263	2.307789234	1	548.9302321	600.67078	487.787048	507.787048	92.88372	7709.348633
264	0.139949653	0	549.0701818	603.24078	473.759735	493.759735	109.481041	9086.926758
265	1.981408568	0	551.0515903	605.81079	592.810791	622.810791	0	0
266	4.96925317	1	556.0208435	607.81079	594.810791	624.810791	0	0
267	0.750791229	1	556.7716347	609.81079	528.888428	548.888428	60.922363	5056.556152
268	4.365576806	1	561.1372115	611.81079	529.069946	549.069946	62.740868	5207.492188
269	0.336342423	1	561.4735539	613.81079	546.85791	566.85791	46.952854	3098.888428
270	0.605101933	0	562.0786559	616.3808	488.162415	508.162415	108.218369	8982.125
271	11.90398757	0	573.9826434	618.95081	460.100189	480.100189	138.850616	13885.06152
272	6.706854094	0	580.6894975	621.52081	562.319214	582.319214	39.201591	2587.304932
273	0.487879728	0	581.1773773	624.09082	611.09082	641.09082	0	0
274	3.167676327	1	584.3450536	626.09082	613.09082	643.09082	0	0
275	0.713780552	1	585.0588341	628.09082	615.09082	645.09082	0	0
276	1.486004935	0	586.5448391	630.66083	617.660828	647.660828	0	0
277	1.076799516	1	587.6216386	632.66083	580.333618	600.333618	32.327179	2133.59375
278	0.973274167	1	588.5949128	634.66083	455.605255	475.605255	159.055573	15905.55762
279	0.48844027	0	589.083353	637.23084	624.230835	654.230835	0	0
280	0.162961722	0	589.2463148	639.80084	626.800842	656.800842	0	0
281	8.405979758	1	597.6522945	641.80084	533.590149	553.590149	88.21067	7321.48584
282	4.867967612	0	602.5202621	644.37085	494.548889	514.548889	129.82196	12982.19629
283	2.268813172	1	604.7890753	646.37085	633.37085	663.37085	0	0
284	0.073850231	1	604.8629255	648.37085	468.600525	488.600525	159.77034	15977.03418
285	4.237398633	0	609.1003242	650.94086	473.164948	493.164948	157.775909	15777.59082
286	1.158225631	0	610.2585498	653.51086	419.247986	439.247986	214.262894	24854.49609
287	0.11074759	0	610.3692974	656.08087	516.500122	536.500122	119.580734	9925.201172
288	5.191935214	0	615.5612326	658.65088	491.524963	511.524963	147.1259	14712.58984
289	3.070131919	0	618.6313645	661.22089	475.145203	495.145203	166.075699	16607.57031
290	1.144243705	0	619.7756082	663.79089	563.282593	583.282593	80.508316	6682.19043
291	0.362181925	0	620.1377901	666.3609	607.374329	627.374329	38.986568	2573.113525
292	2.450075297	1	622.5878654	668.3609	492.403748	512.403748	155.957153	15595.71484
293	0.639279296	1	623.2271447	670.3609	526.674255	546.674255	123.68663	12368.66309
294	5.988781906	0	629.2159266	672.93091	659.930908	689.930908	0	0
295	1.518958429	1	630.7348851	674.93091	590.250061	610.250061	64.680855	5368.510742
296	2.902092062	0	633.6369771	677.50092	450.799561	470.799561	206.701355	23977.35742
297	0.501897346	1	634.1388745	679.50092	666.500916	696.500916	0	0
298	8.130410671	0	642.2692852	682.07092	547.956604	567.956604	114.114342	9471.490234
299	5.58442997	1	647.8537151	684.07092	478.799316	498.799316	185.271591	21491.50391
300	1.875243891	0	649.728959	686.64093	499.978516	519.978516	166.66243	16666.24219
301	1.914973028	0	651.643932	689.21094	563.96405	583.96405	105.246887	8735.491211
302	0.839872474	0	652.4838045	691.78095	463.07312	483.07312	208.70784	24210.10938
303	0.019720234	0	652.5035248	694.35095	508.809021	528.809021	165.541946	16554.19531
304	4.444285881	1	656.9478106	696.35095	604.736206	624.736206	71.614723	5944.021973
305	0.069410251	0	657.0172209	698.92096	685.920959	715.920959	0	0
306	0.026941575	1	657.0441625	700.92096	625.053528	645.053528	55.867439	3687.250977
307	2.611135981	0	659.6552984	703.49097	690.490967	720.490967	0	0
308	0.503027013	1	660.1583255	705.49097	589.915344	609.915344	95.57563	7932.777344
309	1.393107985	0	661.5514334	708.06097	640.210938	660.210938	47.850056	3158.10376
310	1.283822695	0	662.8352561	710.63098	589.696411	609.696411	100.934547	8377.567383
311	0.510070949	1	663.3453271	712.63098	699.630981	729.630981	0	0
312	3.288532287	0	666.6338594	715.20099	582.847351	602.847351	112.353615	9325.349609
313	10.83282354	1	677.4666829	717.20099	650.229492	670.229492	46.9715	3100.119141
314	3.950312242	1	681.4169951	719.20099	589.775085	609.775085	109.42588	9082.347656
315	1.305444802	1	682.7224399	721.20099	554.42334	574.42334	146.777679	14677.76758
316	1.527765746	0	684.2502057	723.771	710.770996	740.770996	0	0
317	9.205200921	0	693.4554066	726.341	558.28009	578.28009	148.060898	14806.08984
318	0.274937768	0	693.7303444	728.91101	715.911011	745.911011	0	0
319	0.460416285	0	694.1907607	731.48102	628.973877	648.973877	82.507141	6848.092773
320	0.623060312	0	694.813821	734.05103	575.87439	595.87439	138.17662	13817.66211
321	5.178625429	1	699.9924464	736.05103	639.643188	659.643188	76.407829	6341.849609
322	2.188511293	1	702.1809577	738.05103	653.616394	673.616394	64.434631	5348.074219
323	2.704663523	1	704.8856212	740.05103	668.512146	688.512146	51.53886	3401.564697
324	2.078712621	0	706.9643338	742.62103	573.694641	593.694641	148.926407	14892.64063
325	1.656885812	1	708.6212197	744.62103	660.616455	680.616455	64.004547	5312.377441
326	0.839428721	1	709.4606484	746.62103	614.492859	634.492859	112.128181	9306.638672
327	4.175758247	0	713.6364066	749.19104	672.834167	692.834167	56.356869	3719.553223

328	1.785040266	0	715.4214469	751.76105	681.782104	701.782104	49.978947	3298.610596
329	1.698343355	0	717.1197903	754.33106	741.331055	771.331055	0	0
330	4.655334245	1	721.7751245	756.33106	517.459229	537.459229	218.871796	25389.12891
331	2.262006392	1	724.0371309	758.33106	618.480957	638.480957	119.85009	9947.557617
332	2.34647266	1	726.3836035	760.33106	629.575012	649.575012	110.75605	9192.751953
333	3.685202486	1	730.068806	762.33106	677.300781	697.300781	65.030289	5397.51416
334	1.395868578	1	731.4646746	764.33106	751.331055	781.331055	0	0
335	2.843810623	0	734.3084852	766.90106	627.618896	647.618896	119.282135	9900.416992
336	4.730335687	0	739.0388209	769.47107	713.206177	733.206177	36.264877	2393.481934
337	0.667691021	1	739.7065119	771.47107	758.471069	788.471069	0	0
338	0.114294926	1	739.8208069	773.47107	760.471069	790.471069	0	0
339	1.942670571	1	741.7634774	775.47107	762.471069	792.471069	0	0
340	1.849695136	1	743.6131726	777.47107	542.354797	562.354797	215.116272	24953.48828
341	1.726072201	1	745.3392448	779.47107	711.920532	731.920532	47.550526	3138.334717
342	2.282768257	0	747.622013	782.04108	666.490356	686.490356	95.550705	7930.708496
343	1.824977574	0	749.4469906	784.61108	669.433716	689.433716	95.177353	7899.720215
344	0.062402826	0	749.5093934	787.18109	610.362793	630.362793	156.818283	15681.82813
345	0.50078282	1	750.0101763	789.18109	776.181091	806.181091	0	0
346	2.800763283	1	752.8109395	791.18109	594.792419	614.792419	176.388672	17638.86719
347	1.937505638	1	754.7484452	793.18109	550.77655	570.77655	222.404556	25798.92773
348	0.810415805	1	755.558861	795.18109	677.597412	697.597412	97.583702	8099.447266
349	3.944390763	0	759.5032517	797.7511	705.257507	725.257507	72.493584	6016.967285
350	1.41769445	0	760.9209462	800.32111	707.203125	727.203125	73.118011	6068.794922
351	1.858554977	0	762.7795012	802.89111	751.497925	771.497925	31.39319	2071.950684
352	0.931919364	1	763.7114205	804.89111	740.56842	760.56842	44.322716	2925.299316
353	0.144455645	0	763.8558762	807.46112	724.98407	744.98407	62.477051	5185.595215
354	3.004752321	0	766.8606285	810.03113	739.501953	759.501953	50.529156	3334.924316
355	0.108056631	1	766.9686851	812.03113	699.269714	719.269714	92.761414	7699.197266
356	3.433848023	1	770.4025332	814.03113	715.340759	735.340759	78.690361	6531.299805
357	1.048225343	1	771.4507585	816.03113	726.583191	746.583191	69.44796	5764.180664
358	1.014386097	1	772.4651446	818.03113	591.437744	611.437744	206.593399	23964.83398
359	3.180873402	0	775.646018	820.60114	807.601135	837.601135	0	0
360	0.698715638	1	776.3447336	822.60114	592.601135	612.601135	210.000015	24360.00195
361	2.012133523	1	778.3568672	824.60114	742.347046	762.347046	62.25412	5167.091797
362	0.973400947	1	779.3302681	826.60114	774.169922	794.169922	32.431217	2140.460449
363	1.285418145	1	780.6156862	828.60114	700.628418	720.628418	107.972733	8961.736328
364	1.141438478	0	781.7571247	831.17114	671.999695	691.999695	139.171432	13917.14356
365	0.160295509	0	781.9174202	833.74115	628.600952	648.600952	185.140167	21476.25977
366	2.35860638	1	784.2760266	835.74115	670.874756	690.874756	144.866409	14486.64063
367	2.677609237	0	786.9536358	838.31116	760.339722	780.339722	57.971428	3826.114258
368	0.212364474	1	787.1660003	840.31116	682.063843	702.063843	138.247345	13824.73438
369	2.501005792	0	789.6670061	842.88117	762.17981	782.17981	60.701366	5038.213379
370	0.198023648	1	789.8650298	844.88117	831.881165	861.881165	0	0
371	0.396062764	0	790.2610925	847.45117	834.451172	864.451172	0	0
372	1.730352272	1	791.9914448	849.45117	703.640503	723.640503	125.810677	12581.06738
373	3.829550537	0	795.8209953	852.02118	740.792114	760.792114	91.229034	7572.009766
374	4.505703494	0	800.3266988	854.59119	666.691101	686.691101	167.90007	16790.00781
375	3.726276869	1	804.0529757	856.59119	660.709839	680.709839	175.881363	17588.13672
376	2.078483367	0	806.1314591	859.16119	846.161194	876.161194	0	0
377	2.380965647	1	808.5124247	861.16119	713.50708	733.50708	127.654106	12765.41016
378	2.113129665	0	810.6255544	863.7312	850.731201	880.731201	0	0
379	0.220194038	1	810.8457484	865.7312	756.114746	776.114746	89.616486	7438.168457
380	11.01500969	0	821.8607581	868.30121	808.686829	828.686829	39.614368	2614.54834
381	0.103589368	1	821.9643475	870.30121	772.078857	792.078857	78.222374	6492.457031
382	0.298566429	1	822.2629139	872.30121	669.600403	689.600403	182.70079	21193.29102
383	0.643819234	0	822.9067331	874.87122	799.93988	819.93988	54.931358	3625.469727
384	2.710012795	0	825.6167459	877.44122	864.441223	894.441223	0	0
385	1.510859344	1	827.1276053	879.44122	818.058472	838.058472	41.382782	2731.263672
386	0.045333024	1	827.1729383	881.44122	868.441223	898.441223	0	0
387	5.68361808	1	832.8565564	883.44122	716.160156	736.160156	147.281067	14728.10645
388	2.026466616	1	834.883023	885.44122	758.9328	778.9328	106.508423	8840.199219
389	3.681572397	1	838.5645954	887.44122	825.423035	845.423035	42.018181	2773.199951
390	0.090090228	1	838.6546856	889.44122	772.714355	792.714355	96.726837	8028.327637
391	0.24056572	0	838.8952513	892.01123	879.01123	909.01123	0	0
392	3.075356855	1	841.9706082	894.01123	781.054077	801.054077	92.957146	7715.442871
393	2.251130717	1	844.2217389	896.01123	827.447876	847.447876	48.563324	3205.179443
394	2.053772209	1	846.2755111	898.01123	710.559143	730.559143	167.452103	16745.21094

395	2.5147228	0	848.7902339	900.58124	887.581238	917.581238	0	0
396	3.589480193	1	852.3797141	902.58124	889.581238	919.581238	0	0
397	2.514570787	0	854.8942849	905.15125	812.507141	832.507141	72.644089	6029.459473
398	1.664921707	1	856.5592066	907.15125	894.151245	924.151245	0	0
399	2.924800817	0	859.4840074	909.72125	896.721252	926.721252	0	0
400	0.471288699	0	859.9552961	912.29126	815.663208	835.663208	76.628029	6360.126465
401	1.351826348	1	861.3071225	914.29126	838.833923	858.833923	55.457325	3660.18335
402	0.836735807	0	862.1438583	916.86127	849.871094	869.871094	46.990158	3101.350342
403	0.630582923	0	862.7744412	919.43127	906.431274	936.431274	0	0
404	1.077753056	0	863.8521943	922.00128	814.264221	834.264221	87.737061	7282.175781
405	2.506863601	1	866.3590579	924.00128	745.079041	765.079041	158.922256	15892.22559
406	1.250307231	1	867.6093651	926.00128	804.989075	824.989075	101.01223	8384.014648
407	4.220568457	0	871.8299336	928.57129	864.983093	884.983093	43.588184	2876.820068
408	0.003178676	0	871.8331122	931.1413	838.281982	858.281982	72.859337	6047.325195
409	1.274465397	0	873.1075776	933.7113	789.776794	809.776794	123.934502	12393.4502
410	1.838309372	0	874.945887	936.28131	923.281311	953.281311	0	0
411	4.873786791	1	879.8196738	938.28131	759.75769	779.75769	158.523636	15852.36328
412	1.253908967	1	881.0735828	940.28131	927.281311	957.281311	0	0
413	7.03997257	0	888.1135553	942.85132	742.126953	762.126953	180.72438	20964.02734
414	1.53477687	0	889.6483322	945.42133	843.712219	863.712219	81.709129	6781.85791
415	0.64080164	1	890.2891338	947.42133	799.285522	819.285522	128.135803	12813.58008
416	2.829665097	1	893.1187989	949.42133	936.421326	966.421326	0	0
417	3.912532543	0	897.0313315	951.99133	892.52063	912.52063	39.470703	2605.066406
418	3.280581634	1	900.3119131	953.99133	823.402283	843.402283	110.589035	9178.889648
419	0.661178193	1	900.9730913	955.99133	796.318787	816.318787	139.672531	13967.25293
420	1.122485759	1	902.0955771	957.99133	865.196594	885.196594	72.794739	6041.963379
421	0.583048168	1	902.6786252	959.99133	946.991333	976.991333	0	0
422	0.638785234	1	903.3174105	961.99133	904.379089	924.379089	37.612255	2482.408936
423	6.578762043	1	909.8961725	963.99133	783.861206	803.861206	160.130112	16013.01074
424	0.756261797	0	910.6524343	966.56134	953.56134	983.56134	0	0
425	1.044310343	1	911.6967446	968.56134	824.998474	844.998474	123.562889	12356.28906
426	2.209009966	1	913.9057546	970.56134	813.194702	833.194702	137.366669	13736.66699
427	1.923008817	1	915.8287634	972.56134	959.56134	989.56134	0	0
428	1.064489189	0	916.8932526	975.13135	913.476501	933.476501	41.654846	2749.219727
429	0.38034199	0	917.2735946	977.70136	964.701355	994.701355	0	0
430	3.076045189	0	920.3496398	980.27136	899.147827	919.147827	61.123505	5073.250977
431	0.695146443	1	921.0447862	982.27136	883.46936	903.46936	78.801987	6540.564941
432	0.172290209	1	921.2170765	984.27136	852.873047	872.873047	111.398285	9246.057617
433	0.356484507	1	921.573561	986.27136	814.860291	834.860291	151.411057	15141.10547
434	8.242375484	1	929.8159364	988.27136	821.570862	841.570862	146.700485	14670.04883
435	0.02607599	1	929.8420124	990.27136	873.091553	893.091553	97.17984	8065.926758
436	1.344786796	1	931.1867992	992.27136	825.879028	845.879028	146.392319	14639.23145
437	1.761761372	1	932.9485606	994.27136	858.342285	878.342285	115.929047	9622.111328
438	7.201292416	1	940.149853	996.27136	983.271362	1013.271362	0	0
439	1.938906981	1	942.08876	998.27136	985.271362	1015.271362	0	0
440	7.235937476	0	949.3246975	1000.8414	896.702881	916.702881	84.138519	6983.49707
441	3.158576103	1	952.4832736	1002.8414	841.795898	861.795898	141.045471	14104.54688
442	5.225449354	1	957.7087229	1004.8414	927.580505	947.580505	57.260838	3779.215332
443	0.959145528	0	958.6678685	1007.4114	896.327454	916.327454	91.083908	7559.964355
444	1.82156678	1	960.4894352	1009.4114	815.465881	835.465881	173.94548	17394.54883
445	0.599141823	0	961.0885771	1011.9814	998.981445	1028.981445	0	0
446	0.128531897	1	961.217109	1013.9814	948.051453	968.051453	45.929913	3031.374268
447	0.529756445	1	961.7468654	1015.9814	815.801147	835.801147	180.180252	20900.91016
448	1.461050405	1	963.2079158	1017.9814	936.495239	956.495239	61.486134	5103.349121
449	0.262222512	1	963.4701383	1019.9814	907.927307	927.927307	92.054062	7640.487305
450	0.652127325	0	964.1222656	1022.5514	885.690369	905.690369	116.861046	9699.466797
451	6.982967815	0	971.1052335	1025.1213	1012.12134	1042.121338	0	0
452	5.461174636	1	976.5664081	1027.1213	892.74939	912.74939	114.371971	9492.874023
453	2.575269381	0	979.1416775	1029.6913	867.327332	887.327332	142.363953	14236.39551
454	1.491181429	0	980.6328589	1032.2612	862.101074	882.101074	150.160141	15016.01367
455	0.727855499	1	981.3607144	1034.2612	1021.26123	1051.26123	0	0
456	0.667133378	0	982.0278478	1036.8312	959.452087	979.452087	57.379086	3787.019531
457	0.803258221	1	982.831106	1038.8312	982.389526	1002.389526	36.441654	2405.14917
458	0.470328594	0	983.3014346	1041.4011	849.230347	869.230347	172.170807	17217.08008
459	1.937859367	1	985.239294	1043.4011	981.528381	1001.528381	41.872768	2763.602783
460	0.195303416	0	985.4345974	1045.9711	1032.97107	1062.971069	0	0
461	3.619296676	0	989.0538941	1048.541	1035.54102	1065.541016	0	0

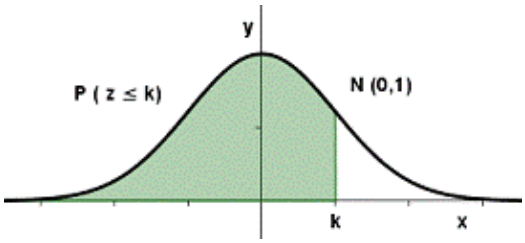
462	8.317913228	0	997.3718073	1051.111	985.866211	1005.866211	45.244766	2986.154541
463	4.201601032	0	1001.573408	1053.6809	1040.68091	1070.680908	0	0
464	9.06721497	1	1010.640623	1055.6809	991.615601	1011.615601	44.065319	2908.311035
465	5.430707657	1	1016.071331	1057.6809	874.985474	894.985474	162.695404	16269.54004
466	0.302010894	0	1016.373342	1060.2509	985.397278	1005.397278	54.853565	3620.335205
467	1.133444399	1	1017.506786	1062.2509	889.529724	909.529724	152.721146	15272.11426
468	0.168426079	1	1017.675212	1064.2509	1012.75415	1032.75415	31.496763	2078.786377
469	1.525328918	0	1019.200541	1066.8208	926.219299	946.219299	120.601501	12060.15039
470	3.372695983	0	1022.573237	1069.3907	975.320862	995.320862	74.069901	6147.801758
471	2.254377166	0	1024.827614	1071.9607	916.67749	936.67749	135.283234	13528.32324
472	2.506093408	1	1027.333708	1073.9607	927.390869	947.390869	126.569855	12656.98535
473	1.424234222	1	1028.757942	1075.9607	934.875793	954.875793	121.084908	12108.49121
474	6.547610022	1	1035.305552	1077.9607	1064.96069	1094.960693	0	0
475	1.56743305	1	1036.872985	1079.9607	930.562683	950.562683	129.39801	12939.80078
476	1.736516183	1	1038.609501	1081.9607	1027.01672	1047.016724	34.943913	2306.29834
477	0.253498005	0	1038.862999	1084.5306	925.466248	945.466248	139.064377	13906.4375
478	0.11691548	0	1038.979915	1087.1006	1011.72131	1031.721313	55.379307	3655.03418
479	8.794804505	0	1047.774719	1089.6705	1076.67053	1106.670532	0	0
480	8.513175736	0	1056.287895	1092.2405	961.707092	981.707092	110.533401	9174.272461
481	0.274153078	1	1056.562048	1094.2405	985.554749	1005.554749	88.685753	7360.91748
482	2.878644022	1	1059.440692	1096.2405	984.015991	1004.015991	92.224487	7654.632324
483	0.700847338	1	1060.141539	1098.2405	1043.75256	1063.752563	34.487957	2276.205078
484	0.934135487	1	1061.075675	1100.2405	992.645203	1012.645203	87.595276	7270.407715
485	1.592357183	0	1062.668032	1102.8104	1039.38709	1059.387085	43.423294	2865.9375
486	0.696454908	0	1063.364487	1105.3804	1021.37585	1041.375854	64.004547	5312.377441
487	0.428003257	1	1063.79249	1107.3804	1094.38037	1124.380371	0	0
488	0.166290881	1	1063.958781	1109.3804	997.690857	1017.690857	91.689522	7610.230469
489	7.637827537	0	1071.596609	1111.9503	927.369141	947.369141	164.581177	16458.11719
490	4.153697707	0	1075.750306	1114.5203	1101.52026	1131.520264	0	0
491	3.928360195	1	1079.678667	1116.5203	886.279297	906.279297	210.240952	24387.95117
492	4.31250383	0	1083.99117	1119.0902	1008.87402	1028.874023	90.216148	7487.94043
493	3.262245974	0	1087.253416	1121.6602	1108.66016	1138.660156	0	0
494	1.505090296	1	1088.758507	1123.6602	988.367798	1008.367798	115.292336	9569.263672
495	1.859302108	1	1090.617809	1125.6602	1009.95984	1029.959839	95.70031	7943.125488
496	0.30664914	1	1090.924458	1127.6602	1062.95068	1082.950684	44.709446	2950.823486
497	0.692040526	0	1091.616498	1130.2301	1045.46704	1065.467041	64.763016	5375.330078
498	6.264447058	0	1097.880946	1132.8	1046.07434	1066.074341	66.725655	5538.229492
499	1.959695223	1	1099.840641	1134.8	957.067871	977.067871	157.732147	15773.21484
500	9.947741461	1	1109.788382	1136.8	973.136963	993.136963	143.663101	14366.31055
501	3.061892766	1	1112.850275	1138.8	1125.80005	1155.800049	0	0
502	0.591847394	1	1113.442122	1140.8	915.785767	935.785767	205.014267	23781.6543
503	0.333228035	0	1113.77535	1143.37	1130.37	1160.369995	0	0
504	4.534784027	0	1118.310134	1145.9399	1132.93994	1162.939941	0	0
505	4.181741497	0	1122.491876	1148.5099	1071.18994	1091.189941	57.319946	3783.116455
506	1.69129767	0	1124.183174	1151.0798	973.914612	993.914612	157.165222	15716.52246
507	3.906175066	0	1128.089349	1153.6498	1095.521	1115.520996	38.12875	2516.497559
508	2.168767893	1	1130.258117	1155.6498	1098.81873	1118.818726	36.831104	2430.852783
509	1.361702983	0	1131.61982	1158.2197	1077.15662	1097.156616	61.063141	5068.240723
510	3.055561544	0	1134.675381	1160.7897	992.178955	1012.178955	148.610718	14861.07227
511	0.737646155	0	1135.413027	1163.3596	1070.78003	1090.780029	72.579575	6024.104492
512	0.020483566	0	1135.433511	1165.9296	1152.92957	1182.929565	0	0
513	5.729041272	1	1141.162552	1167.9296	1110.76148	1130.761475	37.168053	2453.091553
514	3.367476283	1	1144.530028	1169.9296	1118.26013	1138.260132	31.669491	2090.186279
515	0.113735958	0	1144.643764	1172.4995	1031.02039	1051.020386	121.479111	12147.91113
516	1.32042816	1	1145.964192	1174.4995	1037.43359	1057.433594	117.065865	9716.466797
517	0.068890097	1	1146.033083	1176.4995	1040.91821	1060.918213	115.58136	9593.25293
518	1.656111749	1	1147.689194	1178.4995	1019.07788	1039.077881	139.421677	13942.16797
519	5.354147941	0	1153.043342	1181.0695	1168.06946	1198.069458	0	0
520	3.416624694	1	1156.459967	1183.0695	1119.3346	1139.334595	43.734875	2886.501709
521	0.403088996	1	1156.863056	1185.0695	1134.22778	1154.227783	30.841686	2035.55127
522	6.968465369	0	1163.831521	1187.6394	1115.31567	1135.315674	52.323685	3453.363281
523	1.063265253	0	1164.894787	1190.2094	1109.64844	1129.648438	60.560875	5026.552734
524	1.860172776	1	1166.754959	1192.2094	1179.20935	1209.209351	0	0
525	0.367183394	1	1167.122143	1194.2094	1073.06751	1093.067505	101.141808	8394.770508
526	5.861555035	0	1172.983698	1196.7793	1104.75781	1124.757813	72.021431	5977.778809
527	2.235791134	1	1175.219489	1198.7793	1129.5	1149.5	49.279263	3252.431396
528	0.475099706	0	1175.694589	1201.3492	1067.14905	1087.149048	114.200157	9478.613281

529	0.135231619	0	1175.82982	1203.9192	1059.64294	1079.642944	124.276192	12427.61914
530	0.005554421	1	1175.835375	1205.9192	969.001587	989.001587	216.917572	25162.4375
531	3.043764039	1	1178.879139	1207.9192	1155.08862	1175.088623	32.830505	2166.813477
532	7.038140691	1	1185.917279	1209.9192	1013.01904	1033.019043	176.900085	17690.00781
533	0.805907696	1	1186.723187	1211.9192	1158.88001	1178.880005	33.039131	2180.582764
534	1.045334828	0	1187.768522	1214.4891	1126.51245	1146.512451	67.976746	5642.069824
535	0.300609694	0	1188.069132	1217.0591	1098.63879	1118.638794	98.420341	8168.888184
536	1.516530146	1	1189.585662	1219.0591	1051.73914	1071.739136	147.319901	14731.99023
537	0.049305692	1	1189.634967	1221.0591	1084.8396	1104.8396	116.219543	9646.22168
538	5.903496876	1	1195.538464	1223.0591	1210.05908	1240.059082	0	0
539	0.570890091	1	1196.109354	1225.0591	1050.58801	1070.588013	154.471054	15447.10547
540	3.425787473	0	1199.535142	1227.629	1074.99377	1094.993774	132.635208	13263.52051
541	0.184268949	1	1199.719411	1229.629	1092.76794	1112.767944	116.861046	9699.466797
542	0.787104529	0	1200.506515	1232.199	1051.48071	1071.480713	160.718216	16071.82129
543	3.536171253	1	1204.042687	1234.199	1059.26367	1079.263672	154.935318	15493.53223
544	3.438584647	0	1207.481271	1236.7689	1223.76892	1253.768921	0	0
545	6.628068897	1	1214.10934	1238.7689	1170.8064	1190.806396	47.962505	3165.525391
546	2.687238565	1	1216.796579	1240.7689	1073.09876	1093.098755	147.670166	14767.0166
547	1.831841699	1	1218.62842	1242.7689	1179.96729	1199.967285	42.80162	2824.906982
548	0.79911458	0	1219.427535	1245.3389	1232.33887	1262.338867	0	0
549	10.35140429	1	1229.778939	1247.3389	1234.33887	1264.338867	0	0
550	0.638039019	0	1230.416978	1249.9088	1236.90881	1266.908813	0	0
551	1.571619655	0	1231.988598	1252.4788	1154.16736	1174.167358	78.311409	6499.847168
552	4.567536517	0	1236.556134	1255.0487	1178.59375	1198.59375	56.454914	3726.024414
553	2.744019026	1	1239.300154	1257.0487	1128.80298	1148.802979	108.245682	8984.391602
554	1.615612902	1	1240.915766	1259.0487	1145.99365	1165.993652	93.055115	7723.574707
555	2.152637067	1	1243.068403	1261.0487	1107.13171	1127.131714	133.917007	13391.70117
556	0.689668185	1	1243.758072	1263.0487	1202.76856	1222.768555	40.280197	2658.49292
557	1.232422901	1	1244.990495	1265.0487	1201.38721	1221.387207	43.661514	2881.659912
558	2.406282632	0	1247.396777	1267.6187	1131.37	1151.369995	116.248634	9648.636719
559	1.644477127	1	1249.041254	1269.6187	1211.00781	1231.007813	38.610821	2548.314209
560	1.226812291	0	1250.268067	1272.1886	1119.7544	1139.754395	132.434204	13243.41992
561	2.144271977	0	1252.412339	1274.7585	1212.10291	1232.102905	42.655632	2815.271729
562	1.314931943	0	1253.727271	1277.3285	1125.29517	1145.295166	132.03334	13203.33398
563	2.319115065	0	1256.046386	1279.8984	1266.89844	1296.898438	0	0
564	2.043990475	0	1258.090376	1282.4684	1187.09155	1207.091553	75.376793	6256.273926
565	5.009923593	0	1263.1003	1285.0383	1272.03833	1302.03833	0	0
566	1.386098019	1	1264.486398	1287.0383	1274.03833	1304.03833	0	0
567	1.62917145	0	1266.115569	1289.6083	1181.8949	1201.894897	87.713432	7280.214844
568	0.569385739	0	1266.684955	1292.1782	1181.45654	1201.456543	90.721741	7529.904297
569	0.193287253	0	1266.878242	1294.7482	1211.74304	1231.743042	63.005081	5229.421875
570	0.48484666	0	1267.363089	1297.3181	1227.45276	1247.452759	49.865314	3291.11084
571	1.839503968	1	1269.202593	1299.3181	1286.31812	1316.318115	0	0
572	0.406846597	1	1269.609439	1301.3181	1288.31812	1318.318115	0	0
573	13.17189253	0	1282.781332	1303.8881	1231.9668	1251.966797	51.921295	3426.80542
574	0.576009719	0	1283.357342	1306.458	1178.83923	1198.839233	107.618736	8932.355469
575	5.633480509	0	1288.990822	1309.028	1242.39185	1262.391846	46.636093	3077.982178
576	1.039861448	0	1290.030684	1311.5979	1122.99536	1142.995361	168.602524	16860.25195
577	1.383345776	0	1291.414029	1314.1678	1301.16785	1331.167847	0	0
578	1.918152632	0	1293.332182	1316.7378	1078.37109	1098.371094	218.366684	25330.53516
579	0.176130466	1	1293.508312	1318.7378	1216.04773	1236.047729	82.690117	6863.279785
580	1.44048726	0	1294.9488	1321.3077	1244.97046	1264.970459	56.337261	3718.259277
581	2.692982613	0	1297.641782	1323.8777	1190.44788	1210.447876	113.429855	9414.677734
582	2.76138991	0	1300.403172	1326.4476	1313.44763	1343.447632	0	0
583	1.908423483	0	1302.311596	1329.0176	1262.64197	1282.641968	46.375622	3060.791016
584	0.4653304	0	1302.776926	1331.5875	1154.42236	1174.422363	157.165222	15716.52246
585	0.802325443	1	1303.579252	1333.5875	1225.87415	1245.874146	87.713432	7280.214844
586	6.002833809	0	1309.582085	1336.1575	1202.84131	1222.841309	113.316147	9405.240234
587	2.353714175	1	1311.9358	1338.1575	1325.15747	1355.157471	0	0
588	1.266527175	0	1313.202327	1340.7274	1327.72742	1357.727417	0	0
589	1.482750933	1	1314.685078	1342.7274	1262.96729	1282.967285	59.760159	3944.17041
590	1.622835836	0	1316.307913	1345.2974	1289.70288	1309.702881	35.594521	2349.238281
591	0.385898621	0	1316.693812	1347.8673	1274.31189	1294.31189	53.55537	3534.654541
592	2.813979199	1	1319.507791	1349.8673	1111.16516	1131.165161	218.702164	25369.45117
593	0.074797369	0	1319.582589	1352.4373	1339.43726	1369.437256	0	0
594	5.247015019	0	1324.829604	1355.0072	1301.84631	1321.846313	33.160934	2188.621582
595	2.061554116	1	1326.891158	1357.0072	1344.0072	1374.007202	0	0

596	1.918222935	1	1328.809381	1359.0072	1140.74927	1160.749268	198.257919	22997.91797
597	0.533488881	0	1329.34287	1361.5771	1197.27673	1217.276733	144.300461	14430.0459
598	0.219568241	1	1329.562438	1363.5771	1188.89539	1208.895386	154.681763	15468.17578
599	3.082120557	0	1332.644558	1366.1471	1219.54553	1239.545532	126.601585	12660.1582
600	0.176163511	0	1332.820722	1368.717	1201.31946	1221.319458	147.397629	14739.7627
601	0.4818028	0	1333.302525	1371.287	1358.28699	1388.286987	0	0
602	2.466577658	0	1335.769102	1373.8569	1312.05689	1332.056885	41.800102	2758.806641
603	0.339331637	0	1336.108434	1376.4269	1325.17175	1345.171753	31.25518	2062.841797
604	0.558175612	1	1336.66661	1378.4269	1168.41089	1188.410889	190.016022	22041.85938
605	3.148583405	1	1339.815193	1380.4269	1367.42688	1397.42688	0	0
606	0.681850362	1	1340.497043	1382.4269	1369.42688	1399.42688	0	0
607	2.533092705	0	1343.030136	1384.9968	1371.99683	1401.996826	0	0
608	0.245663669	0	1343.2758	1387.5668	1223.94104	1243.94104	143.625732	14362.57324
609	2.569796338	1	1345.845596	1389.5668	1327.09351	1347.093506	42.473289	2803.237061
610	0.562444438	0	1346.408041	1392.1367	1293.33472	1313.334717	78.801987	6540.564941
611	3.000066336	0	1349.408107	1394.7067	1381.70667	1411.706665	0	0
612	3.013944256	0	1352.422051	1397.2766	1257.93481	1277.934814	119.341805	9905.370117
613	1.59702837	0	1354.01908	1399.8466	1348.78101	1368.781006	31.065561	2050.327148
614	0.285436341	1	1354.304516	1401.8466	1271.22974	1291.229736	110.616844	9181.198242
615	3.129815389	1	1357.434331	1403.8466	1335.3396	1355.3396	48.50692	3201.456787
616	0.153108853	1	1357.58744	1405.8466	1327.0625	1347.0625	58.784111	3879.751221
617	0.712361699	0	1358.299802	1408.4165	1305.95508	1325.955078	82.461441	6844.299805
618	2.786228451	1	1361.08603	1410.4165	1328.22315	1348.223145	62.193359	5162.048828
619	0.746916695	0	1361.832947	1412.9865	1354.62573	1374.625732	38.360718	2531.807373
620	3.676220519	1	1365.509167	1414.9865	1352.25781	1372.257813	42.728615	2820.088623
621	0.112466449	0	1365.621634	1417.5564	1347.50171	1367.501709	50.054745	3303.613037
622	0.053771871	0	1365.675406	1420.1263	1317.87048	1337.870483	82.255905	6827.240234
623	1.184753194	1	1366.860159	1422.1263	1409.12634	1439.126343	0	0
624	0.781599486	1	1367.641758	1424.1263	1253.76563	1273.765625	150.360733	15036.07324
625	7.85805443	0	1375.499813	1426.6963	1413.69629	1443.696289	0	0
626	0.471277433	0	1375.97109	1429.2662	1416.26624	1446.266235	0	0
627	0.632304263	0	1376.603395	1431.8362	1354.81152	1374.811523	57.024597	3763.623535
628	2.135435591	0	1378.73883	1434.4061	1421.40613	1451.406128	0	0
629	3.535601011	1	1382.274431	1436.4061	1423.40613	1453.406128	0	0
630	4.027928798	1	1386.30236	1438.4061	1313.39941	1333.399414	105.006676	8715.553711
	aterrizajes	**317**					**43549.146**	**3965367.42**
	despegues	**313**			**valor esperado**		**69.125628**	**6294.234**
					varianza muestral		**3586.1214**	**40529415.1**
					desviación estándar		**59.884233**	**6366.27168**

ANEXO C

Tabla de la distribución Normal Estándar N(0,1)



z	0,00	0,01	0,02	0,03	0,04	0,05	0,06	0,07	0,08	0,09
0,0	0,5000	0,5040	0,5080	0,5120	0,5160	0,5199	0,5239	0,5279	0,5319	0,5359
0,1	0,5398	0,5438	0,5478	0,5517	0,5557	0,5596	0,5636	0,5675	0,5714	0,5753
0,2	0,5793	0,5832	0,5871	0,5910	0,5948	0,5987	0,6026	0,6064	0,6103	0,6141
0,3	0,6179	0,6217	0,6255	0,6293	0,6331	0,6368	0,6406	0,6443	0,6480	0,6517
0,4	0,6554	0,6591	0,6628	0,6664	0,6700	0,6736	0,6772	0,6808	0,6844	0,6879
0,5	0,6915	0,6950	0,6985	0,7019	0,7054	0,7088	0,7123	0,7157	0,7190	0,7224
0,6	0,7257	0,7291	0,7324	0,7357	0,7389	0,7422	0,7454	0,7486	0,7517	0,7549
0,7	0,7580	0,7611	0,7642	0,7673	0,7703	0,7734	0,7764	0,7794	0,7823	0,7652
0,8	0,7881	0,7910	0,7939	0,7967	0,7995	0,8023	0,8051	0,8078	0,8106	0,8133
0,9	0,8159	0,8186	0,8212	0,8238	0,8264	0,8289	0,8315	0,8340	0,8365	0,8389
1,0	0,8413	0,8438	0,8461	0,8485	0,8508	0,8531	0,8554	0,8577	0,8599	0,8621
1,1	0,8643	0,8665	0,8686	0,8708	0,8729	0,8749	0,8770	0,8790	0,8810	0,8930
1,2	0,8849	0,8869	0,8888	0,8907	0,8925	0,8944	0,8962	0,8980	0,8997	0,9015
1,3	0,9032	0,9049	0,9066	0,9082	0,9099	0,9115	0,9131	0,9147	0,9162	0,9177
1,4	0,9192	0,9207	0,9222	0,9236	0,9251	0,9265	0,9279	0,9292	0,9306	0,9319
1,5	0,9332	0,9345	0,9357	0,9370	0,9382	0,9394	0,9406	0,9418	0,9429	0,9441
1,6	0,9452	0,9463	0,9474	0,9484	0,9495	0,9505	0,9515	0,9525	0,9535	0,9545
1,7	0,9554	0,9561	0,9573	0,9582	0,9591	0,9599	0,9608	0,9616	0,9625	0,9633
1,8	0,9641	0,9649	0,9656	0,9664	0,9671	0,9678	0,9686	0,9693	0,9699	0,9706
1,9	0,9713	0,9719	0,9726	0,9732	0,9738	0,9744	0,9750	0,9756	0,9761	0,9767

ANEXO D

Programación de Búsqueda Tabú en C++

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>

#define MAX 630

int tamano;

struct avion{
	int tipo;
	int num_avion;
	float rj;
	float Ej;
	float Lj;
	float Cj;
	float retraso;
	float costo;
	float pj;
};

avion a[MAX];
avion temporal[MAX];

struct mov_tabu{
	int n_avion;
	int ant;
	int post;
};

mov_tabu tabu[5];

int leer_archivo(){
   FILE *archivo;
		FILE *temp2;
		int num;
		int iF,iC;
		char c=0;
		int comas=0, renglones=0;

		archivo = fopen("leer2.csv","r+");
		temp2 = fopen("temp2.txt","w");
		if(archivo == NULL){
			printf("Error, no se puede abrir el archivo\n");
		}
		else{

			while(c!='\n'){
	c = fgetc(archivo);
				if(c==',' || c=='\n'){
					comas++;
					fprintf(temp2," ");
```

```
                        }else
                                fprintf(temp2,"%c",c);
        }

                while(c!=EOF){
                        c = fgetc(archivo);

                        if(c==',' || c=='\n'){
                if(c=='\n')
                    renglones++;
                                fprintf(temp2," ");
           }
                        else
                                fprintf(temp2,"%c",c);
                }
                rewind(archivo);
                fclose(archivo);
                fclose(temp2);

                printf("\ncolumnas: %d, renglones: %d\n",comas,renglones+1);
        temp2 = fopen("temp2.txt","r");
                for(iF=0;iF<9;iF++){
                        for(iC=0;iC<MAX;iC++){
            if(iF==0)
                fscanf(temp2,"%d",&a[iC].num_avion);
            if(iF==1)
                fscanf(temp2,"%d",&a[iC].tipo);
            if(iF==2)
                fscanf(temp2,"%f",&a[iC].pj);
            if(iF==3)
                fscanf(temp2,"%f",&a[iC].rj);
            if(iF==4)
                fscanf(temp2,"%f",&a[iC].Cj);
            if(iF==5)
                fscanf(temp2,"%f",&a[iC].Ej);
            if(iF==6)
                fscanf(temp2,"%f",&a[iC].Lj);
            if(iF==7)
                fscanf(temp2,"%f",&a[iC].retraso);
            if(iF==8)
                fscanf(temp2,"%f",&a[iC].costo);
            }
                        }
                fclose(temp2);
          }

          return 0;
}
int lista_tabu(int i){
   int flag=1, j=0, ant=0, post=0;
   mov_tabu t;
   t.n_avion = a[i].num_avion;
   for(j=0; j<MAX; j++){
          if(a[j].Cj < a[i].rj){
                t.ant = a[j].num_avion;
                t.post = a[j+1].num_avion;
```

```
                    }
                }
    for(j=0; j<5; j++){
            if(tabu[j].n_avion==t.n_avion && tabu[j].ant==t.ant && tabu[j].post==t.post)
                                flag = 0;
            }
    if(flag == 1){
            tabu[4] = tabu[3];
            tabu[3] = tabu[2];
            tabu[2] = tabu[1];
            tabu[1] = tabu[0];
            tabu[0].n_avion = t.n_avion;
            tabu[0].ant = t.ant;
            tabu[0].post = t.post;
            }

    return (flag);
}

void reasignar_vuelo(int i){
     int j=0, pos=0;
     avion temp, temp2, temp3;
     for(j=0; j<MAX; j++){
             if(temporal[j].Cj < temporal[i].rj){
                     pos = j+2;
                     }
             }
     for(j=i+1; j<MAX; j++){
               temp3=temporal[i];
               temporal[j-1]=temporal[j];
             }
     for(j=pos; j<MAX; j++){
               if(j == pos){
                   temp = temporal[j];
                   temporal[j] = temp3;
                   temporal[j].Cj = temporal[j-1].Cj + temporal[j].pj;
                   }
               else{

                       temp2 = temporal[j];
                       temporal[j] = temp;
                       temp = temp2;
                       if(temporal[j-1].Cj >= temporal[j].rj){
                           temporal[j].Cj = temporal[j-1].Cj + temporal[j].pj;
                           }
                       else{
                           temporal[j].Cj = temporal[j].rj + temporal[j].pj;
                           }

                   }
               }
}

void calcular_retraso(){
      int i=0;
      for(i=0; i<MAX; i++){
```

```
            if(temporal[i].Cj > temporal[i].Lj)
                temporal[i].retraso = temporal[i].Cj - temporal[i].Lj;
            if(temporal[i].Cj < temporal[i].Ej)
                temporal[i].retraso = temporal[i].Cj - temporal[i].Ej;
            if(temporal[i].Cj >= temporal[i].Ej && temporal[i].Cj <= temporal[i].Lj)
                temporal[i].retraso = 0;
        }
}

float calcular_retraso_tot(){
    int i=0;
    float R_tot=0;
    for(i=0; i<MAX; i++){
            R_tot = R_tot + temporal[i].retraso;
            }
    return(R_tot);

}

float calcular_costo(){
    int i;
    i=0;
    float costo_tot=0;

    for(i=0; i<MAX; i++){
        if(temporal[i].Cj <= temporal[i].Lj && temporal[i].Cj >= temporal[i].Ej)
                temporal[i].costo=0;
        if(fabs(temporal[i].retraso) <= 60 && fabs(temporal[i].retraso)> 0)
                temporal[i].costo=(200/3)*fabs(temporal[i].retraso);
        if(fabs(temporal[i].retraso) <= 120 && fabs(temporal[i].retraso)> 60)
                temporal[i].costo=(250/3)*fabs(temporal[i].retraso);
        if(fabs(temporal[i].retraso) <= 180 && fabs(temporal[i].retraso)> 120)
                temporal[i].costo=(300/3)*fabs(temporal[i].retraso);
        if(fabs(temporal[i].retraso) <= 240 && fabs(temporal[i].retraso)> 180)
                temporal[i].costo=(350/3)*fabs(temporal[i].retraso);
        costo_tot = costo_tot + temporal[i].costo;
           }
     return(costo_tot);
}

void imprimir(){
   FILE *archivo2;
   int j=0, i=0;

            archivo2 = fopen("resultados.csv","r+");
            if(archivo2 == NULL){
            printf("Error, no se puede abrir el archivo\n");
            }
            else{

            for(j=0; j<9; j++){
              i=0;
              for(i=0; i<MAX; i++){
                  if(j==0){
                     fprintf(archivo2, "%d", a[i].num_avion);
                     fprintf(archivo2, ",");
```

```
                }
            if(j==1){
                fprintf(archivo2, "%d", a[i].tipo);
                fprintf(archivo2, ",");
                 }
            if(j==2){
                fprintf(archivo2, "%f", a[i].pj);
                fprintf(archivo2, ",");
                 }
            if(j==3){
                fprintf(archivo2, "%f", a[i].rj);
                fprintf(archivo2, ",");
                 }
            if(j==4){
                fprintf(archivo2, "%f", a[i].Cj);
                fprintf(archivo2, ",");
                 }
            if(j==5){
                fprintf(archivo2, "%f", a[i].Ej);
                fprintf(archivo2, ",");
                 }
            if(j==6){
                fprintf(archivo2, "%f", a[i].Lj);
                fprintf(archivo2, ",");
                 }
            if(j==7){
                fprintf(archivo2, "%f", a[i].retraso);
                fprintf(archivo2, ",");
                 }
            if(j==8){
                fprintf(archivo2, "%f", a[i].costo);
                fprintf(archivo2, ",");
                 }
                      }
            fprintf(archivo2, "\n");
          }
        }

   fclose(archivo2);

}

main(){
    int i=0, j=0, pos1, pos2, band =1;
    float retraso_tot=43549.14576, costo_tot=3965367.418, c_tot, R_tot, max1, max2, f;
    for(i=0; i<5; i++){
            tabu[i].n_avion = 0;
            tabu[i].ant = 0;
            tabu[i].post = 0;
            }

    leer_archivo();
    for(i=0; i<MAX; i++){
            temporal[i] = a[i];
            }
```

```
while(j<5){
i=0;
max1=0;
max2=0;
pos1=0;
pos2=0;
R_tot=0;
c_tot=0;
f=0;

for(i=0; i<MAX; i++){
    if(temporal[i].Cj - temporal[i].rj > 5){
       if (temporal[i].retraso > max){
          max1 = temporal[i].retraso;
          pos1 = i;
          }
       else {
          if(temporal[i].retraso > max2){
                max2 = temporal[i].retraso;
                pos2 = i;
                }
           }
          }
       }

if(lista_tabu(pos1)==1){
      reasignar_vuelo(pos1);
      calcular_retraso();
      R_tot=calcular_retraso_tot();
      c_tot = calcular_costo();

      if(c_tot < costo_tot){
            retraso_tot = R_tot;
            costo_tot = c_tot;
            for(i=0; i<MAX; i++){
                  a[i] = temporal[i];
                }
           }
      else
        j++;
        }

else{
      if(lista_tabu(pos2)==1){
          reasignar_vuelo(pos2);
          calcular_retraso();
          R_tot=calcular_retraso_tot();
          c_tot = calcular_costo();
          if(c_tot < costo_tot){
              retraso_tot = R_tot;
              costo_tot = c_tot;
              for(i=0; i<MAX; i++){
                    a[i] = temporal[i];
                    }
              }
          else
```

```
                j++;
                    }
                }
            }
        printf("\nretraso total= %f", retraso_tot);
        printf("\ncosto toal= %f", costo_tot);
        imprimir();


        fflush(stdin);
        getchar();
        return 0;

}
```

ANEXO E

Resultados de Búsqueda Tabú

Vuelo (j)	aterrizajes/ despegues	Tiempo procesamiento	llegadas a la cola (rj)		dj			
				Cj	Ej	Lj	Rj	costo retraso
1	1	2	0	2	-11	19	0	0
2	1	2	1.728097	4	-9	21	0	0
3	0	2.57	2.351835	6.57	-6.43	23.57	0	0
4	0	2.57	3.62656	9.14	-3.860001	26.139999	0	0
5	0	2.57	4.887257	11.71	-1.290001	28.709999	0	0
6	0	2.57	7.42145	14.28	1.279999	31.279999	0	0
7	1	2	8.88163	16.279999	3.279999	33.279999	0	0
8	1	2	10.407897	18.279999	5.279999	35.279999	0	0
9	1	2	10.452294	20.279999	7.279999	37.279999	0	0
10	1	2	13.409892	22.279999	9.279999	39.279999	0	0
11	0	2.57	14.721812	24.849998	11.849998	41.849998	0	0
12	1	2	17.440933	26.849998	13.849998	43.849998	0	0
13	1	2	19.72508	28.849998	15.849998	45.849998	0	0
14	0	2.57	21.61478	31.419998	18.419998	48.419998	0	0
15	1	2	22.516722	33.419998	20.419998	50.419998	0	0
16	0	2.57	22.683163	35.989998	22.989998	52.989998	0	0
17	1	2	26.897001	37.989998	24.989998	54.989998	0	0
18	1	2	29.833477	39.989998	26.989998	56.989998	0	0
19	0	2.57	30.370218	42.559998	29.559998	59.559998	0	0
20	1	2	31.289454	44.559998	31.559998	61.559998	0	0
21	0	2.57	38.699604	47.129997	34.129997	64.129997	0	0
22	1	2	41.193562	49.129997	36.129997	66.129997	0	0
23	1	2	43.316254	51.129997	38.129997	68.129997	0	0
24	1	2	43.355675	53.129997	40.129997	70.129997	0	0
25	1	2	44.589787	55.129997	42.129997	72.129997	0	0
26	0	2.57	47.579079	57.699997	0.265541	20.265541	37.434456	2470.67407
27	0	2.57	56.436279	60.269997	47.269997	77.269997	0	0
28	1	2	60.026524	62.269997	-12.059326	7.940674	54.329323	3585.73535
29	0	2.57	60.54657	64.839996	-13.48777	6.51223	58.327766	3849.63257
30	1	2	60.67561	66.839996	53.839996	83.839996	0	0
31	1	2	60.787598	68.839996	-9.765404	10.234596	58.6054	3867.9563
32	0	2.57	61.65662	71.409996	8.517075	28.517075	42.892921	2830.93286
33	1	2	62.610538	73.409996	60.409996	90.409996	0	0
34	0	2.57	65.65934	75.979996	2.17345	22.17345	53.806545	3551.23193
35	1	2	68.133659	77.979996	11.418358	31.418358	46.561638	3073.06812
36	0	2.57	69.833565	80.549995	67.549995	97.549995	0	0
37	0	2.57	71.927284	83.119995	18.539547	38.539547	44.580448	2942.30957
38	1	2	72.913155	85.119995	72.119995	102.119995	0	0
39	1	2	74.138405	87.119995	19.326134	39.326134	47.793861	3154.39478
40	1	2	77.50811	89.119995	76.119995	106.119995	0	0
41	1	2	78.018524	91.119995	78.119995	108.119995	0	0
42	0	2.57	78.303108	93.689995	80.689995	110.689995	0	0
43	1	2	78.858574	95.689995	38.202213	58.202213	37.487782	2474.1936
44	1	2	80.372505	97.689995	32.241787	52.241787	45.448208	2999.58179
45	0	2.57	84.03054	100.259995	30.980732	50.980732	49.279263	3252.4314
46	0	2.57	88.862373	102.829994	41.39283	61.39283	41.437164	2734.85278
47	1	2	91.209084	104.829994	-3.808189	16.191811	88.638184	7356.96924
48	1	2	91.951981	106.829994	36.072693	56.072693	50.757301	3349.98193
49	1	2	92.471153	108.829994	3.9506	23.9506	84.879395	7044.98975
50	1	2	93.556389	110.829994	97.829994	127.829994	0	0
51	1	2	95.831421	112.829994	53.215622	73.215622	39.614372	2614.54858
52	1	2	98.590286	114.829994	101.829987	131.829987	0	0
53	1	2	98.744385	116.829994	57.986649	77.986649	38.843346	2563.66089
54	0	2.57	100.849022	119.399994	47.3255	67.3255	52.074493	3436.9165
55	1	2	100.871834	121.399994	7.015923	27.015923	94.384071	7833.87793
56	0	2.57	101.317497	123.969994	110.970001	140.970001	0	0
57	0	2.57	101.477066	126.539993	22.908791	42.908791	83.631203	6941.38965
58	1	2	104.503983	128.539993	60.108261	80.108261	48.431732	3196.49439
59	0	2.57	107.611702	131.110001	32.12925	52.12925	78.980751	6555.40234

60	1	2	113.149506	133.110001	120.110001	150.110001	0	0
61	0	2.57	116.913124	135.680008	122.680008	152.680008	0	0
62	1	2	121.338066	137.680008	85.924095	105.924095	31.755913	2095.89014
63	1	2	127.987625	139.680008	39.23941	59.23941	80.440598	6676.56982
64	1	2	128.076584	141.680008	74.50325	94.50325	47.176758	3113.66602
65	0	2.57	129.425018	144.250015	63.869606	83.869606	60.380409	5011.57373
66	0	2.57	130.66861	146.820023	133.820023	163.820023	0	0
67	1	2	130.8302	148.820023	94.034058	114.034058	34.785965	2295.87378
68	1	2	132.250809	150.820023	69.957954	89.957954	60.862068	5051.55176
69	1	2	133.747147	152.820023	81.891441	101.891441	50.928581	3361.28638
70	1	2	133.963989	154.820023	103.323257	123.323257	31.496765	2078.78662
71	1	2	134.706604	156.820023	55.065407	75.065407	81.754616	6785.6333
72	1	2	137.292877	158.820023	1.733566	21.733566	137.086456	13708.6455
73	1	2	138.322968	160.820023	147.820023	177.820023	0	0
74	1	2	142.60199	162.820023	75.156906	95.156906	67.663116	5616.03857
75	0	2.57	143.211777	165.39003	51.917847	71.917847	93.472183	7758.19141
76	0	2.57	146.288498	167.960037	94.211487	114.211487	53.74855	3547.4043
77	0	2.57	146.447784	170.530045	112.66861	132.66861	37.861435	2498.85474
78	0	2.57	148.867294	173.100052	57.224998	77.224998	95.875053	7957.6294
79	0	2.57	149.826462	175.670059	162.670059	192.670059	0	0
80	1	2	152.617035	177.670059	96.264572	116.264572	61.405487	5096.65527
81	0	2.57	153.784988	180.240067	167.240067	197.240067	0	0
82	0	2.57	157.462143	182.810074	169.810074	199.810074	0	0
83	0	2.57	162.118057	185.380081	60.613213	80.613213	104.766869	8695.65039
84	0	2.57	165.323288	187.950089	69.14782	89.14782	98.802269	8200.58789
85	0	2.57	173.583267	190.520096	81.28614	101.28614	89.233955	7406.41846
86	1	2	179.599258	192.520096	39.884888	59.884888	132.635208	13263.5205
87	0	2.57	180.460709	195.090103	142.259598	162.259598	32.830505	2166.81348
88	0	2.57	184.098663	197.66011	44.149551	64.149551	133.510559	13351.0557
89	0	2.57	185.343399	200.230118	187.230118	217.230118	0	0
90	1	2	185.461975	202.230118	189.230118	219.230118	0	0
91	0	2.57	185.616196	204.800125	28.499588	48.499588	156.300537	15630.0537
92	0	2.57	188.582733	207.370132	149.04512	169.04512	38.325012	2529.45068
93	0	2.57	190.471573	209.94014	196.94014	226.94014	0	0
94	1	2	191.75238	211.94014	149.794632	169.794632	42.145508	2781.60352
95	0	2.57	191.764404	214.510147	48.194733	68.194733	146.315414	14631.541
96	0	2.57	194.520187	217.080154	87.736961	107.736961	109.343193	9075.48535
97	1	2	195.491516	219.080154	206.080154	236.080154	0	0
98	0	2.57	195.909592	221.650162	208.650162	238.650162	0	0
109	0	2.57	222.402039	224.220169	128.714111	148.714111	75.506058	6267.00293
99	0	2.57	200.814728	226.790176	105.188255	125.188255	101.601921	8432.95898
100	1	2	204.007675	228.790176	132.778259	152.778259	76.011917	6308.98926
101	0	2.57	205.18483	231.360184	215.790176	245.790176	0	0
102	0	2.57	208.468948	233.930191	218.360184	248.360184	0	0
103	1	2	215.734894	235.930191	84.22226	104.22226	131.707932	13170.793
104	1	2	218.193726	237.930191	63.745697	83.745697	154.184494	15418.4492
105	0	2.57	218.794769	240.500198	177.415527	197.415527	43.084671	2843.58838
106	1	2	220.260773	242.500198	150.397781	170.397781	72.102417	5984.50049
107	1	2	221.163116	244.500198	228.930176	258.930176	0	0
109	0	2.57	222.402039	247.070206	128.714111	148.714111	98.356094	8163.55566
110	1	2	223.523712	249.070206	235.500183	265.500183	0	0
111	0	2.57	225.523956	251.640213	238.07019	268.07019	0	0
112	0	2.57	226.399384	254.21022	150.698151	170.698151	83.51207	6931.50195
122	1	2	255.959137	257.959137	203.508591	223.508591	34.450546	2273.73608
113	1	2	227.667648	259.959137	133.012695	153.012695	106.946442	8876.55469
114	1	2	228.708221	261.959137	244.640198	274.640198	0	0
115	1	2	230.394028	263.959137	246.640198	276.640198	0	0
116	0	2.57	236.142242	266.529144	174.191589	194.191589	72.337555	6004.01709
117	1	2	239.061966	268.529144	151.985718	171.985718	96.543427	8013.10449
118	1	2	241.643188	270.529144	209.308212	229.308212	41.220932	2720.58154
119	1	2	246.037857	272.529144	255.210205	285.210205	0	0
120	1	2	250.623566	274.529144	120.61673	140.61673	133.912415	13391.2412
122	1	2	255.959137	276.529144	203.508591	223.508591	53.020554	3499.35645
123	1	2	257.504364	278.529144	224.348999	244.348999	34.180145	2255.88965
124	1	2	259.381134	280.529144	192.945648	212.945648	67.583496	5609.43018
125	0	2.57	260.954834	283.099152	268.35022	298.35022	0	0
126	0	2.57	261.36496	285.669159	132.153305	152.153305	133.515854	13351.585

127	1	2	262.331482	287.669159	193.233124	213.233124	74.436035	6178.19092
128	1	2	262.904327	289.669159	210.836594	230.836594	58.832565	3882.94922
129	0	2.57	266.068817	292.239166	218.204926	238.204926	54.034241	3566.25977
130	0	2.57	268.352539	294.809174	226.275146	246.275146	48.534027	3203.24585
138	0	2.57	293.025269	297.379181	297.770264	327.770264	-0.391083	25.811462
131	0	2.57	273.63385	299.949188	282.630249	312.630249	0	0
132	0	2.57	276.906158	302.519196	157.899872	177.899872	124.619324	12461.9326
133	1	2	279.981689	304.519196	202.400146	222.400146	82.119049	6815.88086
134	1	2	280.205872	306.519196	121.799255	141.799255	164.71994	16471.9941
135	1	2	284.728455	308.519196	230.548309	250.548309	57.970886	3826.07861
136	1	2	286.589935	310.519196	177.516663	197.516663	113.002533	9379.20996
138	0	2.57	293.025269	313.089203	297.770264	327.770264	0	0
139	0	2.57	301.471924	315.65921	188.865921	208.865921	106.793289	8863.84277
140	1	2	302.885345	317.65921	302.340271	332.340271	0	0
141	0	2.57	304.795197	320.229218	175.332596	195.332596	124.896622	12489.6621
142	0	2.57	305.825897	322.799225	140.214844	160.214844	162.584381	16258.4385
143	0	2.57	306.801239	325.369232	249.050293	269.050293	56.318939	3717.05005
144	0	2.57	308.613983	327.93924	249.420197	269.420197	58.519043	3862.25684
145	0	2.57	311.013153	330.509247	146.558685	166.558685	163.950562	16395.0566
146	0	2.57	311.618774	333.079254	250.139252	270.139252	62.940002	5224.02002
147	0	2.57	317.380493	335.649261	320.330322	350.330322	0	0
148	0	2.57	325.360565	338.219269	322.90033	352.90033	0	0
155	1	2	336.913574	340.219269	184.173843	204.173843	136.045425	13604.543
155	1	2	336.913574	342.219269	184.173843	204.173843	138.045425	13804.543
149	0	2.57	326.118408	344.789276	269.7565	289.7565	55.032776	3632.16309
150	1	2	328.068542	346.789276	165.323166	185.323166	161.46611	16146.6113
151	0	2.57	331.56781	349.359283	330.040344	360.040344	0	0
152	1	2	334.524414	351.359283	332.040344	362.040344	0	0
155	1	2	336.913574	353.359283	184.173843	204.173843	149.18544	14918.5439
156	0	2.57	338.228638	355.929291	263.461182	283.461182	72.468109	6014.85303
157	1	2	340.298798	357.929291	343.180359	373.180359	0	0
158	1	2	345.425476	359.929291	298.332306	318.332306	41.596985	2745.40088
159	0	2.57	345.785767	362.499298	307.989349	327.989349	34.509949	2277.65674
160	0	2.57	345.954803	365.069305	205.108383	225.108383	139.960922	13996.0918
161	1	2	350.220215	367.069305	352.320374	382.320374	0	0
162	0	2.57	353.564148	369.639313	354.890381	384.890381	0	0
163	1	2	354.322723	371.639313	242.271637	262.271637	109.367676	9077.51758
164	0	2.57	357.046661	374.20932	359.460388	389.460388	0	0
165	0	2.57	359.520569	376.779327	235.509445	255.509445	121.269882	12126.9883
166	0	2.57	360.736938	379.349335	364.600403	394.600403	0	0
167	1	2	365.648529	381.349335	329.188568	349.188568	32.160767	2122.6106
168	1	2	365.684692	383.349335	368.600403	398.600403	0	0
169	1	2	367.533325	385.349335	370.600403	400.600403	0	0
170	0	2.57	369.652039	387.919342	373.17041	403.17041	0	0
171	0	2.57	370.475708	390.489349	321.619659	341.619659	48.86969	3225.39941
172	0	2.57	371.658264	393.059357	287.656189	307.656189	85.403168	7088.46289
173	1	2	373.616699	395.059357	294.887665	314.887665	80.171692	6654.25049
174	0	2.57	376.193726	397.629364	213.278091	233.278091	164.351273	16435.127
175	1	2	376.995911	399.629364	384.880432	414.880432	0	0
176	1	2	380.151215	401.629364	253.469177	273.469177	128.160187	12816.0186
177	0	2.57	382.969208	404.199371	347.225433	367.225433	36.973938	2440.27979
178	1	2	386.31131	406.199371	232.550385	252.550385	153.648987	15364.8984
179	1	2	387.771729	408.199371	393.450439	423.450439	0	0
180	0	2.57	390.975464	410.769379	396.020447	426.020447	0	0
181	0	2.57	392.087891	413.339386	356.909729	376.909729	36.429657	2404.35742
182	1	2	398.896454	415.339386	299.946198	319.946198	95.393188	7917.63477
183	0	2.57	406.38092	417.909393	296.334625	316.334625	101.574768	8430.70606
184	1	2	414.333771	419.909393	239.104889	259.104889	160.804504	16080.4502
185	1	2	415.277649	421.909393	243.298889	263.298889	158.610504	15861.0508
186	1	2	415.505981	423.909393	286.694794	306.694794	117.2146	9728.81152
187	0	2.57	417.226959	426.479401	231.812515	251.812515	174.666885	17466.6895
188	1	2	417.285553	428.479401	346.069244	366.069244	62.410156	5180.04297
189	0	2.57	419.573975	431.049408	318.482239	338.482239	92.567169	7683.0752
190	0	2.57	420.468903	433.619415	361.11319	381.11319	52.506226	3465.41089
191	0	2.57	422.343109	436.189423	339.936096	359.936096	76.253326	6329.02588
192	1	2	431.827698	438.189423	320.839935	340.839935	97.349487	8080.00732
193	1	2	433.15387	440.189423	276.297546	296.297546	143.891876	14389.1875

194	1	2	434.285522	442.189423	366.517883	386.517883	55.671539	3674.32153
195	1	2	434.661835	444.189423	429.440491	459.440491	0	0
196	0	2.57	437.783234	446.75943	355.287903	375.287903	71.471527	5932.13672
197	0	2.57	439.722504	449.329437	434.580505	464.580505	0	0
204	1	2	448.145355	451.329437	311.021362	331.021362	120.308075	12030.8076
198	1	2	443.190979	453.329437	436.580505	466.580505	0	0
199	0	2.57	443.36142	455.899445	381.107635	401.107635	54.791809	3616.25928
207	1	2	453.409515	457.899445	352.656464	372.656464	85.242981	7075.16748
200	0	2.57	443.538422	460.469452	346.10614	366.10614	94.363312	7832.15479
201	0	2.57	444.360901	463.039459	349.789703	369.789703	93.249756	7739.72949
202	0	2.57	446.02951	465.609467	446.860535	476.860535	0	0
204	1	2	448.145355	467.609467	311.021362	331.021362	136.588104	13658.8105
205	0	2.57	448.522369	470.179474	281.070801	301.070801	169.108673	16910.8672
207	1	2	453.409515	472.179474	352.656464	372.656464	99.52301	8260.41016
208	1	2	455.167114	474.179474	460.000549	490.000549	0	0
209	0	2.57	457.536011	476.749481	413.897552	433.897552	42.851929	2828.2273
210	1	2	459.571686	478.749481	362.889526	382.889526	95.859955	7956.37647
211	0	2.57	460.392487	481.319489	467.140564	497.140564	0	0
212	0	2.57	464.150665	483.889496	291.749695	311.749695	172.139801	17213.9805
213	1	2	466.417023	485.889496	325.109772	345.109772	140.779724	14077.9727
214	1	2	468.418884	487.889496	353.422852	373.422852	114.466644	9500.73145
215	1	2	469.03775	489.889496	413.604065	433.604065	56.285431	3714.83838
216	0	2.57	472.932373	492.459503	421.963562	441.963562	50.495941	3332.73218
217	0	2.57	473.211823	495.02951	418.412781	438.412781	56.61673	3736.7041
218	0	2.57	475.518433	497.599518	327.888611	347.888611	149.710907	14971.0908
219	0	2.57	476.716187	500.169525	415.25177	435.25177	64.917755	5388.17383
220	0	2.57	482.683075	502.739532	488.560608	518.560608	0	0
221	0	2.57	484.118408	505.30954	491.130615	521.130615	0	0
222	0	2.57	485.329956	507.879547	493.700623	523.700623	0	0
234	1	2	508.512329	510.512329	425.268829	445.268829	65.2435	5415.21045
223	0	2.57	486.536682	513.082336	433.579071	453.579071	59.503265	3927.21558
224	0	2.57	487.167877	515.652344	444.364929	464.364929	51.287415	3384.96924
225	1	2	488.006348	517.652344	462.843994	482.843994	34.80835	2297.35107
240	0	2.57	515.375061	520.222351	376.2146	396.2146	124.007751	12400.7754
226	0	2.57	488.257416	522.792358	410.389862	430.389862	92.402496	7669.40723
227	1	2	489.784119	524.792358	437.669128	457.669128	67.12323	5571.22803
228	0	2.57	491.100769	527.362366	507.980652	537.980652	0	0
229	1	2	492.170227	529.362366	438.771423	458.771423	70.590942	5859.04834
230	1	2	494.286926	531.362366	416.627625	436.627625	94.734741	7862.9834
231	1	2	496.466888	533.362366	414.414703	434.414703	98.947662	8212.65625
232	0	2.57	499.742432	535.932373	403.634216	423.634216	112.298157	9320.74707
234	1	2	508.512329	537.932373	425.268829	445.268829	92.663544	7691.07422
235	0	2.57	510.605103	540.50238	402.320496	422.320496	118.181885	9809.09668
236	0	2.57	511.24762	543.072388	435.17453	455.17453	87.897858	7295.52197
237	1	2	512.651917	545.072388	441.575714	461.575714	83.496674	6930.22412
238	1	2	513.336853	547.072388	422.839172	442.839172	104.233215	8651.35645
240	0	2.57	515.375061	549.642395	376.2146	396.2146	153.427795	15342.7793
241	0	2.57	516.168884	552.212402	536.830688	566.830688	0	0
242	1	2	516.191284	554.212402	538.830688	568.830688	0	0
243	0	2.57	516.498413	556.78241	473.458221	493.458221	63.324188	5255.90772
244	1	2	517.005371	558.78241	401.186188	421.186188	137.596222	13759.6221
245	1	2	518.830627	560.78241	481.572754	501.572754	59.209656	3907.8374
246	0	2.57	519.814148	563.352417	492.708008	512.708008	50.644409	3342.53101
247	0	2.57	520.968872	565.922424	437.167053	457.167053	108.755371	9026.69531
248	1	2	521.418091	567.922424	436.472748	456.472748	111.449677	9250.32324
249	1	2	524.700195	569.922424	490.830658	510.830658	59.091766	3900.05664
250	1	2	529.817749	571.922424	556.54071	586.54071	0	0
251	1	2	529.834106	573.922424	510.067261	530.067261	43.855164	2894.44092
252	0	2.57	530.470886	576.492432	398.495544	418.495544	157.996887	15799.6885
253	0	2.57	532.684082	579.062439	401.533569	421.533569	157.52887	15752.8867
254	0	2.57	532.84375	581.632446	480.046265	500.046265	81.586182	6771.65332
255	1	2	533.387695	583.632446	510.05542	530.05542	53.577026	3536.08374
256	0	2.57	536.381165	586.202454	503.380188	523.380188	62.822266	5214.24805
257	0	2.57	539.599365	588.772461	488.501831	508.501831	80.27063	6662.4624
258	0	2.57	543.647827	591.342468	533.366211	553.366211	37.976257	2506.43311
259	0	2.57	544.07782	593.912476	578.530762	608.530762	0	0
260	1	2	545.293274	595.912476	538.657043	558.657043	37.255432	2458.8584

261	0	2.57	545.968933	598.482483	534.118958	554.118958	44.363525	2927.99268
262	0	2.57	546.622437	601.05249	585.670776	615.670776	0	0
263	1	2	548.930237	603.05249	487.787048	507.787048	95.265442	7907.03174
264	0	2.57	549.07019	605.622498	473.759735	493.759735	111.862762	9284.60938
265	0	2.57	551.051575	608.192505	592.810791	622.810791	0	0
266	1	2	556.020813	610.192505	594.810791	624.810791	0	0
267	1	2	556.771606	612.192505	528.888428	548.888428	63.304077	5254.23828
287	0	2.57	610.369324	614.762512	516.500122	536.500122	78.26239	6495.77832
268	1	2	561.137207	616.762512	529.069946	549.069946	67.692566	5618.48291
269	1	2	561.473572	618.762512	546.85791	566.85791	51.904602	3425.70361
270	0	2.57	562.078674	621.33252	488.162415	508.162415	113.170105	9393.11914
271	0	2.57	573.982666	623.902527	460.100189	480.100189	143.802338	14380.2334
272	0	2.57	580.689514	626.472534	562.319214	582.319214	44.15332	2914.11914
273	0	2.57	581.177368	629.042542	611.09082	641.09082	0	0
274	1	2	584.345032	631.042542	613.09082	643.09082	0	0
275	1	2	585.058838	633.042542	615.09082	645.09082	0	0
276	0	2.57	586.544861	635.612549	617.660828	647.660828	0	0
297	1	2	634.138855	637.612549	666.500916	696.500916	-28.888367	1906.6322
277	1	2	587.621643	639.612549	580.333618	600.333618	39.278931	2592.40942
278	1	2	588.59491	641.612549	455.605255	475.605255	166.007294	16600.7285
279	0	2.57	589.083374	644.182556	624.230835	654.230835	0	0
280	0	2.57	589.246338	646.752563	626.800842	656.800842	0	0
281	1	2	597.652283	648.752563	533.590149	553.590149	95.162415	7898.48047
300	0	2.57	649.728943	652.29895	499.978516	519.978516	132.320435	13232.043
282	0	2.57	602.520264	654.868958	494.548889	514.548889	140.320068	14032.0068
283	1	2	604.789063	656.868958	633.37085	663.37085	0	0
284	1	2	604.862915	658.868958	468.600525	488.600525	170.268433	17026.8438
285	0	2.57	609.100342	661.438965	473.164948	493.164948	168.274017	16827.4023
303	0	2.57	652.50354	664.008972	508.809021	528.809021	135.199951	13519.9951
287	0	2.57	610.369324	666.578979	516.500122	536.500122	130.078857	13007.8857
288	0	2.57	615.561218	669.148987	491.524963	511.524963	157.624023	15762.4023
289	0	2.57	618.631348	671.718994	475.145203	495.145203	176.573792	17657.3789
290	0	2.57	619.775635	674.289001	563.282593	583.282593	91.006409	7553.53174
291	0	2.57	620.137817	676.859009	607.374329	627.374329	49.48468	3265.98877
292	1	2	622.587891	678.859009	492.403748	512.403748	166.455261	16645.5254
293	1	2	623.227173	680.859009	526.674255	546.674255	134.184753	13418.4756
294	0	2.57	629.215942	683.429016	659.930908	689.930908	0	0
295	1	2	630.734863	685.429016	590.250061	610.250061	75.178955	6239.85352
297	1	2	634.138855	687.429016	666.500916	696.500916	0	0
298	0	2.57	642.269287	689.999023	547.956604	567.956604	122.042419	12204.2422
300	0	2.57	649.728943	692.569031	499.978516	519.978516	172.590515	17259.0508
301	0	2.57	651.643921	695.139038	563.96405	583.96405	111.174988	9227.52441
303	0	2.57	652.50354	697.709045	508.809021	528.809021	168.900024	16890.002
304	1	2	656.947815	699.709045	604.736206	624.736206	74.972839	6222.74561
305	0	2.57	657.017212	702.279053	685.920959	715.920959	0	0
306	1	2	657.044189	704.279053	625.053528	645.053528	59.225525	3908.88477
307	0	2.57	659.655273	706.84906	690.490967	720.490967	0	0
308	1	2	660.158325	708.84906	589.915344	609.915344	98.933716	8211.49805
309	0	2.57	661.551453	711.419067	640.210938	660.210938	51.20813	3379.73657
310	0	2.57	662.835266	713.989075	589.696411	609.696411	104.292664	8656.29102
311	1	2	663.345337	715.989075	699.630981	729.630981	0	0
312	0	2.57	666.63385	718.559082	582.847351	602.847351	115.711731	9604.07324
313	1	2	677.466675	720.559082	650.229492	670.229492	50.32959	3321.75293
314	1	2	681.416992	722.559082	589.775085	609.775085	112.783997	9361.07129
315	1	2	682.722412	724.559082	554.42334	574.42334	150.135742	15013.5742
316	0	2.57	684.250183	727.129089	710.770996	740.770996	0	0
331	1	2	724.037109	729.129089	618.480957	638.480957	90.648132	7523.79492
317	0	2.57	693.455383	731.699097	558.28009	578.28009	153.419006	15341.9004
318	0	2.57	693.730347	734.269104	715.911011	745.911011	0	0
319	0	2.57	694.190735	736.839111	628.973877	648.973877	87.865234	7292.81445
320	0	2.57	694.813843	739.409119	575.87439	595.87439	143.534729	14353.4727
321	1	2	699.992432	741.409119	639.643188	659.643188	81.76593	6786.57227
322	1	2	702.180969	743.409119	653.616394	673.616394	69.792725	5792.7959
323	1	2	704.88562	745.409119	668.512146	688.512146	56.896973	3755.2002
324	0	2.57	706.964355	747.979126	573.694641	593.694641	154.284485	15428.4482
341	1	2	745.339233	749.979126	711.920532	731.920532	18.058594	1191.86719
325	1	2	708.621216	751.979126	660.616455	680.616455	71.362671	5923.10156

326	1	2	709.460632	753.979126	614.492859	634.492859	119.486267	9917.36035
327	0	2.57	713.636414	756.549133	672.834167	692.834167	63.714966	5288.34229
348	1	2	755.558838	758.549133	677.597412	697.597412	60.951721	5058.99268
328	0	2.57	715.421448	761.119141	681.782104	701.782104	59.337036	3916.24439
329	0	2.57	717.119812	763.689148	741.331055	771.331055	0	0
348	1	2	755.558838	765.689148	677.597412	697.597412	68.091736	5651.61426
331	1	2	724.037109	767.689148	618.480957	638.480957	129.208191	12920.8193
332	1	2	726.383606	769.689148	629.575012	649.575012	120.114136	12011.4141
333	1	2	730.068787	771.689148	677.300781	697.300781	74.388367	6174.23438
334	1	2	731.464661	773.689148	751.331055	781.331055	0	0
335	0	2.57	734.308472	776.259155	627.618896	647.618896	128.640259	12864.0254
336	0	2.57	739.038818	778.829163	713.206177	733.206177	45.622986	3011.11719
359	0	2.57	775.645996	781.39917	807.601135	837.601135	-26.201965	1729.32971
337	1	2	739.706482	783.39917	758.471069	788.471069	0	0
338	1	2	739.820801	785.39917	760.471069	790.471069	0	0
366	1	2	784.276001	787.39917	670.874756	690.874756	96.524414	8011.52637
361	1	2	778.356873	789.39917	742.347046	762.347046	27.052124	1785.44019
339	1	2	741.763489	791.39917	762.471069	792.471069	0	0
341	1	2	745.339233	793.39917	711.920532	731.920532	61.478638	5102.72705
342	0	2.57	747.622009	795.969177	666.490356	686.490356	109.478821	9086.74219
343	0	2.57	749.44696	798.539185	669.433716	689.433716	109.105469	9055.75391
344	0	2.57	749.509399	801.109192	610.362793	630.362793	170.746399	17074.6406
376	0	2.57	806.13147	803.679199	846.161194	876.161194	-42.481995	2803.81152
345	1	2	750.010193	805.679199	776.181091	806.181091	0	0
348	1	2	755.558838	807.679199	677.597412	697.597412	110.081787	9136.78809
376	0	2.57	806.13147	810.249207	846.161194	876.161194	-35.911987	2370.19116
349	0	2.57	759.503235	812.819214	705.257507	725.257507	87.561707	7267.62158
350	0	2.57	760.920959	815.389221	707.203125	727.203125	88.186096	7319.4458
351	0	2.57	762.77948	817.959229	751.497925	771.497925	46.461304	3066.44605
352	1	2	763.711426	819.959229	740.56842	760.56842	59.390808	3919.79346
353	0	2.57	763.855896	822.529236	724.98407	744.98407	77.545166	6436.24902
354	0	2.57	766.860657	825.099243	739.501953	759.501953	65.59729	5444.5752
355	1	2	766.968689	827.099243	699.269714	719.269714	107.829529	8949.85059
356	1	2	770.402527	829.099243	715.340759	735.340759	93.758484	7781.9541
383	0	2.57	822.906738	831.66925	799.93988	819.93988	11.72937	774.138428
357	1	2	771.450745	833.66925	726.583191	746.583191	87.08606	7228.14307
359	0	2.57	775.645996	836.239258	807.601135	837.601135	0	0
361	1	2	778.356873	838.239258	742.347046	762.347046	75.892212	6299.05371
362	1	2	779.330261	840.239258	774.169922	794.169922	46.069336	3040.57617
363	1	2	780.615662	842.239258	700.628418	720.628418	121.61084	12161.084
364	0	2.57	781.757141	844.809265	671.999695	691.999695	152.80957	15280.957
366	1	2	784.276001	846.809265	670.874756	690.874756	155.934509	15593.4512
367	0	2.57	786.953613	849.379272	760.339722	780.339722	69.039551	5730.28272
368	1	2	787.166016	851.379272	682.063843	702.063843	149.31543	14931.543
369	0	2.57	789.666992	853.94928	762.17981	782.17981	71.76947	5956.86621
370	1	2	789.865051	855.94928	831.881165	861.881165	0	0
371	0	2.57	790.261108	858.519287	834.451172	864.451172	0	0
372	1	2	791.991455	860.519287	703.640503	723.640503	136.878784	13687.8789
373	0	2.57	795.820984	863.089294	740.792114	760.792114	102.29718	8490.66602
376	0	2.57	806.13147	865.659302	846.161194	876.161194	0	0
377	1	2	808.512451	867.659302	713.50708	733.50708	134.152222	13415.2227
378	0	2.57	810.625549	870.229309	850.731201	880.731201	0	0
379	1	2	810.845764	872.229309	756.114746	776.114746	96.114563	7977.50879
380	0	2.57	821.860779	874.799316	808.686829	828.686829	46.112488	3043.42432
381	1	2	821.964355	876.799316	772.078857	792.078857	84.720459	7031.79785
383	0	2.57	822.906738	879.369324	799.93988	819.93988	59.429443	3922.34326
384	0	2.57	825.61676	881.939331	864.441223	894.441223	0	0
385	1	2	827.127625	883.939331	818.058472	838.058472	45.880859	3028.13672
386	1	2	827.172913	885.939331	868.441223	898.441223	0	0
387	1	2	832.856567	887.939331	716.160156	736.160156	151.779175	15177.918
388	1	2	834.882996	889.939331	758.9328	778.9328	111.006531	9213.54199
414	0	2.57	889.648315	892.509338	843.712219	863.712219	28.797119	1900.60986
389	1	2	838.564575	894.509338	825.423035	845.423035	49.086304	3239.69605
390	1	2	838.654663	896.509338	772.714355	792.714355	103.794983	8614.9834
391	0	2.57	838.895264	899.079346	879.01123	909.01123	0	0
392	1	2	841.970581	901.079346	781.054077	801.054077	100.025269	8302.09766
393	1	2	844.221741	903.079346	827.447876	847.447876	55.63147	3671.677

394	1	2	846.275513	905.079346	710.559143	730.559143	174.520203	17452.0195
395	0	2.57	848.790222	907.649353	887.581238	917.581238	0	0
396	1	2	852.3797	909.649353	889.581238	919.581238	0	0
397	0	2.57	854.894287	912.21936	812.507141	832.507141	79.712219	6616.11426
398	1	2	856.559204	914.21936	894.151245	924.151245	0	0
399	0	2.57	859.484009	916.789368	896.721252	926.721252	0	0
400	0	2.57	859.955322	919.359375	815.663208	835.663208	83.696167	6946.78174
401	1	2	861.307129	921.359375	838.833923	858.833923	62.525452	5189.61231
402	0	2.57	862.14386	923.929382	849.871094	869.871094	54.058289	3567.84717
403	0	2.57	862.774414	926.49939	906.431274	936.431274	0	0
404	0	2.57	863.852173	929.069397	814.264221	834.264221	94.805176	7868.82959
405	1	2	866.35907	931.069397	745.079041	765.079041	165.990356	16599.0352
406	1	2	867.609375	933.069397	804.989075	824.989075	108.080322	8970.66699
407	0	2.57	871.829956	935.639404	864.983093	884.983093	50.656311	3343.31641
408	0	2.57	871.83313	938.209412	838.281982	858.281982	79.927429	6633.97656
409	0	2.57	873.107605	940.779419	789.776794	809.776794	131.002625	13100.2627
410	0	2.57	874.945862	943.349426	923.281311	953.281311	0	0
411	1	2	879.819702	945.349426	759.75769	779.75769	165.591736	16559.1738
412	1	2	881.073608	947.349426	927.281311	957.281311	0	0
414	0	2.57	889.648315	949.919434	843.712219	863.712219	86.207214	7155.19873
415	1	2	890.289124	951.919434	799.285522	819.285522	132.633911	13263.3906
416	1	2	893.118774	953.919434	936.421326	966.421326	0	0
417	0	2.57	897.031311	956.489441	892.52063	912.52063	43.968811	2901.94141
418	1	2	900.31189	958.489441	823.402283	843.402283	115.087158	9552.23438
419	1	2	900.973083	960.489441	796.318787	816.318787	144.170654	14417.0654
445	0	2.57	961.088562	963.658569	998.981445	1028.98145	-35.322876	2331.30981
420	1	2	902.095581	965.658569	865.196594	885.196594	80.461975	6678.34375
448	1	2	963.207886	967.658569	936.495239	956.495239	11.16333	736.779785
421	1	2	902.67865	969.658569	946.991333	976.991333	0	0
422	1	2	903.317383	971.658569	904.379089	924.379089	47.27948	3120.4458
423	1	2	909.896179	973.658569	783.861206	803.861206	169.797363	16979.7363
424	0	2.57	910.652405	976.228577	953.56134	983.56134	0	0
425	1	2	911.696716	978.228577	824.998474	844.998474	133.230103	13323.0098
426	1	2	913.905762	980.228577	813.194702	833.194702	147.033875	14703.3877
427	1	2	915.828735	982.228577	959.56134	989.56134	0	0
428	0	2.57	916.89325	984.798584	913.476501	933.476501	51.322083	3387.25732
459	1	2	985.239319	987.239319	981.528381	1001.52838	0	0
429	0	2.57	917.273621	989.809326	964.701355	994.701355	0	0
430	0	2.57	920.349609	992.379333	899.147827	919.147827	73.231506	6078.21484
431	1	2	921.0448	994.379333	883.46936	903.46936	90.909973	7545.52783
432	1	2	921.217102	996.379333	852.873047	872.873047	123.506287	12350.6289
433	1	2	921.573547	998.379333	814.860291	834.860291	163.519043	16351.9043
434	1	2	929.815918	1000.37933	821.570862	841.570862	158.808472	15880.8477
435	1	2	929.842041	1002.37933	873.091553	893.091553	109.287781	9070.88574
436	1	2	931.186829	1004.37933	825.879028	845.879028	158.500305	15850.0303
437	1	2	932.948547	1006.37933	858.342285	878.342285	128.037048	12803.7051
438	1	2	940.149841	1008.37933	983.271362	1013.27136	0	0
439	1	2	942.088745	1010.37933	985.271362	1015.27136	0	0
440	0	2.57	949.324707	1012.94934	896.702881	916.702881	96.24646	7988.45606
441	1	2	952.483276	1014.94934	841.795898	861.795898	153.153442	15315.3438
442	1	2	957.70874	1016.94934	927.580505	947.580505	69.368835	5757.61328
443	0	2.57	958.667847	1019.51935	896.327454	916.327454	103.191895	8564.92773
445	0	2.57	961.088562	1022.08936	998.981445	1028.98145	0	0
446	1	2	961.217102	1024.08936	948.051453	968.051453	56.037903	3698.50147
448	1	2	963.207886	1026.08936	936.495239	956.495239	69.594116	5776.31152
449	1	2	963.470154	1028.08936	907.927307	927.927307	100.162048	8313.4502
450	0	2.57	964.122253	1030.6593	885.690369	905.690369	124.968933	12496.8936
451	0	2.57	971.105225	1033.22925	1012.12134	1042.12134	0	0
452	1	2	976.566406	1035.22925	892.74939	912.74939	122.479858	12247.9863
453	0	2.57	979.141663	1037.79919	867.327332	887.327332	150.471863	15047.1865
454	0	2.57	980.632874	1040.36914	862.101074	882.101074	158.268066	15826.8066
455	1	2	981.360718	1042.36914	1021.26123	1051.26123	0	0
456	0	2.57	982.027832	1044.93909	959.452087	979.452087	65.487	5435.4209
457	1	2	982.831116	1046.93909	982.389526	1002.38953	44.549561	2940.271
459	1	2	985.239319	1048.93909	981.528381	1001.52838	47.410706	3129.10645
460	0	2.57	985.43457	1051.50903	1032.97107	1062.97107	0	0
461	0	2.57	989.053894	1054.07898	1035.54102	1065.54102	0	0

462	0	2.57	997.371826	1056.64893	985.866211	1005.86621	50.782715	3351.65918
463	0	2.57	1001.57343	1059.21887	1040.68091	1070.68091	0	0
464	1	2	1010.64063	1061.21887	991.615601	1011.6156	49.603271	3273.81592
465	1	2	1016.07135	1063.21887	874.985474	894.985474	168.233398	16823.3398
466	0	2.57	1016.37335	1065.78882	985.397278	1005.39728	60.391541	5012.49805
467	1	2	1017.50678	1067.78882	889.529724	909.529724	158.259094	15825.9092
468	1	2	1017.67523	1069.78882	1012.75415	1032.75415	37.034668	2444.28809
469	0	2.57	1019.20056	1072.35877	926.219299	946.219299	126.139465	12613.9463
470	0	2.57	1022.57324	1074.92871	975.320862	995.320862	79.607849	6607.45166
471	0	2.57	1024.82764	1077.49866	916.67749	936.67749	140.821167	14082.1172
472	1	2	1027.33374	1079.49866	927.390869	947.390869	132.107788	13210.7793
473	1	2	1028.75794	1081.49866	934.875793	954.875793	126.622864	12662.2861
474	1	2	1035.30554	1083.49866	1064.96069	1094.96069	0	0
475	1	2	1036.87293	1085.49866	930.562683	950.562683	134.935974	13493.5977
492	0	2.57	1083.99121	1088.0686	1008.87402	1028.87402	59.19458	3906.84229
476	1	2	1038.6095	1090.0686	1027.01672	1047.01672	43.05188	2841.42407
477	0	2.57	1038.86304	1092.63855	925.466248	945.466248	147.172302	14717.2305
478	0	2.57	1038.97986	1095.2085	1011.72131	1031.72131	63.487183	5269.43604
479	0	2.57	1047.77466	1097.77844	1076.67053	1106.67053	0	0
480	0	2.57	1056.28784	1100.34839	961.707092	981.707092	118.641296	9847.22754
481	1	2	1056.56201	1102.34839	985.554749	1005.55475	96.79364	8033.87207
482	1	2	1059.44067	1104.34839	984.015991	1004.01599	100.332397	8327.58887
483	1	2	1060.14148	1106.34839	1043.75256	1063.75256	42.595825	2811.32446
484	1	2	1061.07568	1108.34839	992.645203	1012.6452	95.703186	7943.36426
485	0	2.57	1062.66809	1110.91834	1039.38709	1059.38709	51.53125	3401.0625
486	0	2.57	1063.3645	1113.48828	1021.37585	1041.37585	72.112427	5985.33154
487	1	2	1063.79248	1115.48828	1094.38037	1124.38037	0	0
503	0	2.57	1113.77539	1118.05823	1130.37	1160.37	-12.311768	812.57666
488	1	2	1063.95874	1120.05823	997.690857	1017.69086	102.367371	8496.49219
489	0	2.57	1071.59656	1122.62817	927.369141	947.369141	175.259033	17525.9023
490	0	2.57	1075.75037	1125.19812	1101.52026	1131.52026	0	0
492	0	2.57	1083.99121	1127.76807	1008.87402	1028.87402	98.894043	8208.20508
493	0	2.57	1087.25342	1130.33801	1108.66016	1138.66016	0	0
494	1	2	1088.75855	1132.33801	988.367798	1008.3678	123.970215	12397.0215
495	1	2	1090.6178	1134.33801	1009.95984	1029.95984	104.378174	8663.38867
496	1	2	1090.92444	1136.33801	1062.95068	1082.95068	53.387329	3523.56372
497	0	2.57	1091.61646	1138.90796	1045.46704	1065.46704	73.440918	6095.59619
498	0	2.57	1097.88098	1141.47791	1046.07434	1066.07434	75.403564	6258.49609
499	1	2	1099.8407	1143.47791	957.067871	977.067871	166.410034	16641.0039
500	1	2	1109.78833	1145.47791	973.136963	993.136963	152.340942	15234.0938
501	1	2	1112.85022	1147.47791	1125.80005	1155.80005	0	0
503	0	2.57	1113.77539	1150.04785	1130.37	1160.37	0	0
504	0	2.57	1118.31018	1152.6178	1132.93994	1162.93994	0	0
505	0	2.57	1122.49182	1155.18774	1071.18994	1091.18994	63.997803	5311.81738
506	0	2.57	1124.18323	1157.75769	973.914612	993.914612	163.843079	16384.3086
507	0	2.57	1128.08936	1160.32764	1095.521	1115.521	44.806641	2957.23828
508	1	2	1130.25806	1162.32764	1098.81873	1118.81873	43.508911	2871.58814
509	0	2.57	1131.61987	1164.89758	1077.15662	1097.15662	67.740967	5622.5
510	0	2.57	1134.67542	1167.46753	992.178955	1012.17896	155.288574	15528.8574
511	0	2.57	1135.41309	1170.03748	1070.78003	1090.78003	79.257446	6578.36816
512	0	2.57	1135.43347	1172.60742	1152.92957	1182.92957	0	0
513	1	2	1141.1626	1174.60742	1110.76148	1130.76148	43.845947	2893.83252
514	1	2	1144.53003	1176.60742	1118.26013	1138.26013	38.34729	2530.92114
515	0	2.57	1144.6438	1179.17737	1031.02039	1051.02039	128.156982	12815.6982
516	1	2	1145.96423	1181.17737	1037.43359	1057.43359	123.743774	12374.377
517	1	2	1146.03308	1183.17737	1040.91821	1060.91821	122.259155	12225.916
531	1	2	1178.87915	1185.17737	1155.08862	1175.08862	10.088745	665.857178
518	1	2	1147.68921	1187.17737	1019.07788	1039.07788	148.099487	14809.9492
519	0	2.57	1153.04334	1189.74731	1168.06946	1198.06946	0	0
533	1	2	1186.72315	1191.74731	1158.88001	1178.88001	12.86731	849.242432
520	1	2	1156.45996	1193.74731	1119.3346	1139.3346	54.41272	3591.2395
521	1	2	1156.86304	1195.74731	1134.22778	1154.22778	41.519531	2740.28906
522	0	2.57	1163.83154	1198.31726	1115.31567	1135.31567	63.001587	5229.13184
523	0	2.57	1164.89478	1200.88721	1109.64844	1129.64844	71.23877	5912.81787
524	1	2	1166.75501	1202.88721	1179.20935	1209.20935	0	0
525	1	2	1167.12219	1204.88721	1073.06751	1093.06751	111.819702	9281.03516
526	0	2.57	1172.98364	1207.45715	1104.75781	1124.75781	82.699341	6864.04541

527	1	2	1175.21948	1209.45715	1129.5	1149.5	59.957153	3957.17212
528	0	2.57	1175.69458	1212.0271	1067.14905	1087.14905	124.878052	12487.8047
529	0	2.57	1175.82983	1214.59705	1059.64294	1079.64294	134.954102	13495.4102
531	1	2	1178.87915	1216.59705	1155.08862	1175.08862	41.508423	2739.55591
533	1	2	1186.72315	1218.59705	1158.88001	1178.88001	39.717041	2621.32471
534	0	2.57	1187.76856	1221.16699	1126.51245	1146.51245	74.654541	6196.32715
535	0	2.57	1188.06909	1223.73694	1098.63879	1118.63879	105.098145	8723.14648
536	1	2	1189.58569	1225.73694	1051.73914	1071.73914	153.997803	15399.7803
537	1	2	1189.63501	1227.73694	1084.8396	1104.8396	122.897339	12289.7344
538	1	2	1195.53845	1229.73694	1210.05908	1240.05908	0	0
539	1	2	1196.10938	1231.73694	1050.58801	1070.58801	161.148926	16114.8926
540	0	2.57	1199.53516	1234.30689	1074.99377	1094.99377	139.31311	13931.3105
541	1	2	1199.71936	1236.30689	1092.76794	1112.76794	123.53894	12353.8945
542	0	2.57	1200.50647	1238.87683	1051.48071	1071.48071	167.396118	16739.6113
543	1	2	1204.04273	1240.87683	1059.26367	1079.26367	161.613159	16161.3164
544	0	2.57	1207.48132	1243.44678	1223.76892	1253.76892	0	0
545	1	2	1214.10938	1245.44678	1170.8064	1190.8064	54.640381	3606.26514
546	1	2	1216.79663	1247.44678	1073.09876	1093.09876	154.348022	15434.8027
547	1	2	1218.62842	1249.44678	1179.96729	1199.96729	49.479492	3265.64648
548	0	2.57	1219.42749	1252.01672	1232.33887	1262.33887	0	0
549	1	2	1229.77893	1254.01672	1234.33887	1264.33887	0	0
550	0	2.57	1230.41699	1256.58667	1236.90881	1266.90881	0	0
551	0	2.57	1231.98865	1259.15662	1154.16736	1174.16736	84.989258	7054.1084
552	0	2.57	1236.55615	1261.72656	1178.59375	1198.59375	63.132813	5240.02344
553	1	2	1239.30017	1263.72656	1128.80298	1148.80298	114.923584	9538.65723
554	1	2	1240.91577	1265.72656	1145.99365	1165.99365	99.73291	8277.83203
555	1	2	1243.06836	1267.72656	1107.13171	1127.13171	140.594849	14059.4844
556	1	2	1243.75806	1269.72656	1202.76856	1222.76856	46.958008	3099.22852
557	1	2	1244.99048	1271.72656	1201.38721	1221.38721	50.339355	3322.39746
558	0	2.57	1247.39673	1274.29651	1131.37	1151.37	122.926514	12292.6514
559	1	2	1249.04126	1276.29651	1211.00781	1231.00781	45.288696	2989.05396
560	0	2.57	1250.26807	1278.86646	1119.7544	1139.7544	139.112061	13911.2061
561	0	2.57	1252.41235	1281.4364	1212.10291	1232.10291	49.333496	3256.01074
562	0	2.57	1253.7273	1284.00635	1125.29517	1145.29517	138.711182	13871.1182
563	0	2.57	1256.04639	1286.57629	1266.89844	1296.89844	0	0
564	0	2.57	1258.09033	1289.14624	1187.09155	1207.09155	82.054688	6810.53906
565	0	2.57	1263.10034	1291.71619	1272.03833	1302.03833	0	0
577	0	2.57	1291.41406	1294.28613	1301.16785	1331.16785	-6.881714	454.193115
566	1	2	1264.48645	1296.28613	1274.03833	1304.03833	0	0
567	0	2.57	1266.1156	1298.85608	1181.8949	1201.8949	96.961182	8047.77832
568	0	2.57	1266.68494	1301.42603	1181.45654	1201.45654	99.969482	8297.4668
569	0	2.57	1266.8783	1303.99597	1211.74304	1231.74304	72.25293	5996.99316
579	1	2	1293.5083	1305.99597	1216.04773	1236.04773	69.948242	5805.7041
570	0	2.57	1267.36304	1308.56592	1227.45276	1247.45276	61.113159	5072.39209
571	1	2	1269.20264	1310.56592	1286.31812	1316.31812	0	0
572	1	2	1269.6095	1312.56592	1288.31812	1318.31812	0	0
573	0	2.57	1282.78137	1315.13586	1231.9668	1251.9668	63.169067	5243.03272
574	0	2.57	1283.3573	1317.70581	1178.83923	1198.83923	118.866577	9865.92578
575	0	2.57	1288.99085	1320.27576	1242.39185	1262.39185	57.883911	3820.33814
577	0	2.57	1291.41406	1322.8457	1301.16785	1331.16785	0	0
579	1	2	1293.5083	1324.8457	1216.04773	1236.04773	88.797974	7370.23193
580	0	2.57	1294.94885	1327.41565	1244.97046	1264.97046	62.44519	5182.95068
593	0	2.57	1319.58264	1329.9856	1339.43726	1369.43726	-9.45166	623.80957
581	0	2.57	1297.64172	1332.55554	1190.44788	1210.44788	122.107666	12210.7666
597	0	2.57	1329.3429	1335.12549	1197.27673	1217.27673	117.848755	9781.44629
582	0	2.57	1300.4032	1337.69544	1313.44763	1343.44763	0	0
583	0	2.57	1302.31165	1340.26538	1262.64197	1282.64197	57.623413	3803.14526
605	1	2	1339.81519	1342.26538	1367.42688	1397.42688	-25.161499	1660.65894
584	0	2.57	1302.77698	1344.83533	1154.42236	1174.42236	170.412964	17041.2969
585	1	2	1303.57922	1346.83533	1225.87415	1245.87415	100.961182	8379.77832
586	0	2.57	1309.58203	1349.40527	1202.84131	1222.84131	126.563965	12656.3965
587	1	2	1311.93579	1351.40527	1325.15747	1355.15747	0	0
588	0	2.57	1313.20227	1353.97522	1327.72742	1357.72742	0	0
589	1	2	1314.68506	1355.97522	1262.96729	1282.96729	73.007935	6059.65869
590	0	2.57	1316.30786	1358.54517	1289.70288	1309.70288	48.842285	3223.59082
591	0	2.57	1316.69385	1361.11511	1274.31189	1294.31189	66.803223	5544.66748
593	0	2.57	1319.58264	1363.68506	1339.43726	1369.43726	0	0

594	0	2.57	1324.82959	1366.25501	1301.84631	1321.84631	44.408691	2930.97363
595	1	2	1326.89111	1368.25501	1344.0072	1374.0072	0	0
597	0	2.57	1329.3429	1370.82495	1197.27673	1217.27673	153.548218	15354.8223
598	1	2	1329.56238	1372.82495	1188.89539	1208.89539	163.929565	16392.957
599	0	2.57	1332.64453	1375.3949	1219.54553	1239.54553	135.849365	13584.9365
600	0	2.57	1332.82068	1377.96484	1201.31946	1221.31946	156.645386	15664.5391
601	0	2.57	1333.30249	1380.53479	1358.28699	1388.28699	0	0
602	0	2.57	1335.76904	1383.10474	1312.05689	1332.05689	51.047852	3369.1582
603	0	2.57	1336.1084	1385.67468	1325.17175	1345.17175	40.50293	2673.19336
605	1	2	1339.81519	1387.67468	1367.42688	1397.42688	0	0
606	1	2	1340.49707	1389.67468	1369.42688	1399.42688	0	0
607	0	2.57	1343.03015	1392.24463	1371.99683	1401.99683	0	0
608	0	2.57	1343.27576	1394.81458	1223.94104	1243.94104	150.873535	15087.3535
609	1	2	1345.84558	1396.81458	1327.09351	1347.09351	49.721069	3281.59058
610	0	2.57	1346.40808	1399.38452	1293.33472	1313.33472	86.049805	7142.13379
611	0	2.57	1349.40808	1401.95447	1381.70667	1411.70667	0	0
612	0	2.57	1352.422	1404.52441	1257.93481	1277.93481	126.5896	12658.96
613	0	2.57	1354.01904	1407.09436	1348.78101	1368.78101	38.313354	2528.6814
614	1	2	1354.30457	1409.09436	1271.22974	1291.22974	117.864624	9782.76367
615	1	2	1357.43433	1411.09436	1335.3396	1355.3396	55.754761	3679.81421
616	1	2	1357.5874	1413.09436	1327.0625	1347.0625	66.03186	5480.64453
617	0	2.57	1358.29981	1415.66431	1305.95508	1325.95508	89.709229	7445.86621
618	1	2	1361.08606	1417.66431	1328.22315	1348.22315	69.441162	5763.61621
619	0	2.57	1361.83301	1420.23425	1354.62573	1374.62573	45.608521	3010.16235
620	1	2	1365.50916	1422.23425	1352.25781	1372.25781	49.97644	3298.44507
621	0	2.57	1365.62158	1424.8042	1347.50171	1367.50171	57.30249	3781.96436
622	0	2.57	1365.67542	1427.37415	1317.87048	1337.87048	89.503662	7428.80371
623	1	2	1366.86011	1429.37415	1409.12634	1439.12634	0	0
624	1	2	1367.64172	1431.37415	1253.76563	1273.76563	157.608521	15760.8516
625	0	2.57	1375.49976	1433.94409	1413.69629	1443.69629	0	0
626	0	2.57	1375.97107	1436.51404	1416.26624	1446.26624	0	0
627	0	2.57	1376.60339	1439.08398	1354.81152	1374.81152	64.272461	5334.61426
628	0	2.57	1378.73877	1441.65393	1421.40613	1451.40613	0	0
629	1	2	1382.27441	1443.65393	1423.40613	1453.40613	0	0
630	1	2	1386.30237	1445.65393	1313.39941	1333.39941	112.254517	9317.125
	aterrizajes	**317**					**40502.4969**	**3611543.62**
	despegues	**313**			**valor esperado**		**64.2896777**	**5732.60893**
					varianza muestral		**3042.1138**	**29116028.6**
					desviación estándar		**55.1553606**	**5395.92704**